DRIVE: A Distributed Economic Meta-Scheduler for the Federation of Grid and Cloud Systems

by

Kyle Chard

A thesis
submitted to the Victoria University of Wellington
in fulfilment of the
requirements for the degree of
Doctor of Philosophy
in Computer Science.

Victoria University of Wellington
2011

Abstract

The computational landscape is littered with islands of disjoint resource providers including commercial Clouds, private Clouds, national Grids, institutional Grids, clusters, and data centers. These providers are independent and isolated due to a lack of communication and coordination, they are also often proprietary without standardised interfaces, protocols, or execution environments. The lack of standardisation and global transparency has the effect of binding consumers to individual providers. With the increasing ubiquity of computation providers there is an opportunity to create federated architectures that span both Grid and Cloud computing providers effectively creating a global computing infrastructure. In order to realise this vision, secure and scalable mechanisms to coordinate resource access are required. This thesis proposes a generic meta-scheduling architecture to facilitate federated resource allocation in which users can provision resources from a range of heterogeneous (service) providers.

Efficient resource allocation is difficult in large scale distributed environments due to the inherent lack of centralised control. In a Grid model, local resource managers govern access to a pool of resources within a single administrative domain but have only a local view of the Grid and are unable to collaborate when allocating jobs. Meta-schedulers act at a higher level able to submit jobs to multiple resource managers, however they are most often deployed on a per-client basis and are therefore concerned with only their allocations, essentially competing against one another. In a federated environment the widespread adoption of utility computing models seen in commercial Cloud providers has re-motivated the need for economically aware meta-schedulers. Economies provide a way to represent the different goals and strategies that exist in a competitive distributed environment. The use of economic allocation principles effectively creates an open service market that provides efficient allocation and incentives for participation.

The major contributions of this thesis are the architecture and prototype implementation of the DRIVE meta-scheduler. DRIVE is a Virtual Organisation (VO) based distributed economic meta-scheduler in which members of the VO collaboratively allocate services or resources. Providers joining the VO contribute obligation services to the VO. These contributed services are in effect membership "dues" and are used in the running of the VOs operations – for example allocation, advertising, and general management. DRIVE is independent from a particular class of provider (Service, Grid, or Cloud) or specific economic protocol. This independence enables allocation in federated environments composed of heterogeneous providers in vastly different scenarios. Protocol independence facilitates the use of arbitrary protocols based on specific requirements and infrastructural availability. For instance, within a single organisation where internal trust exists, users can achieve maximum allocation performance by choosing a simple economic protocol.

In a global utility Grid no such trust exists. The same meta-scheduler architecture can be used with a secure protocol which ensures the allocation is carried out fairly in the absence of trust. DRIVE establishes contracts between participants as the result of allocation. A contract describes individual requirements and obligations of each party. A unique two stage contract negotiation protocol is used to minimise the effect of allocation latency. In addition due to the co-op nature of the architecture and the use of secure privacy preserving protocols, DRIVE can be deployed in a distributed environment without requiring large scale dedicated resources.

This thesis presents several other contributions related to meta-scheduling and open service markets. To overcome the perceived performance limitations of economic systems four high utilisation strategies have been developed and evaluated. Each strategy is shown to improve occupancy, utilisation and profit using synthetic workloads based on a production Grid trace. The gRAVI service wrapping toolkit is presented to address the difficulty web enabling existing applications. The gRAVI toolkit has been extended for this thesis such that it creates economically aware (DRIVE-enabled) services that can be transparently traded in a DRIVE market without requiring developer input. The final contribution of this thesis is the definition and architecture of a Social Cloud – a dynamic Cloud computing infrastructure composed of virtualised resources contributed by members of a Social network. The Social Cloud prototype is based on DRIVE and highlights the ease in which dynamic DRIVE markets can be created and used in different domains.

Acknowledgments

I am deeply indebted to my supervisors Kris Bubendorfer and Peter Komisarczuk for making this research possible. Not only did they provide invaluable leadership and guidance, but they also provided great company and friendship.

I would like to thank my examiners; John Hine, David Abramson and Ioan Raciu for their insightful questions and comments which have strengthened my thesis.

I am grateful to Prof. Ian Foster and Ravi Madduri (University of Chicago/Argonne National Laboratory) for supporting my visits to ANL and leading the development of gRAVI, Prof. Omer Rana (University of Cardiff) for inviting me to Cardiff and helping develop the vision of a Social Cloud, and Prof. Rajkumar Buyya (University of Melbourne) for inviting me to work with his team.

I would like to thank all the past and present MCS/ECS graduate students and lecturers for the countless informal discussions and contributions to my research. The school administrative and IT support staff for all of their help. All the members of the DSRG research group and the Memphis community for their technical contribution, company and most importantly weekly sports. Ian Welch for his support and advice. John Haywood for his statistical assistance. Simon Caton for his contribution to the Social Cloud. Miheal Hategan for giving up his summers and sharing his house. Dina Sulakhe and Wei Tan for their collaborative work on gRAVI. And everyone in the Globus group for making me feel welcome during my visits.

I also greatly appreciate the support from the School of Engineering and Computer Science, without which, I would not have been able to take advantage of the many opportunities I have been given. I owe a special thanks to the head of school, John Hine, who has gone out of his way to support my every request. I am also thankful to both the Tertiary Education Commission and Victoria University of Wellington for the scholarships that have made my thesis possible.

Finally, I owe an enormous debt of gratitude to my family and friends for all their support over the years. Most importantly, I can never thank my parents enough for the sacrifices they have made and their unconditional support, encouragement and guidance.

Contents

List of Figures

List of Tables

Listings

Chapter 1

Introduction

1.1 Introduction

Large scale computing has traditionally been available only to a limited number of users. However, the emergence of Grid and Cloud systems has made high performance computing systems more readily accessible. The availability of Cloud providers and the associated utility models predominantly employed has reprised the vision of global computing utilities. Currently providers form isolated islands of computation, with no interaction, coordination or cooperation between one another [1]. Consumers are therefore bound to a single provider due to a lack of global transparency. Federation of Clouds, and federation of Clouds with existing Grid infrastructures, could be used to create a global computing utility capable of offering enormous computing leverage on-demand.

In order to create a federated architecture, secure and scalable mechanisms to manage and coordinate resource access are required. This is essential as the resource providers themselves may be commercial entities, and must be protected from fraud, theft and resource subversion before they can be expected to participate in any such federation. Global resource management over multiple commercial providers re-motivates the use of economically aware allocation mechanisms driven by the underlying allocation principles used by Cloud providers. There are a number of other issues that must also be considered in the design of a federated allocation architecture, for example trust and security, extensibility and flexibility, interoperability, global scalability, and investment in dedicated infrastructure.

Cloud computing has gained much exposure due (in part) to the launch of various commercial offerings, including Amazon Elastic Compute Cloud (EC2)[1] and Google AppEngine[2], and the development of Open Science Clouds such as Nimbus [2] and Eucalyptus [3]. In addition there are hundreds of institutional Grids, national Grids (Broadband enabled Science and Technology

[1]`http://www.aws.amazon.com/ec2/`
[2]`http://www.code.google.com/appengine/`

Grid[3], Australian Partnerships for Advanced Computing Grid [4], ChinaGrid[4]) and production Grids (Enabling Grids for E-sciencE (EGEE) [5], TeraGrid [6, 7], Open Science Grid (OSG) [8, 9], Grid5000 [10]) available to consumers.

Grid and Cloud federation is essentially an extension of the Grid computing model adding further heterogeneity in terms of task domains (jobs, VMs, Services), security infrastructures, allocation algorithms, execution models, and Quality of Service (QoS) guarantees. In Grid environments, resource providers are typically individually managed by heterogeneous Local Resource Managers (LRMs). LRMs have only a local or organisational view of the Grid and are unable to collaborate when allocating resources. This approach therefore leads to optimal local allocations but suboptimal global allocations and results in underutilisation of the Grid as a whole. Grid meta-schedulers focus on global allocation by acting at a higher level able to submit workload to multiple heterogeneous LRMs. However, most meta-schedulers are implemented on a per-client basis and are therefore only concerned with their own clients' allocations rather than achieving globally optimal allocation across LRMs. Client oriented meta-schedulers compete by submitting jobs to resources with no knowledge of other schedulers actions therefore reducing overall efficiency.

DRIVE (Distributed Resource Infrastructure for a Virtual Economy) [11, 12] is a federating economic meta-scheduler capable of creating open service markets. DRIVE is based on a flexible dynamic Virtual Organisation (VO) [13], in which the VO is responsible for providing the coordination between resources and users, and abstracting the underlying provider implementation. The VO itself is executed on resources contributed by the member providers. In DRIVE resource providers (Grid, Cloud or Service) compete in an open economy for the right to host user tasks. Economies are used as they create incentives for participation, provide efficient allocation, and are necessitated by the utility models predominantly used by commercial providers. There are a number of reasons why DRIVE is a suitable base architecture for creating federated distributed systems, in particular it is distributed, decentralised, scalable, and operates using secure allocation protocols that can be hosted on participating providers thus sharing the burden of allocation and reducing the requirement for dedicated infrastructure. DRIVE has been designed to be autonomous and self-managing, while also being extensible through plug-in allocation protocols, valuation functions and bidding policies.

DRIVE is not envisaged to be the sole means of allocation in a large scale federation, rather multiple (economic) meta-schedulers representing different user groups may operate concurrently in the market. Figure 1.1 depicts this vision of a federated architecture in which multiple Grid and Cloud providers are available to users through a collection of high level meta-schedulers including DRIVE, Community Scheduler Framework (CSF) [14, 15, 16], Grid Federation [17, 18] and GridWay [19, 20, 21].

DRIVE is designed to facilitate dynamic service trading as service based models are commonly used in distributed and network based computing. However, most applications are not web

[3]`http://www.bestgrid.org/`

[4]`http://www.cngrid.org/`

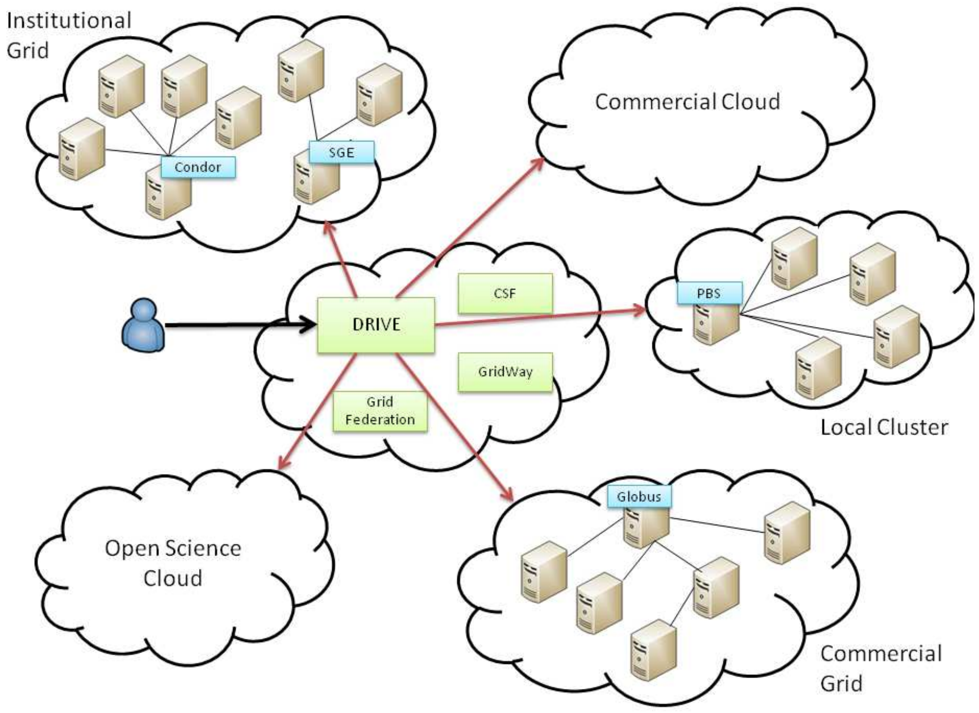


Figure 1.1: A federated environment containing Grid and Cloud based resource providers. A number of meta-schedulers (such as DRIVE, CSF, GridFederation, and GridWay) are used to allocate workload to resource providers.

enabled and the task of creating services is difficult due to the range of complex technologies, tools, standards and languages involved. Exposing services in a market is even more difficult as users need to understand and adhere to different economic protocols, implement valuation functions and bidding policies, and establish and honour contracts. To address the significant difficulties creating and trading services in a DRIVE market this thesis presents the design and implementation of the gRAVI (Grid Remote Application Virtualisation Interface) toolkit. The gRAVI toolkit can be used to wrap arbitrary applications as Web Services Resource Framework (WSRF) [22] Web services which can then be transparently traded in a DRIVE market. All service code, scripts and definition files are created automatically without any developer input. We term this process Software as a Tradable Service (SaaTS).

To evaluate the DRIVE architecture a fully functional federated Grid prototype has been developed. The prototype is job based and is used to evaluate the performance of DRIVE using synthetic Grid workloads allocated over a federated virtualised testbed. To demonstrate the flexibility of the DRIVE architecture the prototype has been extended to allocate Cloud storage in a proof of concept deployment of a Social Cloud. Not only does this Social Cloud deployment highlight the versatility of DRIVE it is a novel contribution in its own right. A Social Cloud is a unique scalable computing environment in which virtualised resources contributed by users of a social network are dynamically provisioned amongst a group of friends.

The remainder of this chapter presents an overview of the DRIVE meta-scheduler, enumerates the contributions made in this thesis, outlines the publications derived from this research and describes the structure of this thesis.

1.2 DRIVE Overview

DRIVE is a distributed economic meta-scheduler in which core management and allocation services are contributed by participating service providers, thus distributing the cost of resource allocation across the members of the VO. These contributed services are in effect membership "dues" and are used in the running of the VOs operations. The distributed nature of DRIVE facilitates implementation of unique distributed (trustworthy) allocation protocols and also limits investment in dedicated infrastructure which reduces potential barriers for entry. The VO model provides scalability from local single organisation Grids through to large scale global federations that span organisational boundaries.

The widespread adoption of utility computing models seen in commercial Cloud providers has re-motivated the need for economic allocation mechanisms. Economic protocols have long been used to allocate computational resources in distributed environments [23] as they provide effective decentralised decision making and offer incentives for participation [24, 25, 26, 27]. Economies are typically scalable and adaptable to rapidly changing market conditions [28]. They also provide a well understood class of protocols with extensive literature outlining their use. Auctions in particular have been shown to provide an efficient means of economic resource allocation due to their ability to establish market prices and represent fine grained QoS requirements [29], in addition they exhibit low communication overhead for sealed bid protocols [25].

A major design goal of DRIVE is the ability to support a range of topologies from small scale local grids through to global federations. To achieve this goal DRIVE provides a plug-in allocation protocol mechanism, allowing protocols to be selected for, and tuned to, specific environments. For example, within an organisation internal trust exists, in this case DRIVE users can achieve maximum allocation performance by choosing an efficient protocol plug-in (for example a simple auction) which does not utilise expensive trust or privacy preserving mechanisms. In a global environment no such trust exists. The same DRIVE architecture can be used with a secure trustworthy protocol [30] plug-in ensuring the allocation is carried out fairly and valuation details are kept private. For large jobs the overhead of using a trustworthy protocol may be insignificant, for smaller jobs however sub-scheduling tools such as Falkon [31] can be used to provide rapid allocation to smaller tasks and therefore amortise the overhead.

DRIVE services are defined by a deployment trust model which outlines the requirements of each service. In order for resources to be allocated jobs they must first join the VO, on joining, a resource provider exposes a set of *obligation* services which it contributes to the VO for the running of the meta-scheduler. Once a member of the VO, the resource provider also exposes *participation* services which are needed to participate in the economy and execute jobs on the resource provider. The remaining *trusted core* services are, in the current architecture, hosted by a set of trusted hosts due to requirements for trust or availability.

DRIVE makes use of a novel two stage contract process to mitigate the effects of allocation latency and encourage negotiation participation. As the result of an allocation DRIVE creates a tentative soft agreement which represents soft-state resource rights, it does not guarantee resources will be available. Upon redemption, the tentative agreement is renegotiated between the parties to establish a hardened binding contract containing specific Service Level Agreements (SLAs) that define agreed upon levels of service.

1.3 Contributions

This thesis makes multiple research contributions in the areas of federated architectures, meta-scheduling, economic resource allocation, web service wrapping toolkits, and Cloud computing. Specifically, this thesis:

1. Presents the design and prototype implementation of DRIVE – a distributed economic meta-scheduler designed to facilitate large scale resource trading in federated distributed environments. DRIVE is unique for a number of reasons:

    - DRIVE is a co-op architecture in which core components of the meta-scheduler are hosted on (potentially untrusted) participating providers. Trust is established through secure economic protocols.
    - DRIVE is allocation protocol independent to provide the option of simultaneously using both (efficient) insecure and secure protocols in different scenarios. Independence is provided by a generic protocol plug-in interface exposed by DRIVE market services. In addition a generic Allocation Component has been designed that can be instantiated as any protocol dependent entity.
    - DRIVE is domain and provider independent. This separation makes DRIVE a suitable platform from which to create a federated Grid and Cloud environment while also facilitating arbitrary service markets. Task independence is achieved by supporting arbitrary term languages for task submission, allocation, reservations and contracts.
    - DRIVE uses a unique two phase SLA negotiation mechanism to encourage negotiation participation and reduce the impact of latency in distributed economies. Soft tenta-

tive agreements are created as the result of a negotiation, upon redemption tentative agreements are hardened into binding contracts containing SLAs.

- A hint based reverse discovery mechanism is used to reduce the burden on allocation entities. Providers receive direct advertisements for suitable negotiations, they also have the option of discovering published negotiations through Publishing Services.
- The DRIVE architecture includes a task-independent flexible advanced reservation model therefore providing consumers with flexibility when describing task requirements and interdependencies. This model has been shown to be advantageous to providers due to the ability to schedule tasks.
- DRIVE components are extensible through user defined policies and plug-in interfaces. For example the DRIVE Agent dynamically loads risk policies and valuation functions at runtime to value requests.
- DRIVE is based on a dynamic VO model to represent consumers, providers, and DRIVE services distributed over multiple administrative domains with proprietary security configurations.

2. Analyses the performance of the DRIVE prototype in the context of auction based allocation. To conduct this analysis several synthetic workloads have been developed to model situations where economic allocation typically performs poorly. A variety of experiments are presented focused on bidding policies, auction throughput, DRIVE service overhead, and economic (profit/penalty) analysis. The results verify the implementation and also serve to quantify the salient performance measurements of DRIVE.

3. Identifies and quantifies the performance benefits of four High Utilisation strategies which can be used to overcome some of the perceived limitations of economic mechanisms. The four strategies studied are overbidding and overbooking, second chance substitute providers, flexible advanced reservations, and just-in-time bidding. These strategies can be employed both through allocation protocols and by participants to increase resource occupancy and therefore optimise overall utilisation. They are shown to improve the performance of economic (auction) protocols in the allocation of resources in a high performance computing environment using DRIVE.

4. Presents the design and implementation of gRAVI – an economically enabled Grid and Web service wrapping toolkit. gRAVI is capable of rapidly web enabling legacy applications such that they can be transparently integrated in a DRIVE-based service market. The generated WSRF services are independent from a particular hosting platform and support GSI security, Grid scheduling, state notifications, persistence and data staging. All service code, scripts and definition files are created automatically without any developer input. gRAVI is able to deploy services directly to the open science cloud Nimbus, while also being able to parse workflow definition files to create strongly typed services. To our knowledge this is the only example of a service wrapping toolkit capable of creating economically enabled services.

5. Defines a Social Cloud – a Cloud computing infrastructure utilising virtualised resources contributed by members of a Social network. Additionally the design and implementation of a proof of concept Social Storage Cloud based on DRIVE is presented which highlights the versatility of the DRIVE architecture. The prototype Social Cloud is implemented as a Facebook application which allows "friends" to trade storage with one another in an open DRIVE market. Performance measurements for the deployed posted-price and auction based marketplaces are shown to exceed the requirements of a small Social Cloud. This is the first example of a Cloud computing environment that leverages the trust relationships established in a Social network.

1.4 Scope

The DRIVE architecture is widely defined covering many aspects of meta-scheduling, economic resource allocation, and SLA management. Due to the size of this undertaking it is important to limit the scope of this thesis to the creation of the DRIVE meta-scheduler and the associated components required to facilitate economic resource allocation over large scale distributed environments. This includes the creation of a distributed service infrastructure to support arbitrary auction protocols it does not include detailed implementation or analysis of these protocols. The primary focus of this thesis is developing allocation mechanisms that can be applied in a federated environment, other aspects of federation such as inter-provider data transfer and heterogeneous task execution is outside the scope of this thesis. Where possible DRIVE leverages existing mechanisms for resource discovery and job submission. DRIVE establishes SLAs as the result of allocation however monitoring and enforcement of these agreements is outside the scope of this thesis. The concept of reputation is referred to throughout this thesis as an example of a non-functional implication of trading, however implementation of reputation mechanisms and protocols is also outside the scope of this research.

1.5 Publications

During my PhD candidature I have collaborated with researchers both internally and externally. This work resulted in several joint publications derived from Chapters in this thesis. Specifically:

The following publications are derived from Chapters 4 and 5 detailing the design and implementation of DRIVE:

- **K. Chard** and K. Bubendorfer, Using Secure Auctions to Build a Distributed Meta-scheduler for the Grid, in Market Oriented Grid and Utility Computing, R. Buyya and K. Bubendorfer, Eds. New York, USA: Wiley Press, 2009.
- **K.Chard** and K. Bubendorfer. A Distributed Economic Meta-scheduler for the Grid, in Proceedings of the 8th IEEE International Symposium on Cluster Computing and the Grid (CCGRID). Lyon, France. 2008.

The following publication is derived from Chapters 5 and 6 describing the design and evaluation of the proposed high utilisation strategies:

- **K.Chard**, K. Bubendorfer, and P. Komisarczuk. High Occupancy Resource Allocation for Grid and Cloud systems, a Study with DRIVE, in Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing (HPDC), Chicago, IL, USA, 2010.

gRAVI was developed in conjunction with Professor Ian Foster, Ravi Madduri and the Globus group (University of Chicago/Argonne National Laboratory), Chapter 7 is derived in part from the following publications:

- **K. Chard**, W. Tan, J. Boverhof, R. Madduri, and I. Foster. Wrap Scientific Applications as WSRF Grid Services using gRAVI, in Proceedings of the 7th IEEE International Conference on Web Services (ICWS), Los Angeles, CA, USA, 2009.

- W. Tan, **K. Chard**, D. Sulakhe, R. Madduri, I. Foster, S. Soiland-Reyes, and C. Goble, Scientific workflows as services in caGrid: a Taverna and gRAVI approach, in Proceedings of the 7th IEEE International Conference on Web Services (ICWS), Los Angeles, USA, CA, 2009.

- **K. Chard**, C. Onyuksel, W. Tan, D. Sulakhe, R. Madduri, and I. Foster, Build Grid Enabled Scientific Workflows using gRAVI and Taverna, in Proceedings of the 4th IEEE International Conference on eScience (eScience), Indianapolis, IN, USA, 2008.

The Social Cloud concept was developed in conjunction with Professor Omer Rana and Simon Caton (Cardiff University), Chapter 8 is partially derived from the following publication:

- **K. Chard**, S. Caton, O. Rana, and K. Bubendorfer. Social Cloud: Cloud Computing in Social Networks, in Proceedings of the 3rd IEEE International Conference on Cloud Computing (CLOUD), Miami, FL, USA, 2010.

1.6 Thesis Organisation

This thesis is organised as follows. Chapter 2 provides an overview of related work in Grid and Cloud computing, meta-scheduling, and economic resource allocation. This chapter begins by developing a taxonomy designed to classify common meta-scheduling approaches before describing a variety of systems with similar goals to DRIVE. Chapter 3 presents five motivating use cases that attempt to characterise common application requirements. Chapter 4 introduces DRIVE and outlines the general architecture independent of a given domain, this chapter focuses on allocation mechanisms, service provider interactions, contract management, VO management, and service discovery. A prototype implementation of DRIVE developed for Grid federation is presented in Chapter 5. The prototype includes functional implementations of all core DRIVE services and includes three reverse auction protocols. The DRIVE architecture is validated experimentally in Chapter 6, this chapter presents various performance measurements and examines the proposed high utilisation strategies designed to overcome economic limitations. Chapter 7 presents the gRAVI toolkit, focusing on the DRIVE extensions used to generate economically enabled services. Chapter 8 defines the concept of a Social Cloud before presenting a storage-based prototype implementation utilising a DRIVE-based service market, quantitative evaluation of the prototype is presented under different market conditions. Finally Chapter 9 concludes this thesis, reviewing the DRIVE architecture, providing a summary of contributions and proposing future research directions.

Chapter 2

Related Work

It has long been realised that enormous (super)computing power can be harnessed by combining the computational resources of individual computing systems [32, 33]. This concept was the major motivation for the development of Cluster, Grid, and more recently Cloud computing. Each of these models manages, and coordinates access to, a group of distributed computing resources. These distributed models can be deployed within organisations, across organisational boundaries, or even formed dynamically from donated resources (Volunteer Computing [34, 35]). While these architectures each facilitate large scale resource sharing increased value can be obtained through utility models, in which consumers have "plug-in" access to global computing resources and providers implement dynamic (metered) billing policies to offset infrastructure and operational costs. The concept of a computing utility was first proposed by John McCarthy in 1961:

> *"if computers of the kind I have advocated become the computers of the future, then computing may someday be organised as a public utility just as the telephone system is a public utility... The computer utility could become the basis of a new and important industry"* [1]

McCarthy's vision has remained unrealised for a number of reasons including lack of motivation (use cases) for users and difficulty creating, coordinating, managing, and securing such infrastructures. To create a true computing utility a federated global environment is required such that users can submit tasks transparently to a pool of distributed heterogeneous resource providers. In such a model consumers and providers may be in different administrative domains, under separate management and control, with differing environments and local policies.

Computing clusters provided the first high-performance distributed architecture combining pools of homogeneous resources for large scale parallel computation. Grid computing built upon this concept by federating heterogeneous resources (Clusters, Supercomputers, Data centers, and Instruments) distributed over a wide area network. In a Grid resources are managed independently and may be contributed from different administrative domains. Grid middleware is used

[1]John McCarthy, MIT Centennial Speech, 1961

to provide security, discovery, resource management, and file transfer. Cloud computing has further developed this vision through the adoption of virtualisation and utility business models. While many clusters, large scale Grid environments, and commercial/non-commercial Cloud providers are readily available worldwide they have effectively formed 'islands' of disconnected computation which lack the ability to interact with one another [1].

In order to exploit distributed architectures in an efficient manner high level schedulers are used to allocate resources in multi-provider environments. Within a Grid system there are multiple levels of scheduling. At the resource level LRMs schedule jobs to individual computing resources such as nodes in clusters or processors in supercomputers. At an enterprise or institution level work is scheduled across multiple LRMs within a single administrative domain using schedulers such as Portable Batch System (PBS) Pro peer scheduling [36] and Load Sharing Facility (LSF) multi cluster [37, 38]. Grid level scheduling (or meta-scheduling) is more complex as the system spans administrative domains and must interact with heterogeneous resources and low level schedulers. General approaches to Grid level scheduling include meta-schedulers [19], community schedulers [15], superschedulers [39], and resource brokers [40, 41, 42]. Meta-schedulers can therefore be used as the basis for federated resource allocation across administrative domains.

The task of a meta-scheduler is four-fold: resource discovery, resource allocation (or matchmaking), job submission, and job monitoring. In Grid environments, resource discovery is required to locate a set of candidate resources through a Grid Resource Information Service (GRIS). GRIS aggregates resource metadata and exposes an interface to discover resources which satisfy consumer requirements. These results are filtered by removing inaccessible or inappropriate resources (based on credentials or requirements of the task). Resource allocation (or matchmaking) is tasked with determining a "suitable" resource to host a given job according to the policies and mechanisms available to the meta-scheduler (for example queue-based, market-based or policy-based scheduling). Finally meta-schedulers abstract heterogeneous job submission and monitoring through a transparent interface. In addition to these four tasks meta-schedulers also integrate with existing security infrastructures and provide functionality such as single sign on, delegation, and secure messaging.

Figure 2.1 shows a Grid system in which 3 providers are registered with GRIS. The Grid level schedulers sit "above" the Grid and are able to discover suitable resources by querying GRIS. Having found suitable resources the job is scheduled on the appropriate resource using the local or Grid execution interface. The meta-schedulers shown in this topology are deployed as an organisation level service allowing multiple users access to the Grid resources through a single scheduler, the broker is deployed for a single user which provides transparent access to the Grid. Figure 2.2 presents a different Grid model based on VO level community meta-schedulers. Each VO has a community scheduler that is responsible for submitting jobs to providers within the VO on behalf of VO users. Both figures highlight the limitations of scheduling approaches as each of the schedulers acts independently of the others, effectively competing for allocation by submitting workload to resources with no mechanisms to communicate or coordinate access. This lack of coordination can lead to overutilisation of some resources while others remain underutilised and typically resources themselves have little ability to influence allocation.

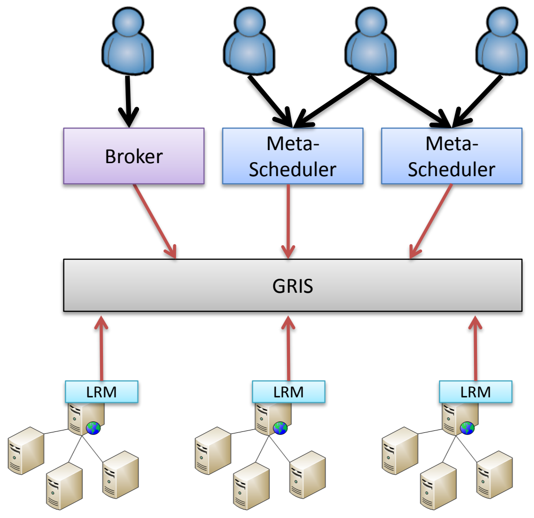


Figure 2.1: Broker and meta-scheduler deployment. The schedulers retrieve resource information from GRIS and schedule jobs independently according to the information discovered.

The concept of Grid and Cloud federation is an extension to the Grid model, as Cloud providers can be viewed as a new type of heterogeneous resource provider. However, due to the utility models predominantly used by Cloud providers meta-scheduling mechanisms must be economically aware. Due to the relatively recent emergence of Cloud computing there has been little research into federations with other types of providers. However many of the approaches proposed for Grid level scheduling provide mechanisms that meet some of the challenges created in large scale federated environments. For example there are a number of meta-schedulers, super-schedulers, and brokers available with different focuses that may be used to provide federated Grid and Cloud management.

This Chapter presents an overview of related work in the area of distributed resource management focusing on (economic) meta-schedulers. The Chapter begins with an overview and definition of Grid and Cloud computing before presenting the taxonomies used to classify the different approaches for distributed resource management and scheduling. Finally systems with similar goals to DRIVE are compared and summarised according to the proposed taxonomies.

2.1 Grid Computing

A Grid is a collection of distributed heterogeneous computing resources that appear to users (or applications) as a single virtual computing system able to deliver defined QoS levels to consumers [43]. Resources in a Grid can be any entity which is shared, including computation re-

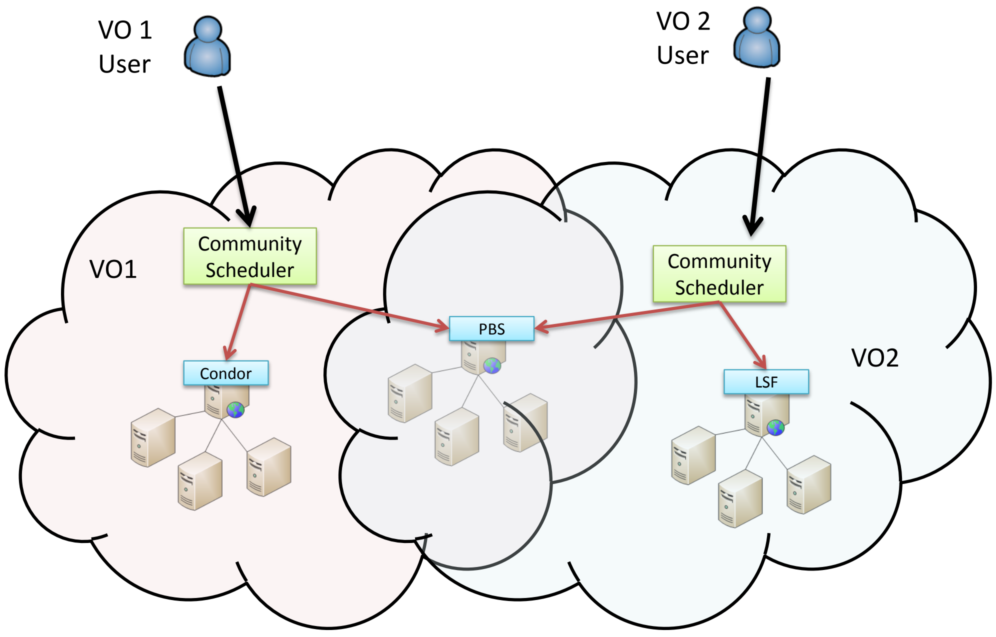


Figure 2.2: Community meta-schedulers deployed within VOs.

sources (desktop computers, clusters, or supercomputers), storage devices, tools, or peripherals such as sensors. Grid environments span organisational boundaries and are inherently distributed with no centralised location or control. The most widely accepted classification of a Grid is Ian Foster's three point checklist [44]. A Grid:

- **Coordinates resources that are not subject to centralised control:** A Grid integrates and coordinates resources and users within different control domains. A Grid addresses the issues of security, policy, payment, membership, and so forth that arise in these domains.
- **Uses standard, open, general-purpose protocols and interfaces:** A Grid is built from multi-purpose protocols and interfaces that address issues such as authentication, authorisation, resource discovery, and resource access.
- **Delivers nontrivial qualities of service:** A Grid allows its constituent resources to be used in a coordinated fashion to deliver various qualities of service, relating for example to response time, throughput, availability, security, and/or co-allocation of multiple resource types to meet complex user demands, so that the utility of the combined system is significantly greater than that of the sum of its parts.

Typically each resource provider in a Grid is independently managed by its own heterogeneous LRM. Commonly used LRMs include Condor [32], PBS, Sun Grid Engine (SGE) [45], and LSF. Application schedulers are used to assign jobs to resources and perform tasks such as resource discovery, scheduling, job submission, and job monitoring.

The concept of a VO is inherent in Grid systems. A VO defines a collection of individuals or institutions grouped together with defined rights on what is shared, who is allowed to share, and under what conditions the sharing can take place. VOs facilitate controlled sharing of resources amongst arbitrary user groups. A VO can encompass a single group within an organisation through to a large group of distributed researchers (and their resources) from a number of different organisations and administrative domains spread all over the world. A Grid can be organised into a number of VOs, each with different policies. As such, the VO provides a way to manage complex Grid systems by establishing and enforcing agreements between resource providers and the VO, and the VO and its members.

The Globus Toolkit (GT) [33] is the predominant Grid middleware system used to create large scale Grid environments. GT provides a set of standard tools, services, and protocols used to build a Grid. GT4 includes components for resource management (Grid Resource Allocation and Management (GRAM) [46, 47]) and resource discovery (Monitoring and Discovery System (MDS) [48, 49]) while also defining the de facto standard Grid Security Infrastructure (GSI) and offering a suite of distributed file transfer mechanisms such as GridFTP and RFT (Reliable File Transfer). There are many national Grids (APAC, ChinaGRID, BeSTGrid), scientific Grids (TeraGrid, OSG), and enterprise/institutional Grids available to users all over the world.

2.2 Cloud Computing

Cloud computing is a scalable elastic computing model in which virtualised resources are provisioned on demand to consumers. Clouds are characterised by their large (massive) scale, resource virtualisation, location transparency, service based interfaces, and utility models.

- In order to make commercial providers economically viable massive scale infrastructures must be developed to handle consumption peaks and also provide fault tolerance to avoid SLA violation.
- Virtualisation hides heterogeneity while also providing the ability to dynamically provision resources.
- By definition Clouds are location transparent, users are unaware of the physical resources or location of resources used, they can also access their resources from anywhere over an open network.
- Standardised self describing service based interfaces facilitate widespread adoption and ease of use.

- Finally utility models are inherent in most Cloud models, resource usage is transparent (through location transparency, standardised interfaces, and virtualisation) and consumers are charged using a metered model in which they pay for the resources used.

Cloud computing is still in its infancy and as such there is no authoritative definition. However, Vaquero et al. [50] have published a meta-definition based on several previously published Cloud definitions. They define a Cloud as:

> *"... a large pool of easily usable and accessible virtualised resources (such as hardware, development platforms and/or services). These resources can be dynamically reconfigured to adjust to a variable load (scale), allowing also for an optimum resource utilisation. This pool of resources is typically exploited by a pay-per-use model in which guarantees are offered by the Infrastructure Provider by means of customised SLAs."*

The broad definition of Cloud computing encompasses a range of service based providers ranging from raw resource providers through to application providers. Due to this large range providers are typically grouped into one of three categories: Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS). Each of these categories essentially builds upon the previous one.

- IaaS forms the basis of Cloud computing, providing virtualised raw resources to consumers. Providers manage a large set of computational and storage resources and offer them to consumers through VMs or virtualised storage. Typically users choose VM templates or create their own VM for the duration of the provision. Billing is by consumption or based on the "size" of the VM. Examples of IaaS Clouds include Storage Clouds (Amazon Simple Storage Service) and Computation Clouds (Amazon Elastic Compute Cloud).

- PaaS sits above IaaS, offering an additional level of abstraction. Rather than raw resources entire application stacks are provisioned, providing consumers with a complete application environment. The advantage for consumers is they no longer need to consider scalability or availability as this is provided by the Cloud. Google AppEngine is the most well known PaaS provider. Like IaaS, consumers are billed for the size or duration of their provision.

- SaaS is the top layer of the Cloud stack offering well defined application services to consumers. Generally SaaS approaches act as if an application is run locally. Billing is typically based on duration, for example monthly fees. There are many examples of SaaS providers such as Google docs [2] and Flikr [3] photo storage.

2.3 A Comparison of Meta-Scheduling Approaches

The range of meta-schedulers currently available differ greatly in terms of focus domain, task domain, and architecture. This section outlines a meta-scheduler taxonomy used to classify

[2] `http://www.docs.google.com`
[3] `http://www.flickr.com`

various approaches to meta-scheduling and distributed resource management. The taxonomy presented extends previous work in broker, scheduling, and economic resource allocation taxonomies [51, 52, 18]. In this brokers, superschedulers, and community schedulers are all considered meta-schedulers therefore the terms are used interchangeably. In the taxonomies a dotted line indicates multiple options can be employed by the same meta-scheduler whereas bold lines indicate generally only one of the options is supported. Fourteen meta-schedulers have been surveyed to create this taxonomy: SORMA [53], GridWay [19, 20, 21], Grid-Federation [17, 18], Community Scheduler Framework (CSF) [14, 15, 16], Nimrod/G [40, 41, 54], Tycoon [55], Bellagio [56], SHARP [57], EMPEROR [58], GMarteGS (GMarte Grid Service) [59], Trader-Federation [60, 61], Multi Agent Federation [62], OurGrid [63, 64], and DRIVE.

Figure 2.3 presents the root taxonomy used to classify the different systems considered in this survey. The *architecture* of each meta-scheduler differs based on the infrastructure on which it is built, the middleware supported and the granularity of deployment. Resource *discovery* is one of the core responsibilities of a meta-scheduler however a number of different approaches are used based on both centralised and decentralised architectures. The *task model* employed influences the type of task allocated and the description representation used by the meta-scheduler. Different *allocation* architectures and scheduling algorithms are used by various meta-schedulers ranging from static scheduling through to dynamic queue and market based models. Increasingly meta-schedulers consider user defined *Quality of Service* (QoS) constraints and provide *advanced reservation* of resources to meet the requirements of complex and interdependent tasks. Participant *registration* differs between meta-scheduling implementations depending on the underlying distributed model. Finally the *trust and security* model used by meta-schedulers depends on how trust is established, how communication and components are secured, and how parties are authenticated and authorised. In addition to these categories the means by which meta-schedulers have been evaluated is also considered in section 2.3.9, this is included to develop performance comparisons with DRIVE. The full meta-scheduler taxonomy is presented in Figure 2.4, due to the size and complexity of this taxonomy the leaves of the root taxonomy (Figure 2.3) are discussed individually in the following sections 2.3.1– 2.3.8.

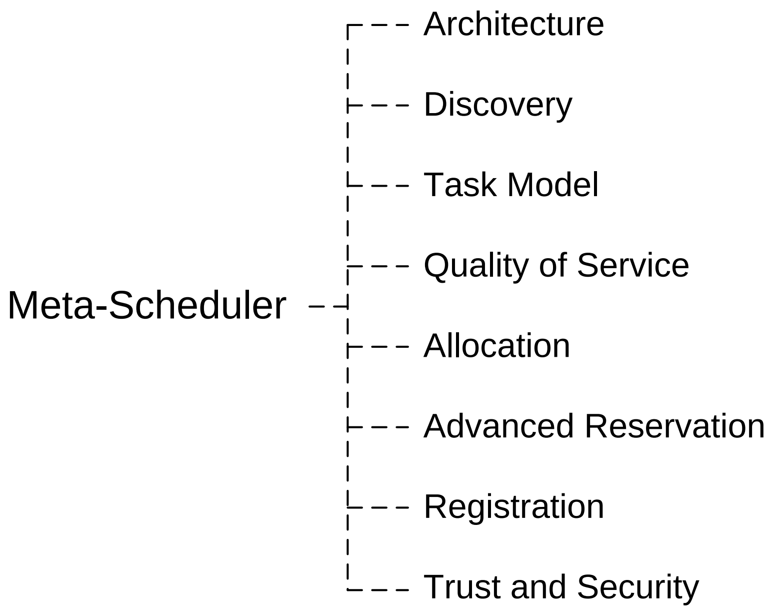


Figure 2.3: Meta-scheduler Taxonomy.

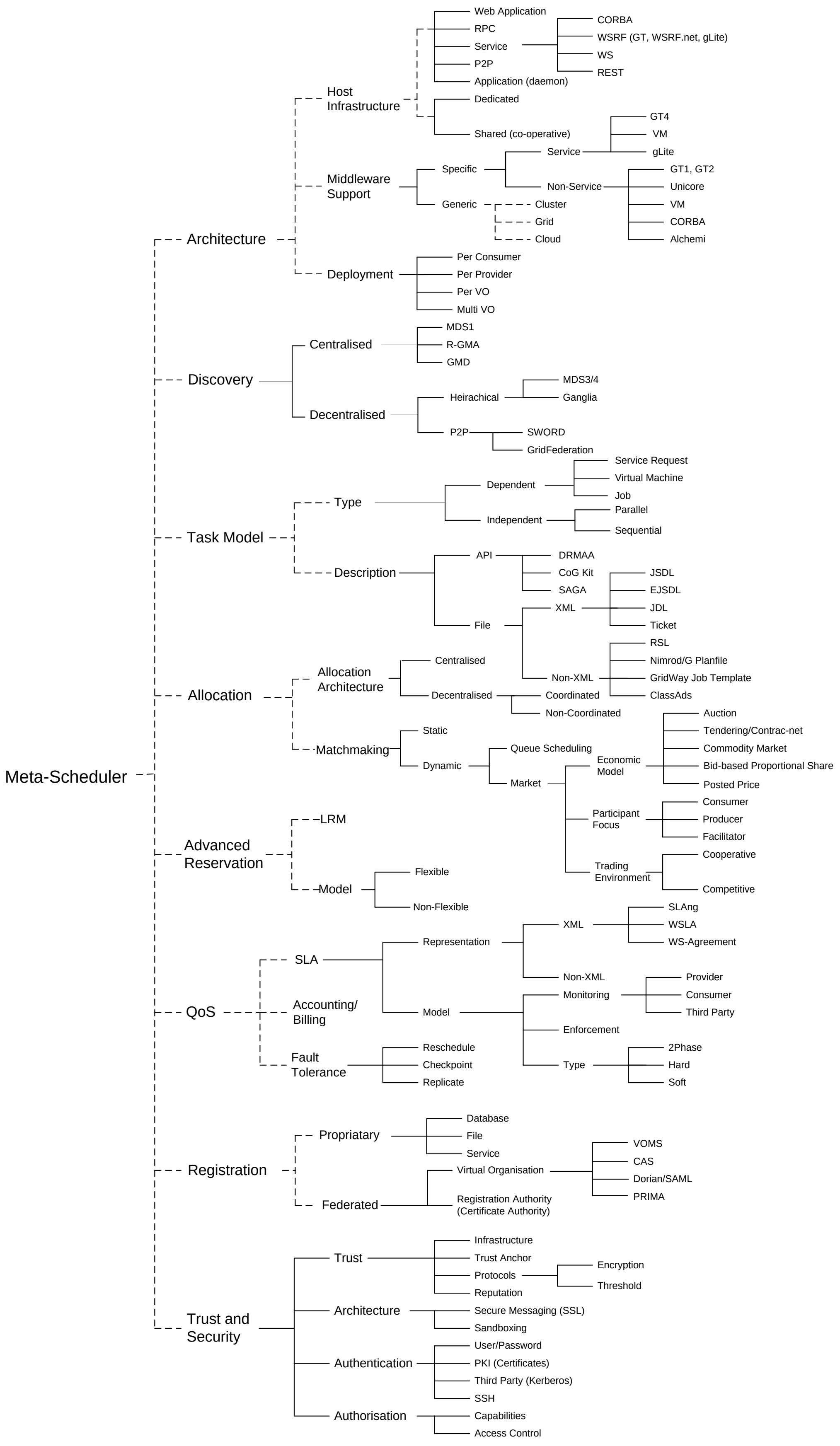


Figure 2.4: Meta-scheduler Taxonomy.

2.3.1 Architecture Taxonomy

The architecture, middleware support and deployment options of a meta-scheduler are generally dependent on the environment and focus of the system. The architecture leaf of the taxonomy is presented in Figure 2.5, this leaf is used to classify the different architectural characteristics seen in common meta-schedulers. Middleware support is extended from [51].

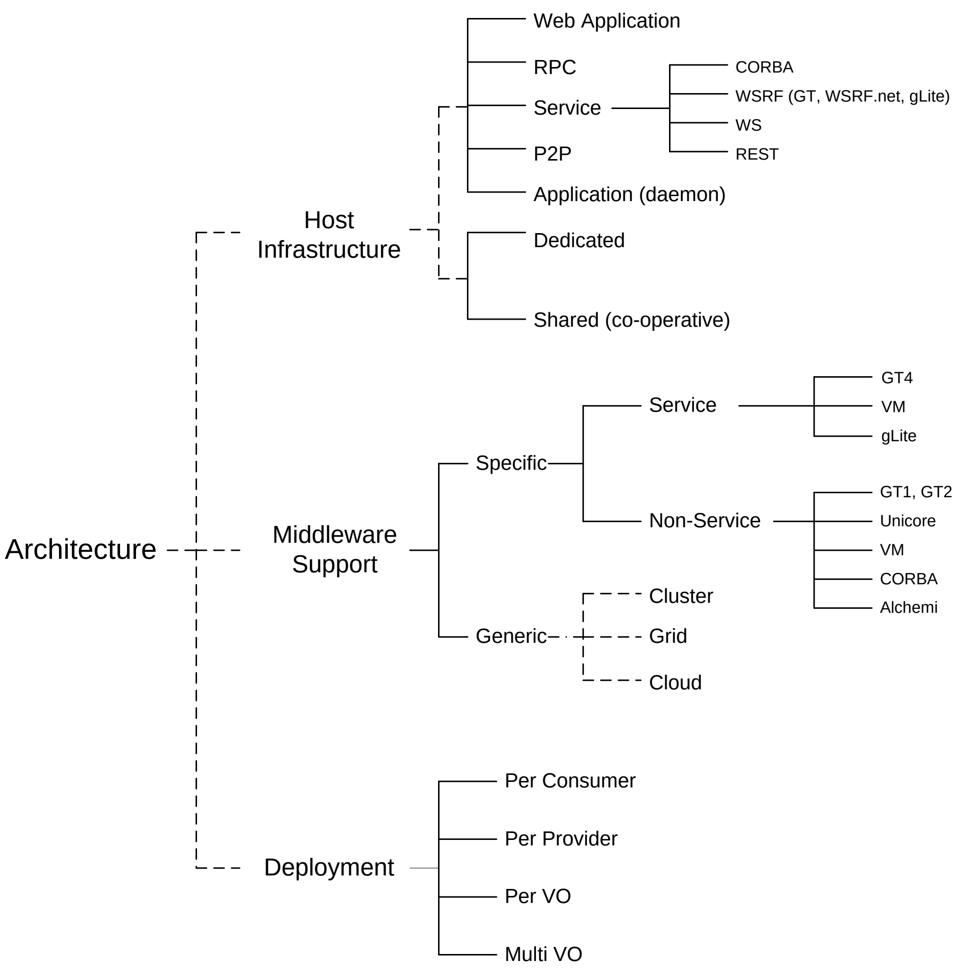


Figure 2.5: Architecture taxonomy.

Meta-schedulers are typically hosted on dedicated resources to provide trust, availability and performance. It is also possible to use shared (or co-operative) infrastructures assuming trust can be established through other means, for example DRIVE establishes trust through secure auction protocols. Increasingly meta-schedulers are service based due to the service based architectures used by many Grid and Cloud providers. Original meta-schedulers were often implemented as standalone applications (or daemon processes) using proprietary communication protocols over socket interfaces. For example Nimrod/G, Tycoon, and Sharp are all application based with proprietary interfaces. Most recent meta-schedulers utilise service based infrastructures to create scalable standardised distributed architectures such as CORBA services used in Trader Federation and WSRF services used in DRIVE, CSF, and GMarteGS. Other options for hosting meta-schedulers include web applications, RPC applications (Bellagio), and P2P based applications (Grid Federation).

Typically meta-schedulers are designed to be used in a particular domain or with a specific type of Grid middleware, those that are not designed for a specific type of middleware (generic) are normally aimed at a particular type of distributed environment (Grid, Cluster, Cloud). Specific middleware support is further classified as service based middleware such as Globus Toolkit 4 or non-service based middleware like pre web service Globus Toolkit or Unicore. The infrastructure support is crucial when considering a federated architecture as the meta-scheduler cannot be bound to a single provider infrastructure. Most of the meta-schedulers considered support service based middleware, however Bellagio and Sharp support non-service based PlanetLab [65] middleware

The final architectural consideration is the deployment granularity of the meta-scheduler. Often meta-schedulers are deployed per-user such that the meta-scheduler provides transparency over the entire heterogeneous resource pool for a single user. For example Nimrod/G, Tycoon, and GridWay can all be deployed per user. Server based meta-schedulers are deployed at the organisation or VO level allowing access for multiple users through the same service or portal, for example Bellagio, Trader-Federation, Emperor and CSF. Multi-VO deployment is required by federated environments, in this configuration a single meta-scheduler is deployed outside the scope of a single VO. DRIVE is not bound to a single deployment model and can be deployed at any of the granularities mentioned.

2.3.2 Resource Discovery Taxonomy

Resource discovery is one of the fundamental functions of a meta-scheduler. The resource discovery taxonomy leaf is presented in Figure 2.6. In most Grid environments resources are discovered through GRIS. The discovery mechanism used is often provided by, or bound to, the middleware platform used. For example in Globus Grids the Globus Monitoring and Discovery System (MDS) provides a group of services used for resource discovery and information propagation. As Grid environments become more complex and the performance requirements increase, centralised approaches are no longer suitable due to a lack of fault tolerance and scalability. Nowadays distributed discovery architectures are often used in large scale Grids.

Original meta-schedulers relied on centralised information services such as MDS1 [48], Relational Grid Monitoring Architecture (R-GMA) [66], and Grid Market Directory (GMD) [67]. Centralised approaches have the advantage of simple management and an efficient single point of contact to update and query registered information ensuring global knowledge. However this centralised service is also a limitation as it presents a single point of failure and does not scale.

As the number of Grid members increases the service suffers from increased network traffic and computational overhead to retrieve and update information. Tycoon, Nimrod/G and OurGrid use centralised discovery architectures.

Distributed information systems can be used to minimise some of the limitations seen in a centralised model. Distributed resource discovery is generally based on hierarchical or Peer to Peer (P2P) architectures. In a hierarchical model individual information services are configured to share information in a tree-like structure. Explicit data links are predefined to specify how information is propagated and discovered. The two most prominent hierarchical resource discovery architectures are MDS3 [49] and Ganglia [68]. Like the Domain Name Service (DNS), hierarchies provide high throughput scalable architectures, in which authoritative results can be found by querying higher services in the tree. This design has the advantage of reducing the need for trusted infrastructure as only authoritative services need to be trusted. Both MDS3 and Ganglia model information services on the structure of VOs, where each VO nominates an information service to store and share information. Like centralised approaches hierarchies often have a single point of failure (the top of the tree). Most current meta-schedulers utilise hierarchical discovery services as they are the predominant model deployed in Grid environments. For example MDS3 is supported by Sharp, Emperor, CSF, GridWay and DRIVE.

In a P2P model a network of peers distributes resource information. P2P architectures are truly distributed in that no single node is more important than any other node, that is there is complete autonomy. By definition this architecture is fault tolerant, self organising and highly scalable. For these reasons there have been various P2P architectures proposed in the literature [69]. Examples of P2P discovery systems used in distributed architectures include SWORD [70] used in Bellagio and PlanetLab, P2P based discovery of tickets used in Sharp, and the Distributed Hash Table (DHT) system used in Grid-Federation.

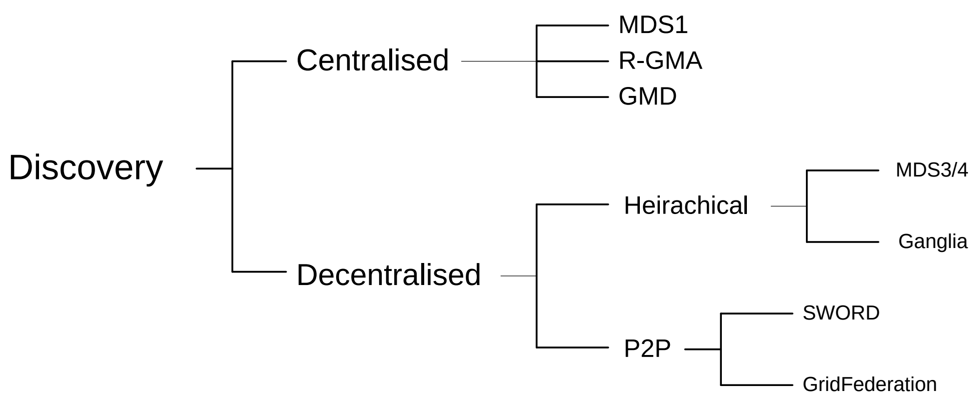


Figure 2.6: Resource discovery taxonomy.

2.3.3 Task Model Taxonomy

In a heterogeneous distributed environment representation of task requirements may differ between consumers, providers, and the allocation framework. To facilitate distributed allocation tasks must be represented such that all parties understand requirements. The task model used is typically dependent on the chosen middleware, however some meta-schedulers are capable of representing generic tasks. Figure 2.7 outlines the task model leaf of the meta-scheduler taxonomy. Most meta-schedulers are designed for job based tasks, due to the focus on Grid environments. As more Service Oriented Grids and Cloud based architectures are developed it is expected that meta-schedulers will increasingly aim to support service requests and VMs as the predominant task models.

Task requirements are typically expressed in a *task description language*. As jobs are the most common task type used in meta-scheduling architectures, the description types expressed in this taxonomy are heavily job-based, however a similar classification can be applied to other domains. In addition some of the languages considered could be applied in other domains, for example in Chapter 8 EJSDL is used to represent storage requirements in a Cloud environment. Most job based task descriptions are file based using either XML or non-XML based files. Common XML description files include Job Submission Description Language (JSDL) [71] used by Gridway and EJSDL (JSDL with economic extensions) [72] used by SORMA, while GMarte and Grid Federation also use independent XML descriptions. Of the non-XML descriptions Globus, EMPEROR, and CSF use Globus Resource Specification Language (RSL) [73], Nimrod/G uses a proprietary plan-file, and OurGrid uses a Job Description File. In some architectures tasks are not described with definition files at all, rather they use direct API calls to specify the requirements of a task. For example GridWay supports DRMAA (Distributed Resource Management Application API) [74] to specify and submit task requests. Other well known APIs include Globus Commodity Grid (CoG) Kit [75] and Simple API for Grid Applications (SAGA) [76, 77], neither of these APIs are used by any of the meta-schedulers considered in this survey.

2.3.4 Resource Allocation Taxonomy

Arguably the most important task of meta-scheduling is resource allocation. Allocation involves matching consumer task requirements with provider resource capacity. Resource allocation relies on different architectures and matchmaking algorithms to determine how tasks should be assigned to resources. Figure 2.8 shows the resource allocation taxonomy leaf. Within a meta-scheduler allocation architectures are either centralised or decentralised. In a centralised approach a single entity schedules all jobs and resources in the system. In a decentralised architecture a group of allocation entities individually manage subsets of resources in the system. Centralised approaches are simple to create and manage, and, due to the global view of the system, they make fast efficient scheduling decisions. However centralised approaches are prone to failure and are less scalable than distributed models. Bellagio, EMPEROR, and GMarteGS use centralised allocation services.

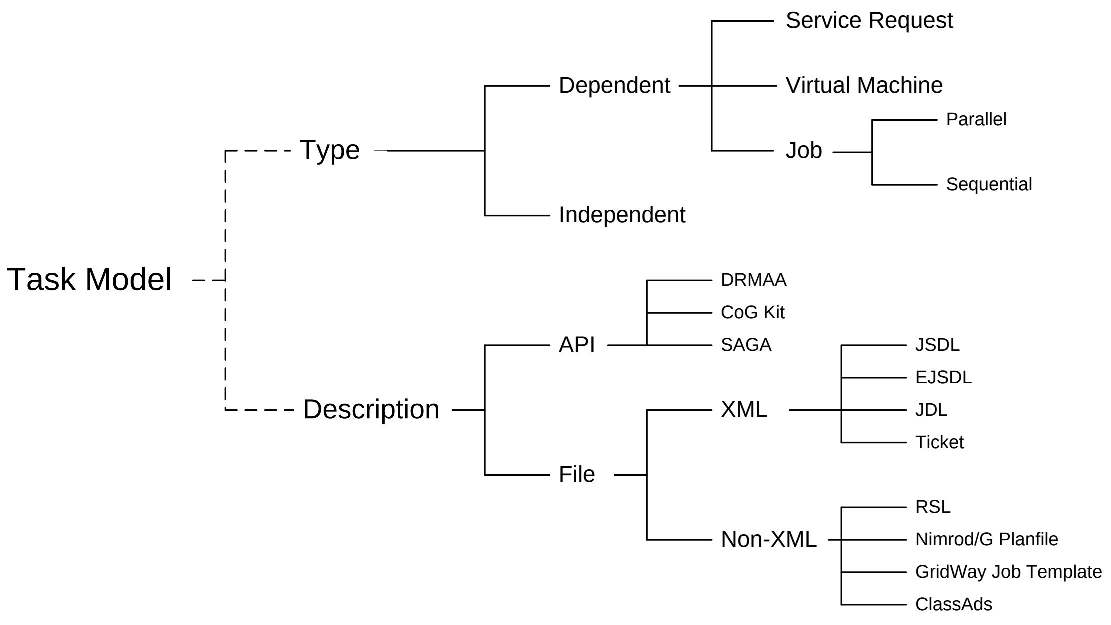


Figure 2.7: Task model taxonomy.

While decentralised approaches may scale and provide fault tolerance, it is more difficult to coordinate allocation decisions as multiple entities are required to schedule tasks. This lack of coordination may result in less optimal allocation. In addition it is difficult to provide trustworthy distributed allocation environments. Decentralised approaches can be further classified as coordinated or non-coordinated depending on how allocation decisions are made. In coordinated architectures each meta-scheduler consults with other meta-schedulers before allocating resources, for example Sharp brokers coordinate decisions through a second tier market and Grid Federation Agents coordinate through a communication protocol over a P2P information network. In a non-cooperative model (for instance GridWay, Tycoon, Nimrod/G, and CSF) scheduling decisions are made independently with no regard for the actions of other schedulers.

Various approaches are used to match jobs to resources. Generally decisions are either made statically (based on simple policies) or dynamically based on aspects such as current resource load and job requirements. Most meta-schedulers use dynamic queue algorithms or market based protocols to determine a suitable resource to execute a given job. Queue based techniques work by applying policies and heuristics to determine which queue to place a job in and the order in which jobs are removed from queues. Queue based meta-schedulers include CSF, EMPEROR, and GridWay. Market based mechanisms have been shown to provide efficient resource allocation by dynamically balancing supply and demand. Markets are likely to be increasingly used by meta-schedulers due to the utility models employed by commercial resource providers. Multiple meta-schedulers use market metaphors to allocate resources including Nimrod/G, Bellagio, Tycoon,

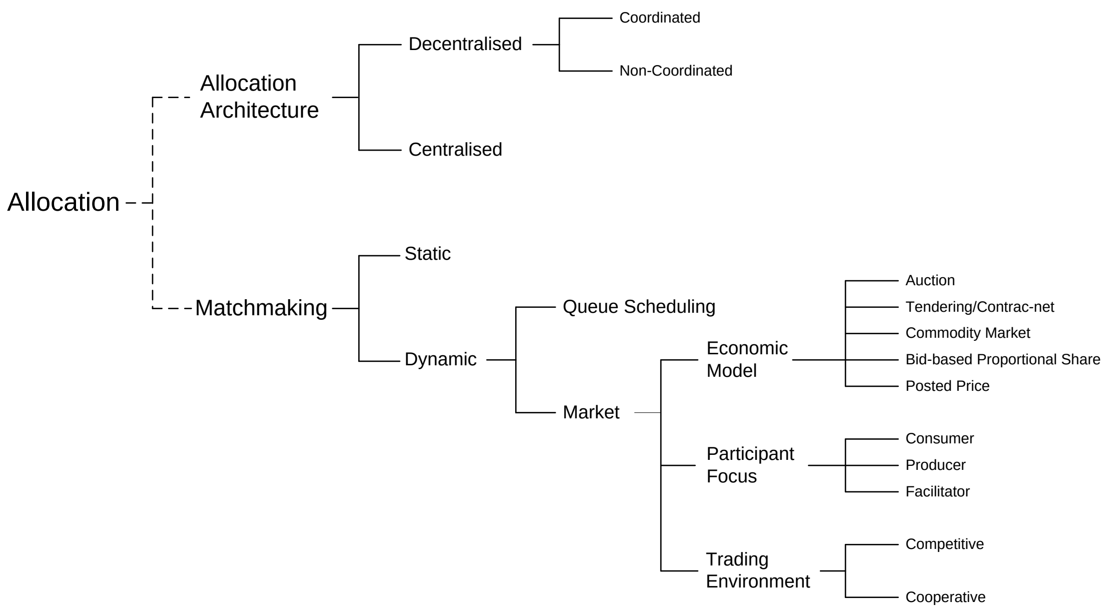


Figure 2.8: Resource allocation and scheduling taxonomy.

Sharp, SORMA, and Grid Federation.

Market-based Allocation

The market taxonomy presented in Figure 2.8 is based on [52], the economic model taxonomy in turn was first presented in [25]. Various market abstractions have been applied to resource allocation in distributed systems, the most common approaches are commodity markets, auctions, and posted price mechanisms.

In a commodity market producers specify prices and consumers are charged for the amount of resource consumed. Commodity markets are used in Nimrod/G, Multi Agent Federation, and Grid Federation. Pricing functions in a commodity market may be flat (similar to posted price) or variable. In a posted price market providers advertise resources at a set price. Posted price models are the predominant model used in Cloud computing, for example Amazon EC2 charges users a fixed price per VM. Auctions facilitate multi-partite negotiation to establish an agreeable price between multiple consumers and a single provider. Auctions are used in Tycoon and SORMA. Reverse auction protocols (or tenders) are well suited to computational resource allocation as they allow a single consumer (user) to solicit bids from multiple providers. The DRIVE prototype is based on reverse auction protocols. Bid based proportional sharing is used in systems like OurGrid and PlanetLab, where resources are allocated in proportion to the bids placed by consumers. Essentially each participating entity wins some part of the allocation based on the relative value of their bid.

Markets aim to provide utility for a particular participant. For example auctions maximise utility for the seller. The participant focus of a market classifies the intended beneficiary of the economic allocation protocol. Consumer focus aims to maximise utility for resource users, for example minimising the money spent or maximising the resources obtained. Producer focus aims to maximise the resource provider utility. Finally facilitator focus represents potential for a facilitator (for example an auctioneer) to gain profit through the negotiation process – this is most applicable in a two tiered market structure.

The final consideration for market based allocation is the trading environment (or motivation for trading). In a cooperative environment participants aim to achieve collective benefits for all participants. For example load balancing aims to share load over all participants and agreement sharing aims to produce optimal execution time. Competitive environments are more common and model the way in which most economies operate, in these environments participants are self interested and aim to maximise their own utility regardless of the effect on other participants. Competition may exist between consumers, providers, or meta-schedulers. Most of the systems analysed in this chapter focus on maximising utility for the consumer and utilise competitive economies.

2.3.5 Quality of Service Taxonomy

QoS support is desirable in many application domains, particularly when economic models are used. Consumers pay for the resources used and they therefore expect to be delivered certain minimum levels of service. The QoS taxonomy leaf presented in Figure 2.9 is in part extended from [51].

At the simplest level QoS can be included in a system through accounting and billing procedures. In this model service provisions are verified and consumers are billed for service usage. Fault tolerance is also an example of QoS commonly supported by meta-schedulers, as many high level schedulers include capabilities to checkpoint, replicate, and reschedule or migrate jobs if services fail or service levels are not delivered. These are however specific forms of QoS. To model a range of application requirements some meta-schedulers provide the ability to deliver generic extensible QoS. A common way to represent QoS levels is by establishing SLAs between consumers and producers to outline specific requirements and obligations of a provision. Most SLA representations are XML based such as WS-Agreement [78] used in SORMA and DRIVE, or XML tickets used in Bellagio and Sharp.

While many meta-schedulers enable SLA creation, very few actually provide any form of guarantee. In the QoS taxonomy leaf soft agreements define a best effort approach to SLA guarantees, whereas hard agreements provide some form of guarantee. Nimrod/G, Tycooon, and Grid-Federation support a soft contract model, whereas Bellagio and SORMA utilise a hard model in which some aspects of the agreement are guaranteed. Systems such as Multi-Agent Federation and DRIVE make use of a progressive two phase architecture in which initial soft SLAs are renegotiated into hard binding agreements. After SLA establishment it is implicitly assumed the

SLA will be monitored and enforced. Due to the complexities involved very few systems explicitly monitor and/or enforce agreements. Monitoring can be performed by any party involved in the agreement (provider or consumer) however there is obvious incentive to cheat, third party monitoring functions can be used to ensure the validity of the monitoring results. SORMA is the most complete SLA enabled system available, however it has limited mechanisms to monitor and enforce SLAs.

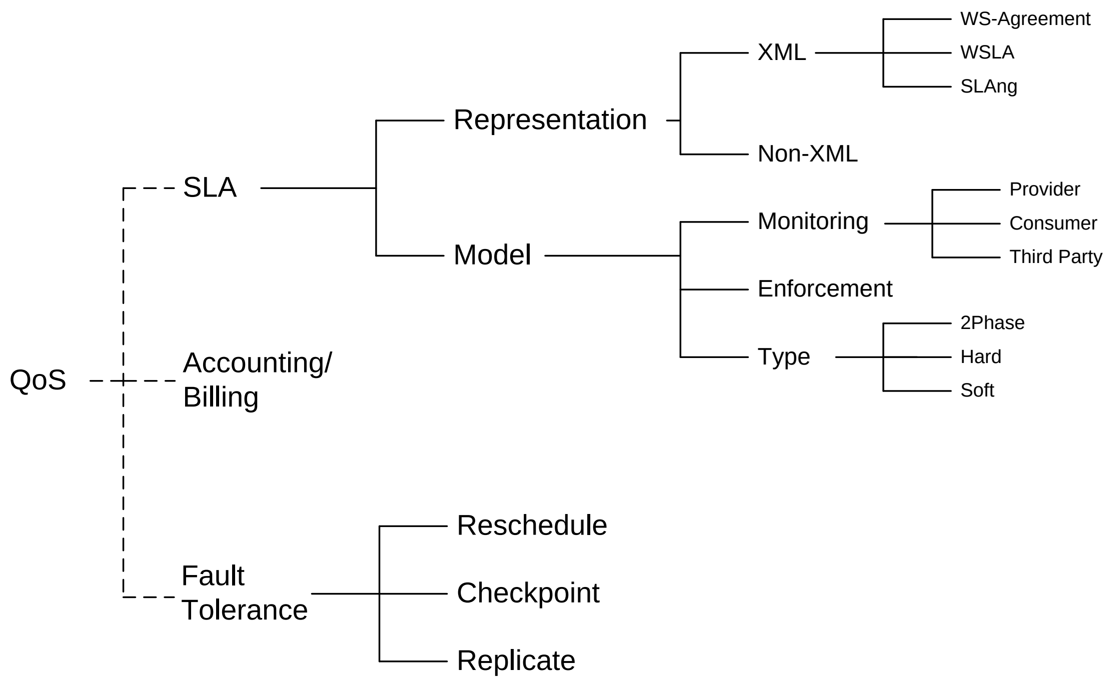


Figure 2.9: Quality of Service taxonomy.

2.3.6 Advanced Reservation Taxonomy

Figure 2.10 presents the expansion of the advanced reservation taxonomy leaf. Advanced reservations are used for planning and coordination of interdependent tasks. Various low level schedulers and LRMs support advanced reservation, however, it is not commonly supported at the meta-scheduler level. There are two general advanced reservation approaches available to meta-schedulers the first is to pass the reservation support on to a LRM, the second is to implement a new reservation architecture. In the case of a new reservation architecture reservations are defined as being flexible or non-flexible. Flexible reservations define a window in which the task must execute, whereas non-flexible reservations specify a time in the future for the job to start. Of the meta-schedulers considered only Tycoon, Sharp and SORMA define reservation capabilities.

Due to the lack of meta-scheduling reservation support various LRMs and schedulers have been surveyed to assess common approaches to reservation. GARA (Globus Architecture for Reservation and Allocation) [79] was one of the first projects to define a basic advanced reservation architecture supporting QoS reservations over heterogeneous Grid resources. Since this time other schedulers and LRMs have evolved to support advanced reservations, such as Catalina[4], Moab/Maui[5], SGE, LSF and PBS Pro. Advanced reservations in Catalina follow the GridForum Advanced Reservation API [80], supporting creation, modification and deletion of reservations through simple standardised mechanisms. Maui implements a queue based approach to advanced reservations, allowing a timeframe to be specified in which a job must be run. However once submitted to a queue users are unable to change any aspects of the reservation.

There have been some efforts in the meta-scheduler community to leverage advanced reservation capable scheduling. GridWay currently does not support advanced reservations, however there are plans in place to develop support through execution services. There are also third party designs which leverage GridWay to facilitate advanced reservation in economic frameworks [81]. CSF provides an advanced reservation service which interacts directly with reservation compliant LRMs. GridBus [82] has been used to create reservations on Aneka using the Alternate Offers Protocol to provide bilateral negotiation to establish guaranteed reservations [83]. In this architecture a Reservation Manager is co-located with a resource scheduler to manage reservations across a pool of executor nodes, the GridBus broker is then enhanced to negotiate with reservation aware negotiation clients. Finally, GridSim [84] has also been extended to support advanced reservation [85] allowing complex simulations and fine grained analysis of various reservation algorithms.

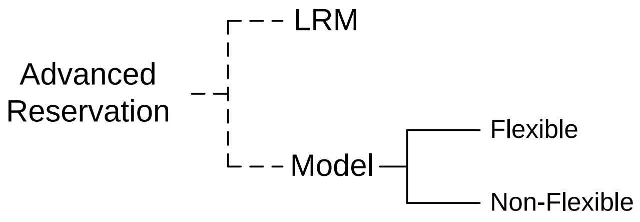


Figure 2.10: Advanced reservation taxonomy.

2.3.7 Registration Taxonomy

Managing membership when spanning administrative domains is necessary to authenticate and authorise actions within the Grid. Figure 2.11 outlines the registration leaf of the root taxonomy. There are two approaches to VO membership management in federated architectures; firstly a

[4]`http://www.sdsc.edu/catalina/`
[5]`http://www.clusterresources.com/pages/products/maui-cluster-scheduler.php`

new proprietary registration/membership architecture can be created, or secondly existing registration architectures can be federated. Traditional centralised system registration relies on local file or database based mechanisms to store and check user credentials. In distributed domains these approaches can still be applied however there are some limitations, including: scalability, lack of mechanisms to imply global (or VO) credentials, and difficulties making access decisions. Both Bellagio and Multi Agent Federation rely on database-based registration mechanisms. Offline registration can be employed to support registration of distributed users, in this approach Grid administrators issue credentials for users manually.

Certificate authorities provide an automated way of issuing credentials that can be used to verify a users identity. Normally a registration authority is included in the certificate authority to allow users to register with a given CA and then obtain credentials which have been granted implied privileges within the Grid through the CA. Hierarchical CA models can be used to create a distributed or federated registration infrastructure in which entities can establish trust through the issuing CA. GSI supports chaining of CAs to provide certificates with global credentials from any CA.

In modern Grid systems VO membership services are used to register and authenticate users allowing VO based credentials to be used by a group of users. This has the effect of reducing resource mappings and improving scalability. Several VO management systems have been developed including the Community Authorisation Service (CAS) [86], PRivilege Management and Authorisation (PRIMA) [87] and the Virtual Organisation Membership Service (VOMS)[88, 89]. These systems add VO registration information to existing user credentials, therefore registering users with a particular VO. Resources can then define access control policies regarding both individuals and VO members. Both GridWay and the DRIVE prototype rely on VOMS to manage dynamic VO membership. SORMA uses a combination of Dorian and SAML to provide VO like registration, Dorian [90] provides VO functionality by leveraging existing organisation identity management and creating a federation of users over the Grid system. SAML is then used to exchange authentication and authorisation statements between domains.

2.3.8 Trust and Security Taxonomy

Distributed architectures that span administrative (security) domains create additional security complications not seen in centralised systems. There are a number of levels of security in a distributed environment including authenticating participants, authorising user actions, establishing trust between participants and meta-scheduling components, ensuring secure communications over insecure networks and securing platforms and tasks from one another. The heterogeneity of resources, users and security infrastructure along with the decentralised nature of management architectures requires the use of specialised security infrastructure and protocols. Figure 2.12 presents the trust and security taxonomy leaf, this leaf is based on the taxonomy presented in [18].

Establishing trust in particular components or mechanisms is a difficult task, most often a

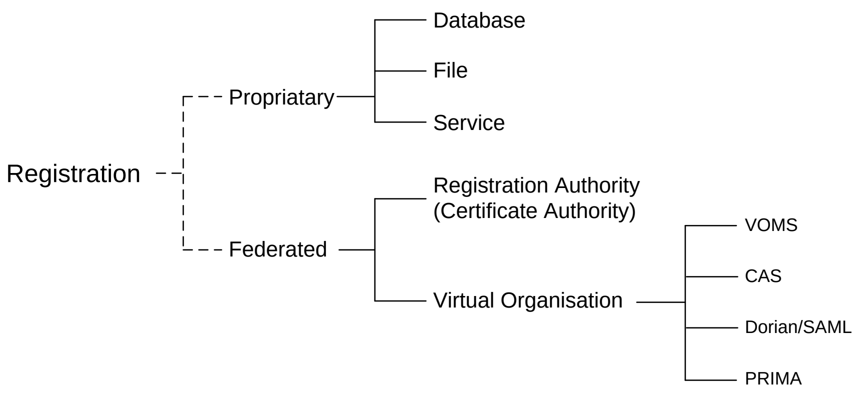


Figure 2.11: User registration taxonomy.

trustworthy infrastructure is required. In many situations however, it is unreasonable to build large scale secure infrastructures for all components. Trust must therefore be established in different ways such as using a trust anchor, gaining implied trust through a users previous actions, or enforcing trust through the mechanisms and protocols used. Some services (such as DNS and MDS) rely on a trust anchor to ensure trustworthy information, in this model the top level of the hierarchy is hosted on dedicated infrastructure which therefore provides a trust base for services hosted elsewhere. Reputation of entities can be used to infer trust, establishing ones reputation can be based on previous interactions between individuals or can be aggregated through a third party reputation services [91]. Trustworthy mechanisms and protocols can be used to ensure trust in the actions performed, for example the DRIVE prototype includes secure auction protocols which provide guarantees over the allocation process. Generally these protocols rely on encryption or distributed threshold based mechanisms [92, 93].

Distributed environments are inherently insecure as resources host potentially unsafe applications, users run applications on potentially untrusted resources, and communication is often over public networks. Several architectural mechanisms can be employed to minimise security risks. For example resource providers typically "sandbox" user tasks or host applications within limited scope VMs. Sandboxing is used in OurGrid and most Cloud providers define restricted VMs. From a users perspective tasks can be replicated to ensure validity of results and encryption can be used to safeguard data. Communication between tasks and/or entities commonly make use of secure messaging techniques such as SSL(Secure Sockets Layer)/TLS (Transport Layer Security). Tycoon, DRIVE, GMarte, and SORMA all use secure messaging techniques.

Like centralised systems accountability relies on authorising actions based on the identity of users (or VOs) established through authentication. The simplest mechanism used for authentication is username and password pairs, however this approach is susceptible to compromise. In Grid environments providers cannot maintain mappings for every user in the system, therefore users/passwords must be group based which in turn removes accountability as providers can no

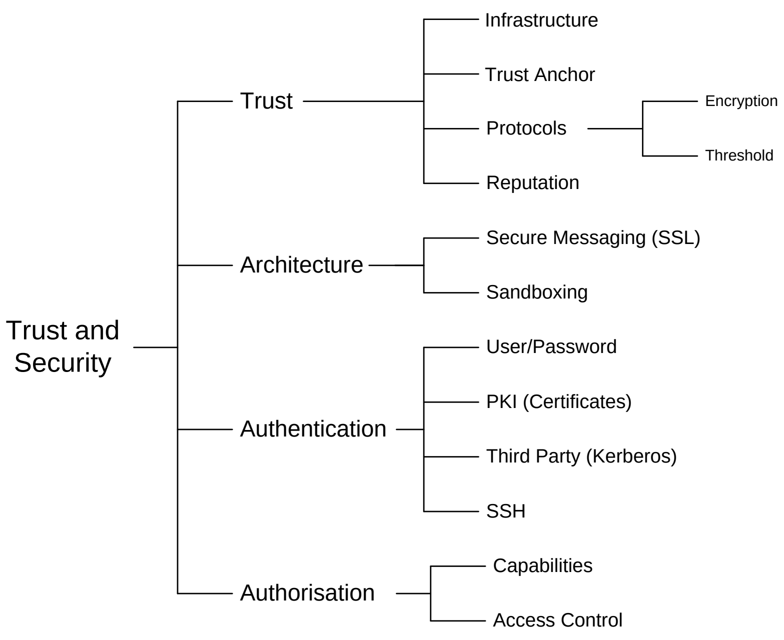


Figure 2.12: Trust and security taxonomy.

longer identify individual users. Authenticating users in Grid environments is most often based on X.509 certificates which in turn use Public Key Cryptography (PKI). In the standard model users obtain certificate based credentials which have been signed by a trusted CA, providers then infer trust in the user as they trust the CA. Of the meta-schedulers surveyed Nimrod/G, OurGrid, Emperor, CSF, GridWay, GmarteGS and SORMA all use certificates for authentication. Third party authentication mechanisms such as Kerberos can also be used to authenticate users. Kerberos is a network authentication protocol which allows users to prove their identities securely using symmetric key cryptography or PKI. The limitation with Kerberos is the single point of failure and requirement for continuous availability of the server to function. Secure Shell (SSH) based authentication is used in various Cloud offerings and PlanetLab as it does not require trusted third parties. SSH is based on PKI to authenticate users based on their public/private key.

Authorisation in distributed environments uses similar mechanisms to centralised systems. Two options are used to authorise actions, either providers maintain access control lists (ACL) specifying permitted user actions or users maintain capabilities which provide access when presented. In general most systems make use of ACL due to simplicity and the fact it allows providers to implement their own access policies and revoke rights instantly. Grid-map files used in Globus Grids are ACLs. Some VO based credentials include capabilities, for example VOMS allows capability specification at the VO level.

2.3.9 Evaluation Taxonomy

While the means of evaluation is not relevant to the architecture or properties of the meta-scheduler it provides an insight into the focus of the development and also presents criteria which can be used to compare and evaluate DRIVE. Figure 2.13 presents the evaluation taxonomy. In general meta-schedulers are evaluated in one of four ways: simulation, testbed deployment, production deployment, and application based experimentation. The most common means of evaluation is through simulated deployments using platforms like GridSim [85], this approach is the easiest way to generalise results and create large scale test platforms. However it does not examine the entire architecture and can only be used to analyse the performance of mechanisms and algorithms. Application focused evaluation is used to evaluate meta-schedulers through deployment and experimentation of a specific application example. Testbed deployment allows experimental results to be gathered within a closed environment, this provides the easiest way to evaluate the performance of the meta-scheduler as a whole. In a testbed environment it is difficult to analyse scalability and high performance situations due to the limited size. Finally production deployment provides the most complete analysis as results can be obtained from a real world environment in which real workloads are submitted by active user groups.

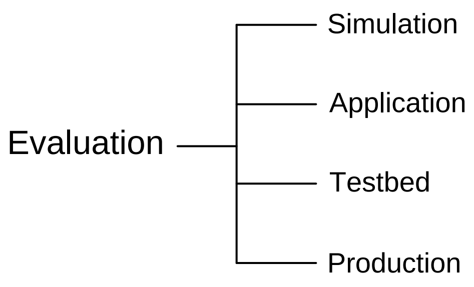


Figure 2.13: Evaluation taxonomy.

2.4 System Classification

Tables 2.1, 2.2, 2.3, 2.4, 2.5, and 2.6 summarise the classification of each of the meta-scheduling approaches surveyed in this chapter according to the taxonomies presented. Each system considered is also summarised in Section 2.5.

2.5 Specific Systems

The remainder of this chapter presents a detailed survey of a range of meta-scheduling architectures. The systems described in sections 2.5.1– 2.5.13 may be designed for, or operate in, differ-

System	**Architecture**				**Discovery**
	Host	**Dedicated**	**Middleware**	**Deployment**	
SORMA (System)	Web Service (WS/WSRF)	Yes	Service	Multi-VO	Decentralised P2P
GridWay (MS)	Application (GWDaemon) Web Service (WSRF)	Yes	Generic MAD (GRAM WS&PreWS, LCG,..)	Consumer or VO	Generic MAD (MDS1 &3)
Grid/Alchemi Federation (System)	P2P Application	No	Non-Service (Alchemi)	Consumer or Provider	Decentralised P2P
CSF (CS)	Web Service (WSRF)	Yes	Service (GRAM) Non Service (PreWS GRAM, LSF)	VO	Decentralised (MDS3)
Nimrod/G (Broker)	Application	Yes	Non-Service (Globus, Legion, Condor)	Consumer	Centralised (MDS 1/2)
Tycoon (System)	Application(C)	Yes	Service (VMs)	Consumer	Centralised (SLS)
Bellagio (System)	Web App and Service	Yes	Non-Service (PlanetLab)	VO	Decentralised P2P (SWORD)
SHARP (Framework)	Application (Python)	Yes	Non-Service (PlanetLab)	Consumer or VO	Decentralised (MDS3) and P2P (using tickets)
EMPEROR (MS)	Web Service (OGSA)	Yes	Service (GRAM)	VO	Decentralised (MDS3)
GMarteGS (MS)	Web Service (WSRF)	Yes	Service (GRAM) Non-Service (Pre WS GRAM, PBS, SGE)	VO	Centralised (MDS1) Decentralised (MDS3)
Trader-Federation (System)	Service (CORBA)	Yes	Service CORBA	VO	Decentralised (CORBA/ODP) P2P between traders
Multi-Agent Federation (Arch)	N.A	N.A	N.A	Not Bound	Centralised
OurGrid (System)	Service (XMPP)	Yes	Desktop and Clusters through LRM	N.A	Centralised (Extending to P2P NodeWiz)
DRIVE (MS)	Web Service (WSRF)	No	Generic (Prototypes GRAM & Service)	Not bound	Decentralised (MDS3)

Table 2.1: System architecture and discovery classification. Each system also specifies its classification: Architecture (Arch), Broker, Community Scheduler (CS), Framework, Meta-scheduler (MS), System.

System	**Task Model**		**Allocation Architecture**	**Matchmaking**
	Type	**Description**		
SORMA	Job	XML (EJSDL)	Centralised (Decentralised but Coordinated through central Trading Service)	Market
GridWay	Job (Sequential/Parallel) or API (DRMAA)	XML (JSDL) Non-XML (GW Job Template)	Decentralised Non-Coordinated	Queue
Grid/Alchemi Federation	Alchemi Thread/Job	XML or API	Decentralised Coordinated	Market
CSF	Job	Non-XML (RSL)	Decentralised Non-Coordinated	Queue
Nimrod/G	Job (Sequential/Param Sweep)	Non-XML (Nimrod/G planfile)	Decentralised Non-Coordinated	Market
Tycoon	Job	Non-XML	Decentralised Non-Coordinated	Market
Bellagio	Service Request	Non-XML (XOR Language)	Centralised	Market
SHARP	Service Request	XML (Sharp Ticket)	Decentralised Coordinated	Market
EMPEROR	Job	Non-XML (RSL)	Centralised	Static
GMarteGS	Job	XML	Centralised	Static
Trader-Federation	Object/Service Request	N.A	Decentralised Coordinated	Static
Multi-Agent Federation	Job	Non-XML (Meta-SLA request)	Decentralised Coordinated	Market
OurGrid	Job (Parallel)	Non-XML (Job Description File)	Decentralised Coordinated	Dynamic Sharing (Reputation)
DRIVE	Generic (Prototypes Job & Service request)	Generic (Prototypes XML - JSDL, EJSDL)	Decentralised Coordinated	Market

Table 2.2: Task and allocation classification.

System	Economic Model	Participant Focus	Trading Environment
SORMA	Generic (Auction Focus)	Consumer	Competitive
Grid/Alchemi Federation	Commodity Market	Consumer	Competitive
Nimrod/G	Commodity Market	Consumer	Competitive
Tycoon	Auction	Consumer	Competitive
Bellagio	Combinatorial Auction	Consumer	Competitive (Semi Cooperative)
SHARP	Bartering	Consumer	Competitive
Multi-Agent Federation	Commodity Market	Consumer	Competitive
DRIVE	Generic (Auction Focus)	Consumer Producer	Competitive Cooperative

Table 2.3: Market based allocation classification.

System	QoS	SLA Model	Representation	Advanced Reservation
SORMA	Accounting/Billing	Hard, Monitoring & Enforcement	WS-Agreement	Flexible
GridWay	Accounting/Billing Rescheduling Checkpointing	N.A	N.A	N.A
Grid/Alchemi Federation	Accounting/Billing	Soft	N.A	N.A
CSF	N.A	Soft	WS-Agreement	LRM
Nimrod/G	Accounting/Billing Rescheduling	Soft	N.A	N.A
Tycoon	Accounting/Billing	Soft	N.A	Non-Flexible
Bellagio	Accounting/Billing	Hard, Enforcement	XML Ticket	N.A
SHARP	Accounting/Billing	Two Phase	XML Ticket	Non-Flexible
EMPEROR	N.A	N.A	N.A	LRM
GMarteGS	Fault Tolerance	N.A	N.A	N.A
Trader-Federation	N.A	N.A	N.A	N.A
Multi-Agent Federation	Rescheduling	Two Phase	N.A	N.A
OurGrid	N.A	N.A	N.A	N.A
DRIVE	Accounting/Billing	Two Phase	WS-Agreement	Flexible

Table 2.4: Quality of Service and advanced reservation classification.

System	Registration	Trust	Architecture	Authentication	Authorisation
SORMA	CA, Dorian/SAML	Infrastructure	SSL	PKI (X509), SAML Delegation	SAML Assertion/ACL from server
GridWay	VOMS or CA (Unix credentials for daemon)	Infrastructure	N.A	GSI PKI (X509)	ACL (Daemon based on Unix)
Grid/Alchemi Federation	Alchemi Registration	Infrastructure	N.A	Alchemi Role Security (Username/-Password)	ACL
CSF	Offline	Infrastructure	N.A	GSI PKI (X509)	ACL
Nimrod/G	N.A	Infrastructure	N.A	GSI PKI (X509)	N.A
Tycoon	Tycoon Management	Infrastructure	SSL	SSH	Authorisation File
Bellagio	VO Database	Infrastructure	N.A	VO Database	N.A
SHARP	N.A	Protocols (PKI)	N.A	PKI (Tickets)	Capability (Ticket)
EMPEROR	Offline	Infrastructure	N.A	GSI PKI (X509)	ACL
GMarte Grid Service	Offline	Infrastructure	Secure Messaging	GSI PKI (X509)	ACL
Trader-Federation	N.A	Infrastructure	N.A	N.A	N.A
Multi-Agent Federation	Database	Infrastructure	N.A	N.A	N.A
OurGrid	Open Registration (Web)	Infrastructure Resource Reputation	Sandboxing	GSI PKI (X509)	N.A
DRIVE	VOMS/CA	Protocols & Infrastructure	SSL	GSI PKI (X509)	ACL

Table 2.5: Registration, trust and security classification.

System	**Evaluation**
SORMA	Simulation, Deployment, Application
GridWay	Large scale deployments (OSG, TeraGrid), Application, Testbed (Benchmarking, scalability, resource consumption, adaptive scheduling)
Grid/Alchemi Federation	Simulation (GridSim), Testbed (Overhead, response time)
CSF	Application
Nimrod/G	Simulation, Testbed, Application (deadline constraint)
Tycoon	Simulation, Testbed, Deployed Tycoon Grid (Analysed proportional sharing, scheduling error, fairness, latency)
Bellagio	Testbed, synthetic workloads (Efficiency and fairness)
SHARP	Testbed (Allocation time, resource efficiency, oversubscription)
EMPEROR	Virtualised Testbed (Scheduling, load prediction)
GMarteGS	Application
Trader-Federation	Simulations, Testbed (Load balancing)
Multi-Agent Federation	N.A
OurGrid	Deployment, Applications
DRIVE	Testbed, two prototypes, synthetic workloads (policies, efficiency, economic aspects, throughput, computational burden)

Table 2.6: Evaluation classification.

ent domains, however each includes specific shared goals or focuses with DRIVE. The systems considered include brokers, meta-schedulers, superschedulers, and general economic allocation architectures.

This section is organised according to the perceived relevance of each system to DRIVE. SORMA is perhaps the most similar to DRIVE as it facilitates resource trading in an open market and has a strong contract management architecture. GridWay is the most commonly used and mature meta-scheduling architecture available, it focuses on high performance static allocation using a modular architecture, implements a number of standards, and supports a wide range of Grid providers. Grid Federation uses a P2P based architecture to create dynamic federations. CSF shares a similar co-op deployment model to DRIVE using services hosted by the community to allocate resources. Nimrod/G, Tycoon, Bellagio and SHARP provide a range of economic focused schedulers implementing different market abstractions, the latter two share a unique contract structure using tickets which is similar to the contract architecture used in DRIVE. EMPEROR has an extensible plug-in architecture designed to analyse resource prediction models. GMarteGS is an example of extension of an existing meta-scheduler to a distributed Grid service infrastructure. Finally Trader-Federation, Multi-Agent Federation and OurGrid differ significantly from DRIVE in terms of domain, architecture, and goals. However each has interesting architectural properties which may be valuable in a DRIVE context and exercise the taxonomy presented in the previous section.

2.5.1 SORMA

SORMA [53] (Self-Organising ICT Resource Management) implements a complete open Grid market consisting of self-interested resource brokers and user agents. Like DRIVE, the SORMA market facilitates dynamic trading of compute resources and services on demand through economic protocols. Users are presented with complete transparency as they submit tasks without knowledge of the internal allocation or execution mechanisms. SORMA includes complex valuation functions and bidding policies to achieve high quality allocation. A thorough SLA architecture is provided including WS-agreement establishment, monitoring, and enforcement components. Unlike other economic resource allocation projects, focus in SORMA has also been given to the business aspects of the system with development of sustainable and customisable business models.

In the market consumers are represented by *Buyer Agents* and providers by *Seller Agents*. Bids and offers are configured dynamically using economic aspects, policies, and technical input. Resource allocation uses an auction mechanism with winning negotiations turned into binding contracts. In the market competing consumers discover services and place bids. Discovery of services makes use of a decentralised information bus implemented as a P2P overlay network. The market components determine if bids match existing offers to ensure required levels of service can be delivered. The auction result is passed to the contract management components to create contracts from matching pairs of bids and offers.

The SORMA implementation is based on raw resource trading and makes use of EJSDL (JSDL with economic extensions) to describe jobs. Contracts are expressed in WS-Agreement using EJSDL as the term language. A proprietary SORMA reservation specification is used to capture the period of provision. Due to the size and scope of SORMA different approaches have been taken in each of the components, for example components differ in terms of architecture and interface. Some components are implemented as web services (Axis, WSRF, JAX-WS) while others are implemented as proprietary applications.

The security infrastructure uses SAML and Dorian to manage VO membership. Dorian provides a mapping between external security domains and SORMA which allows accounts managed in external domains to be federated and managed in SORMA. Users can therefore use their existing credentials to authenticate with SORMA. SAML provides distributed identity management and single sign on by allowing the exchange of security messages and interoperating with other authentication authorities. Using SAML and Dorian, SORMA consumers can authenticate in their own organisation, obtain a SAML assertion, and use this assertion in exchange for SORMA credentials (proxy certificate). This architecture essentially creates a dynamic VO in which users can use domain specific credentials in the wider Grid community.

The core SORMA architecture can be conceptually thought of in three areas: consumer components, provider components, and the open market. Consumers have access to a range of tools aimed at simplifying interaction with the market, including performance modelling, demand modelling and bid generation capabilities. A SORMA API provides access to these mechanisms to login, bid, submit payments, execute jobs and monitor execution. Resources are represented by an EERM (Economically Enhanced Resource Manager) which calculates prices for jobs based on various metrics (client, resource status/predictions, and policies). A System Performance Guard acts to monitor SLA violations and may take actions to suspend or terminate jobs depending on real time performance. Semantic Web Rule Language (SWRL) is used to represent adaptive policies. Within the EERM an Economic Resource Manager (ERM) interacts with LRMs using plug-in Tycho connectors, these connectors perform various tasks, for example mapping the JSDL description to the native representation before submitting to the LRM scheduler. The open market is composed of multiple interacting services including the Market Exchange, Market Information, Payment, Trading Management, Contract Management, and Billing.

SORMA has been designed with similar goals to DRIVE and as such their are a number of similarities between the architectures. However they differ in respect of the market, federation capabilities, implementation architecture, and protocol focus. SORMA aims to create a complete market infrastructure imposing architecture, tooling and protocols on providers and consumers, whereas DRIVE aims to facilitate federation over different existing providers therefore relaxing the requirements on joining the market. SORMA is built from different service and non-service based components hosted on dedicated infrastructure while DRIVE is designed with scalable standardised services for every component and has minimal infrastructure requirements due to the novel co-op architecture and support for trustworthy protocols. While both SORMA and DRIVE support different economic protocols the participant focus differs, SORMA focuses on resource auctions where consumers compete for resources, whereas the DRIVE prototype utilises reverse auctions facilitating provider competition for a consumers task. Much of the research in SORMA on business strategies, policies and valuation functions is equally valuable in a DRIVE context, however this is outside the scope of this thesis.

2.5.2 GridWay

GridWay [19, 20, 21] is a lightweight client side meta-scheduler which facilitates large scale resource sharing across administrative domains. GridWay relies on a modular architecture composed of *Middleware Access Drivers* (MAD) which facilitate interaction with arbitrary information services and execution components. Initial versions of GridWay have used Globus middleware for discovery (MDS) and execution (GRAM), however it has since been extended to interact with a wide variety of Grid middleware (for example Globus WS and pre-WS, LCG, ARC, CREAM). The central component of GridWay is a client side (or VO) daemon process that abstracts job submission through a command line interface or programming API using DRMAA (Distributed Resource Management Application API) [74]. In addition most functionality is also exposed through a WSRF interface called GridGateway. GSI security is used for authentication and resource access. User registration and management depends on the interface used: the daemon process relies on UNIX user authentication and groups to authorise actions, while the service implementation is capable of using Grid/VO credentials embedded in standard Grid proxies.

Gridway provides decentralised resource management in which users (or VOs) interact with a GridWay service to schedule tasks over distributed resources. The major GridWay component involved in this process is the *Submission Agent* (SA) that performs job submission and monitoring. The SA contains several modular components including a Request Manager, Dispatch Manager, Submission Manager and Performance Manager. Client applications (or CLI commands) communicate with the Request Manager to submit a job, the job is defined using the API, configuration file or job template (specified in GridWay format or JSDL). The Dispatch Manager then retrieves a list of prioritised resources (from information services via a resource selector) and submits the job to the specified resource by invoking the Submission Manager. The Submission Manager prepares the executable and host environment and then executes the job (in a GridWay wrapper) on the designated resource. The modular architecture facilitates runtime module selection on a per job basis for, amongst other things, resource selection, middleware access, and performance evaluation. This architecture allows users to choose appropriate information services and middleware execution environments.

Arbitrary scheduling policies can be implemented by consumers and providers. For example, supported job prioritisation policies include fixed, fair share, urgent, waiting time, and deadline policies whereas resource prioritisation policies consider fixed, ranked, usage, and failure rate policies. GridWay provides fault tolerance and adaptive scheduling by periodically monitoring jobs to ensure performance. In the event of poor performance adaptive scheduling techniques reschedule the job to better suited resources. There is no explicit SLA definition or advanced reservation support, however these features could be built on top of the GridWay architecture. Gridway has been deployed, used, and evaluated in various large scale distributed environments including EGEE, OSG and TeraGrid.

GridWay is arguably the most commonly used and mature meta-scheduler available for performing resource allocation in Grid systems. However the design of GridWay differs significantly when compared to DRIVE. In particular GridWay does not support economic allocation which would make it infeasible to use GridWay to create a federated system incorporating commercial providers. GridWay is deployed following a centralised dedicated deployment model rather than the co-op model used in DRIVE. In addition it is a best-effort scheduler with no SLA support or reservation architecture. The modular architecture used in GridWay is similar to DRIVE and supports protocol extensibility and arbitrary resource provider implementation.

2.5.3 Grid-Federation/Alchemi-Federation

Grid-Federation [17, 18] facilitates policy based resource sharing between clusters, therefore creating a cooperative federation of clusters. The computational economy from Nimrod/G is employed to provide deadline and budget based allocation. Grid-Federation models a highly dynamic environment in which resource state may change frequently, if local resource providers cannot meet the requirements of a task at execution time, the task is transparently migrated to another resource in the federation. A *Grid Federation Agent* (GFA) is used to manage resources at the provider level allowing transparent resource owner defined policy based sharing. Each GFA advertises current usage costs in the *Federation Directory* which is then exposed through a computational economy. Providers join the federation by instantiating a GFA, the GFA coordinates resource access within the entire federation. GFAs interoperate using a communication protocol over the shared federation directory. A *Distributed Information Manager* (DIM) is used to retrieve information for resource discovery and advertising, in the architecture this is P2P based [69]. The directory contains quotes from GFAs in the federation outlining resource descriptions and usage costs. Using this information a price can be calculated and it can be determined if it is best to host the job locally or host it in the federation. The GFA includes a GRM (*Grid Resource Manager*) which hosts queues for both local jobs and jobs submitted from the federation, if necessary the GRM can export jobs from the local queue to the federation queue (migration). In the case of federation a one-to-one proprietary SLA negotiation takes place between the local and remote GRM.

A proof of concept Alchemi-Federation has been implemented to federate clusters of Alchemi [94] desktop Grids. Functional C# implementations of the GFA and GRM components are included along with connectors to access the Alchemi Manager and Grid Peers (information manager). The P2P based resource discovery service is implemented as a Pastry [95] overlay network over the GFAs, maintaining a publish/subscribe index for decentralised resource discovery. The discovery mechanisms are exposed through Java based Web services allowing the GFAs standardised access to query and update the discovery index.

Grid-Federation uses a completely different architecture from DRIVE to support distributed resource provider federation. Although economic principles are used to match tasks with resources the major focus of this work is the P2P protocols used for discovery of resources and offers. While the P2P architecture distributes the burden of discovery and provides autonomy, there is no means of trust. The economic protocols used are inflexible and reliant on Nimrod/G. Additionally Grid-Federation does not create standardised SLAs, rather it uses a proprietary language and protocol.

2.5.4 Community Scheduler Framework

The Community Scheduler Framework (CSF) [14, 15, 16] is an open source framework for implementing community meta-schedulers using a collection of WSRF Grid services. CSF relies on standard GT4 services for resource discovery (MDS) and execution (GRAM). CSF provides a queuing service where submitted jobs are assigned to resources by applying queuing policies, the scheduling mechanisms are user defined and implemented using a plug-in architecture similar to DRIVE. Advanced reservation is supported through a proprietary reservation service which interacts directly with reservation compliant LRMs (it bypasses generic execution services). Like DRIVE, the architecture is composed of several web services: the *Job Service* provides an interface for users to create, submit and monitor jobs. The *Queuing Service* allows users to submit jobs to a queue which schedules jobs according to available (plug-in) scheduling policies (FCFS and throttle based). *Resource Manager Services* (RMS) abstract provider differences and are designed to support alternate execution protocols, for example CSF includes implementations for LSF and also Pre-WS GRAM. Job definitions are specified in RSL and MDS is used to discover resources.

Although the architecture is distributed there is no trust established therefore the system requires dedicated or trusted infrastructure to avoid compromise. The flexibility of CSF is shown in [16] where plug-in components are added to schedule domain specific applications efficiently. CSF supports WS-Agreement as a means of negotiating the use of resources through the RPS, while the Reservation Service implements the required port types there does not appear to be a complete implementation of WS-Agreement [96]. The co-op architecture used in DRIVE is similar to the community metaphor used in CSF, however CSF does not establish trust in the protocols or components used which therefore limits potential deployment environments. CSF does not support economic allocation techniques instead it relies solely on queue based allocation.

2.5.5 Nimrod/G

Nimrod/G [40, 41, 54] is a Grid resource broker designed to support deadline and budget constrained market based scheduling. Nimrod/G is a Grid-enabled version of Nimrod [97] as such it allows users to execute parameter sweep applications on a Grid. A competitive commodity market is used to allocate resources, the Nimrod/G broker (representing a user) retrieves market prices from individual resource providers represented by an agent (*Grid trader*). QoS constraints (deadline and budget) can be specified by users in a Nimrod/G planfile, provider prices then reflect the cost of resources required to supply the specified QoS. Nimrod/G brokers are independent entities with no communication or coordination between them, the topology forms a competitive trading environment between brokers. Pre web service Globus components are used for discovery (MDS1/2), execution (GRAM) and security (GSI). The broker is implemented as a standalone application utilising TCP/IP communication sockets. As the broker is designed to represent a single user it is assumed to be deployed in a trusted environment. Evaluation has been focused on application performance (parameter sweep), testbed deployment, and simulation results analysing deadline and budget scheduling. The GridBus Broker [42] extends Nimrod/G to take a data centric approach to scheduling, in particular it is able to access remote data repositories and optimise allocation based on data transfer.

2.5.6 Tycoon

Tycoon [55] is a market based distributed resource allocation system in which providers auction resources to consumers. Each provider hosts an independent auctioneer to conduct a real-time (first-price) auction for their own (virtualised) resources. Resources are then allocated to users in proportion to their bids. Auctioneers (and therefore providers) are completely independent with no coordination between one another. Consumers bid for resources in a competitive market, specifying deadline and budget requirements. The design implicitly assumes a generic service model abstraction (with focus given to a VM approach) where service requests model simple QoS requirements such as deadline and budget. Proprietary centralised discovery services called *Service Location Services* (SLS) are used to store and retrieve auctioneer information. Application level agents/schedulers (representing consumers) retrieve updated information from the SLS, agents then bid to individual auctioneers for resources. Consumers can specify the focus of their job (latency, throughput) before they compete against one another for the resources on offer through the auction process. Tycoon creates soft SLAs defining the allocated resources and their proportion. Non flexible reservations are also supported. Tycoon has been deployed as linux VServer and Xen [98] based prototypes with a local proportional share scheduler and auctioneer operating on each (virtualised) host. At present allocation only considers CPU cycles due to virtualisation limitations. Evaluation of Tycoon focuses on proportional sharing and allocation using user defined deadline requirements, in particular analysis of scheduling error, fairness and latency have been presented.

2.5.7 Bellagio

Bellagio [56] is a resource discovery and allocation system designed to allocate resources in a federated environment. Bellagio has been designed for use with non-service based PlanetLab [65], and where possible leverages existing PlanetLab services and functionality. Like DRIVE, Bellagio supports combinatorial auctions for resource allocation. In Bellagio, consumers discover resources from a decentralised discovery system and express preferences over time and space through combinatorial bids. Resource allocation relies on a centralised dedicated auctioneer. A strategy proof design is used to provide incentives for consumers to bid their true values. Interestingly the approach aims to allow cooperation between consumers to arrive at optimal allocation by spreading excess demand over time and providers. This approach has been taken to overcome the limitations of proportional share scheduling in competitive environments [56]. Bellagio uses Scalable Wide-Area Overlay-based Resource Discovery service (SWORD) [70] to perform distributed P2P resource discovery based on attributes. The XOR bidding language [99] is used to describe resource preferences in the form of a combinatorial bid. A greedy approximation algorithm determines a set of winners efficiently (not necessarily optimally). SHARE [100] is used for resource allocation and employs the threshold rule [101] to determine a second price (or discount) in combinatorial auctions, this approach gives the same incentive for truthful bidding as the non-approximate method used in DRIVE second-price combinatorial auctions. In Bellagio, a *ticket* is created as the result of an allocation, the ticket contains a list of resource capabilities (rights to use a resource) and must be presented to the resource in order to use it. Essentially this ticket is a binding SLA which is similar in nature to the WS-Agreement SLAs employed by DRIVE.

2.5.8 SHARP

SHARP [57] (Secure Highly Available Resource Peering) is a framework for providing secure resource management in large scale distributed infrastructures. It provides policy-based resource management allowing resource sharing and reservations using a decentralised broker. Individual providers trade resources with peering partners or contribute resources to a federation. Brokers control potentially overlapping groups of resource providers. Allocation in SHARP uses a simple negotiation/bartering process where consumers request access to resources. Service providers are represented by *site agents*, who issue resource *tickets* to consumers according to policies. *User agents* create a second tier market able to collect tickets for an organisation and distribute them amongst members. Additionally brokers trade tickets and index resources by attributes to match resources to requirements. Brokers propagate tickets following distributed resource discovery frameworks. A link state (P2P) routing protocol is used to share ticket information amongst agents. This network of brokers supplements hints provided from traditional discovery systems like MDS3. SHARP includes a two phase ticket structure (similar to DRIVEs two phase contracts). Soft state timed *claims* are generated when consumers request resources, these claims expire over time to free up unclaimed resources (no-shows). *Leases* are issued by site agents when claims are presented by consumers and represent a hard claim over concrete resources. Tickets are self describing and self-certifying removing the need for certificate chains. The SHARP broker is implemented as a standalone trusted Python application and is designed to provision VMs in PlanetLab. Node managers create a VM (using VServers) when a ticket is redeemed. To join the SHARP system sites negotiate contracts outlining the expectations of one another including the promise of providing resources.

2.5.9 EMPEROR

EMPEROR [58] is a Grid meta-scheduler framework designed to analyse the performance of dynamic scheduling algorithms and accuracy of resource prediction models. The architecture supports arbitrary scheduling algorithms and includes multiple models for predicting host load and memory resources used to estimate the run time of a job. The centralised service based implementation uses Open Grid Services Architecture (OGSA) services and makes use of common Globus services for tasks such as security (GSI), resource discovery (MDS) and job execution (GRAM). Jobs are described in non-XML Globus RSL therefore removing the need to map description languages. Advanced reservation is also supported if the underlying LRM is reservation capable. Resource information is constantly disseminated from *Resource Prediction Systems* (RPS) running on each host. This information is periodically sent to MDS which is then in turn used for load forecasting. EMPEROR measures utilisation and memory usage and creates predictor models based on statistical characteristics of usage, this information is combined with job profiling techniques [102] to create an estimated run time of a job on a potentially heavily loaded host.

2.5.10 GMarte Grid Service

GMarteGS (GMarte Grid Service) [59] is an example of an existing application-based meta-scheduler (GMarte) that has been Grid enabled by adding a WSRF interface and integrating common Grid services and tools. GMarteGS is a complete functional meta-scheduler that includes job submission, scheduling, data transfer, job execution and monitoring. The underlying GMarte [103] meta-scheduling framework is aimed at simplifying Grid usage for scientific applications. Originally GMarte was developed as a standalone Java library which exposed a user API for application deployment. GMarteGS includes a layered architecture with access to disparate Grid information services and LRMs through GRAM. The WSRF extensions act as a wrapper to create a centralised WSRF meta-scheduler. An abstraction layer (*Grid Resource*) is used to combine different information sources (LDAP, MDS2, MDS4), this consolidated information is then used to make scheduling decisions. A plug in scheduling (resource selection) policy interface is provided to facilitate arbitrary user defined policies, the implementation includes policies to minimise execution time and data transfer but does not consider any further QoS metrics. WS-Resources are used to represent user *sessions* for individual users allowing stateful management of job execution. Like DRIVE, state is stored in WS-ResourceProperties and WS-notifications can be used to retrieve state changes.

2.5.11 Trader-Federation

Trader-Federation [60, 61] aims to improve resource allocation and load balancing in an object-oriented distributed environment. The distributed federation model uses CORBA (Common Object Request Broker Architecture) and RM-ODP (Reference Model for Open Distributed Processing). In this model a *trader* is used to mediate between consumers and providers, each trader represents a group of local users, clients and servers. This grouping is similar to a VO. The trader is implicitly trusted and therefore requires dedicated or trustworthy infrastructure. In a large scale environment it is noted that a single trader is not sufficient, federation is proposed to allow traders to work together to increase the opportunities presented to providers and consumers. Traders cooperate to maximise a trading function, focusing on load balancing across the federation. Within the federation a simple negotiation marketplace is created for traders to negotiate user requested QoS levels. Traders maintain offers from providers that are used to determine if a request can be satisfied. Requests are propagated using a *trading graph* which specifies trading paths between traders. Like P2P mechanisms, if a trader cannot satisfy a request locally they forward it to other traders within the federation following the trading graph. CORBA facilitates run time discovery of services and objects using a centralised trader service. Unlike DRIVE, trader-federation does not create an open market, rather it creates a decentralised mechanism for trading agents to exchange requests in CORBA systems.

2.5.12 Multi-Agent Federation

The Multi-Agent superscheduler [62] uses a group of cooperating agents to efficiently schedule jobs on distributed resources. A unique multi-stage hierarchical SLA negotiation protocol based on Contract Net [104] is used to allocate resources. There are three types of agents in this system: *User agents*, *Local Scheduler* (LS) agents and *SuperScheduler* (SS) agents. User agents represent consumers and negotiate SLAs with SS agents. LS agents represent resources essentially acting as a resource provider. LS agents schedule jobs assigned by SS agents. SS agents provide distributed allocation functionality, acting as mediators between user agents (consumers) and LS agents (providers). Databases are used to store (and discover) resource state information, SLA information, and executing job information.

Two SLA levels are defined: Meta-SLA negotiation occurs between SS agents and user agents to define high-level descriptions of the job (resource information, deadlines, budget), Sub-SLA negotiation occurs between SS agents and LS agents by decomposing the meta-SLA into low level raw resource requirements (processors, memory, disk). User agents construct and send meta-SLAs to SS agents (as a job description), the SS agent forwards the meta-SLA to neighbouring SS agents. Each SS agent queries known LS agents for availability, SS agents then select the best resources to satisfy the job requirements and submit a bid. The user agent in turn selects the best SS agent bid. Having been awarded the agreement the SS agent creates sub-SLAs and negotiates with local LS agents to host the job. This creates a two tiered market structure with both wholesale (LSA-SSA) and retail (UA-SSA) markets. The mediator (SSA) agents map high level user requirements to low level provider capabilities in these two markets. Competitive commodity markets are used to match requirements with capabilities. This two phase SLA model is very different to DRIVE, here two types of SLAs are created between consumer/mediator, and mediator/producer, assuming an agreement is reached between SS and LS agents then the second phase of negotiation occurs between LS and user agents. DRIVE uses a two phase model to represent the allocation process by hardening an initial agreement through an additional SLA negotiation phase.

2.5.13 OurGrid

OurGrid [63, 64] is a federated cooperative P2P architecture, in which users share resources amongst an open distributed community. OurGrid is free to join assuming users contribute resources to the system. A unique network of favours (reputation) model is used to allocate resources. Essentially peers prioritise usage based on previous interactions, that is, entities that share the most are prioritised when requesting resources. This is a similar architecture to the credit based system used in the Social Cloud DRIVE prototype (Chapter 8). Resource providers are represented by an *OurGrid peer*, which participates in P2P based scheduling. Consumers interact with OurGrid using a broker (MyGrid) or public web portal. Jobs are described using a proprietary (non-XML) language called Job Description Files (JDF). Every resource hosts an *OurGrid worker* or SWAN (Sand-boxing Without A Name) service, to provide a VM (VMWare, VServer) based application hosting environment, this has the advantage of providing a trustworthy sandboxed environment to execute user jobs. A centralised discovery mechanism is used for peers to register and discover one another, due to the scalability issues they intend to replace it with a decentralised P2P request propagation system in the future (NodeWiz [105]). The most recent version of OurGrid relies on an XMPP (Extensible Messaging and Presence Protocol) server for communication and hosting of peers. Unlike most distributed platforms, OurGrid does not provide QoS guarantees, in this respect the platform provides best effort sharing (similar to Volunteer computing). The lack of QoS specification means there is no need to value resources or create SLAs. In the absence of performance prediction, OurGrid uses replication to optimise performance. Users submit applications (typically bag of tasks) to an OurGrid broker, the broker forwards the request to the pool of registered OurGrid peers. The peers respond to the query depending on availability and reputation of the consumer's site. Assuming there is availability, peers dynamically provision workers from the pool of available resources to host the user application.

2.6 Summary

A wide range of meta-schedulers have been developed with differing focuses and goals. In a Grid, LRMs are responsible for managing workload for individual clusters situated within a single administrative domain. Meta-schedulers aim to provide transparency for users when discovering and submitting tasks to distributed resources, matching job requirements with resource capabilities and allocating tasks using a variety of techniques including queuing, Quality of Service (QoS), and economic based methods. While these schedulers maximise individual utilisation they are typically deployed per client or at an organisational level and therefore lack coordination and communication.

The taxonomy presented in this chapter has been used to classify different approaches to meta-schedulers, brokers, superschedulers and economic allocation architectures. Each of the systems surveyed highlights the range of approaches taken. Some concepts and trends can be observed in each of these platforms. It is clear service based architectures are increasingly used as the platform for meta-schedulers due to their inherent standardisation and scalability. Economic allocation protocols have also been re-motivated by the emergence of utility Cloud providers and the quest for global computing environments. Due to this utility focus, in which consumers pay for the resources used, SLAs are vital for establishing guarantees of particular service levels. Of the systems surveyed very few offer standardised SLA mechanisms and only SORMA includes a monitoring and enforcement architecture. Most systems considered in this survey utilised GSI-based security mechanisms for authentication and authorisation as it is the de facto Grid standard. The concept of a VO is also represented in the newer meta-schedulers (SORMA, GridWay) and can be used to represent dynamic groupings in federated environments.

2.6.1 DRIVE

This remainder of this thesis presents the DRIVE meta-scheduler which facilitates federated resource trading in an open market. DRIVE is built from a novel decentralised co-op architecture utilising resources contributed by participating resource providers. DRIVE is implemented as a collection of WSRF web services and is not centred about a single client or client domain, but rather abstracts over collections of resource providers that are members of a VO. Similar to Grid-

Way, the DRIVE prototype uses VOMS to managed the VO, due to its standardisation, scalability, and federation functionality. Uniquely, DRIVE is designed not to be bound to a particular type of middleware so that it may represent different tasks in different domains, however the prototype implementations are aimed at job based Grids and service requests in a Cloud environment. DRIVE also provides a rich economic model for allocations, including support for secure combinatorial allocations, privacy and verifiability. These protocols can be used to establish trust when components are hosted on potentially untrusted providers. To our knowledge these protocols have not been previously explored in any of the meta-scheduling architecture.

DRIVE reverses the general way in which resource allocation is performed. Traditionally meta-schedulers decide where a task should run given requirements by users. DRIVE allows resources to decide if they can meet the requirements of a task and they themselves compete by providing a valuation for executing the task. This models the real world where users have tasks and a market based mechanism is used to determine who performs the task. Like many of the meta-schedulers surveyed, resource discovery in DRIVE is based on open Grid standards, however DRIVE includes a unique hint based model to complement traditional discovery mechanisms. Like SORMA and CSF, DRIVE creates standardised WS-agreement based SLAs as the result of an economic allocation. A novel two phase progressive protocol is used to create binding agreements between participants whilst mitigating the effects of allocation latency. Security mechanisms are based on GSI, making use of VO credentials embedded in certificates and access control decisions by providers. Secure messaging and fine grained resource access security is also implemented.

Chapter 3

Application Scenarios

In the process of designing DRIVE a number of use cases have been considered that characterise application range in distributed Grid, Cloud, and federated systems. A major goal of the DRIVE architecture is to facilitate use in a wide variety of scenarios ranging from single user, single organisation Grids through to large scale federated collaborations using resources provisioned in a global environment. The scenarios presented in this chapter act to motivate the development of DRIVE and are referred back to throughout this thesis. The following section outlines five use cases: a small scale single institution computation of the Tutte polynomial; CyberShake, a large scale multi-provider geophysics computation; outsourcing science in a federated Grid and Cloud scenario; software as a tradable service; and a potential layered market based utility Computing scenario.

3.1 Tutte Polynomial

The first use case considers the needs of a single scientist (or small group of scientists) operating within their own organisation. For this use case we consider the requirements of an individual scientist computing Tutte polynomials [106]. Tutte polynomials are commonly used in a number of different scientific disciplines and represent a computationally expensive process. For a single researcher it would take weeks to execute each of the algorithms against a test suite of graphs on their personal computer. However, when these experiments are broken into jobs they can be run on a small organisational Grid over a weekend.

In graph theory Tutte polynomials can be used to find the number of spanning trees, forests, or connected spanning subgraphs within a graph. Tutte polynomials can also be used in areas such as micro-biology to classify knots – a knot is a form somewhat like a tangled cord with its ends joined. The most well known application of knots is in the study of DNA as the double helix can be viewed as two tangled strands forming a knot. Computation of Tutte polynomials is intensive and is well suited to running on a Grid. The computation can be trivially parallelised into individual jobs that do not transfer large data sets between one another.

The algorithms presented by Haggard et al. [106] provide a way to compute the Tutte polynomial of large graphs. In order to develop and evaluate the proposed algorithms a variety of graph sizes and complexities were considered. The Grid used to test the three proposed algorithms contained approximately 150 desktop computers running Sun Grid Engine [45]. Each experiment tested a single algorithm and was made up of a few hundred jobs (207, 144 and 428 respectively), with each job performing analysis on 100 different sized graphs. Each graph took between a few seconds and a few hours to compute due to the wide range of size and complexity. The total runtime of the three experiments on the Grid was less than 60 hours, with less than 1 MB of data moved between nodes for each job and a resulting data set of approximately 50 MB for all three experiments.

In this use case the requirements for security, trust and privacy are minimal due to the fact the jobs are running within the user's organisation. There are no hard deadlines or QoS constraints and the jobs have a relatively short running time with no dependencies between them. For this scenario a simple (efficient) allocation protocol such as a first price auction protocol can be used as there are no requirements for advanced reservation, deadline considerations, QoS constraints, trust or allocation privacy.

3.2 CyberShake

The second use case considered is the Southern California Earthquake Center (SCEC), a large scale VO encompassing 50 earth science research institutions across the world. SCEC has a production Grid infrastructure used to share participating institution resources amongst collaborating parties. This infrastructure provides the ability for researchers to utilise a large pool of resources, however for many projects the SCEC capabilities are not sufficient. One example is the CyberShake project [107]. CyberShake involves calculation of Probabilistic Seismic Hazard Analysis (PSHA) maps which, amongst other things, are used when designing buildings. Recent advancements in geophysics models allow these PSHA maps to be calculated based on earthquake wave propagation simulations, however these calculations require a large amount of computation and storage. In order to facilitate such large scale calculations large portions of the computation must be outsourced to high end facilities such as TeraGrid.

In CyberShake, PSHA calculations are conducted for each site of interest in stages, with some stages containing thousands of individual jobs. To produce a hazard map of a region at least 1,000 sites need to be calculated. The first stage of Strain Green Tensor (SGT) generation runs over a period of days using 144 or 288 processors and producing up to 10 TB of data. The second stage runs several thousand (on average 5000) individual rupture forecasts each consuming a single processor for between a few minutes and 3 days. The third stage calculates the spectral acceleration and involves the same number of jobs each running for a matter of minutes. Finally the spectral acceleration values are combined and hazard maps are constructed.

The experiments presented in [107] are based on calculation of PSHA maps for two sites of interest using resources provisioned from SCEC and TeraGrid. The resulting experiments involve over 250,000 individual jobs with a total running time equivalent to 1.8 CPU years. The reservation of nodes at two major computing centers for 9.5 days and 7 days respectively is sufficient to perform the calculations.

Scenarios like CyberShake show common infrastructural requirements for large scale science applications. Providing QoS guarantees and enforcing deadlines is more important than in small scale internal experiments due to the need for coordinated computation and storage over a large pool of resources. In order to support scenarios like CyberShake DRIVE must be able to provision resources from dynamic VOs spanning multiple organisations and multiple providers (SCEC and TeraGrid). As there is no expectation of inter organisational trust a secure allocation protocol is required to act in place of a trusted third party to establish trustworthy resource allocation. In this case a verifiable, privacy preserving auction protocol could be used. Additionally advanced reservations and contracts are vital in assuring deadlines are met and levels of QoS can be specified and provided.

3.3 Outsourcing Science

Computational requirements of scientific applications often exceed the combined resource capacity available to users. Researchers now have access to a range of computational resources such as internal organisational resources (Clusters, Grids, supercomputers) and large scale Grid providers such as OSG, TeraGrid and EGEE. These resources are typically shared amongst large user communities and although users may be able to reserve resources there are few QoS guarantees and applications may not be able obtain resources when required. However with the availability of commercial Cloud providers researchers can potentially augment available capacity by purchasing additional compute time. Several studies have considered the possibilities of running scientific applications on commercial providers [108, 109] and augmenting available resources with commercial resources [110, 111]. However dynamic approaches to scheduling usage over all available resources have not yet been explored.

The montage application[1] is a large scale scientific application designed to compute mosaics of images of the sky. Researchers choose particular regions of the sky to create mosaics, depending of the size of the mosaic and the number of input images this task can be incredibly data intensive while also requiring large scale computation. Most often the montage application is organised as a multi-stage workflow. The process of creating mosaics involves re-projecting input images to the coordinate space of the required output, background rectifying the resulting images, and then combining the intermediary images to create the final output mosaic.

The results presented in [111] outline the cost of running both sporadic and large scale montage applications on commercial Cloud providers. The cost using current Cloud technology to create a mosaic of the whole sky was found to be $34,145 assuming all data is already stored in

[1]`http://www.montage.ipac.caltech.edu/`

the Cloud. This cost is a substantial investment that could be used to purchase and maintain significant dedicated resources. It was also found that if a mosaic was likely to be requested within a year it is economically sensible to store the generated mosaic rather than recompute it. Similar results were found by Kondo et al. [109] who concluded current Cloud costs would need to decrease by an order of magnitude to compete with desktop Grids and Volunteer Computing systems.

This type of outsourcing scenario is likely to be explored further in the near future as users realise the potential of augmenting available resources to achieve particular levels of service or to meet deadlines. Transparency in such a scenario is difficult as the allocation fabric requires mechanisms to translate task requirements and representations to the different formats required by the underlying providers. Scheduling algorithms that consider deadline and QoS constraints are commonly represented in computational economies, as such they provide a good basis for federated allocation. To realise such scenarios DRIVE must be able to provision resources from dynamic VOs spanning multiple provider types over disjoint and potentially competitive organisations. Trustworthy protocols are required to establish trusted allocations while keeping commercially sensitive information private. Generic execution mechanisms and data transfer protocols would also need to be developed to facilitate arbitrary resource provider implementations.

3.4 Software as a Service

Software as a Service (Saas) is a scalable application deployment model in which applications are provided to consumers on demand through a service oriented interface. The SaaS model forms the top layer in Cloud computing models [50] operating above infrastructure and platform services. There are many reasons why users may wish to expose an application as a service, for example sharing data or tools, multi-user efficiency, flexibility, extensibility and scalability.

Service oriented approaches have been proposed as a means of advancing science in distributed environments allowing experimentation to be modularised into small tasks and shared with collaborators, hosted on distributed resources, and orchestrated into workflows that model the experimentation process. Service-Oriented Science (SOS) [112] refers to this application of service oriented architectures for enabling scientific experiments, specifically the enablement of scientific research using distributed networks of interoperating services. The use of services allows researchers to share both results and the mechanisms used to obtain them, this facilitates verification and extension by collaborators.

An example of an application that benefits from a SOS approach is the transposon workflow presented in [113]. This workflow is based on a multi-step comparative analysis performed between a genome inserted with mutant libraries (transposons) and the original genome. Sequencing the sites of random transposon insertions is used to assess gene function. In this workflow sequences are iteratively compared against known databases in order to compare sample sequences against known sequences. This process continues until all the sequences are matched or there are no databases remaining. By making components of the various tasks and then exposing them as services (rather than script based applications) the individual services can be hosted anywhere (including scalable infrastructures such as Grids or Clouds), can be replaced or extended, and the workflow can be dynamically modified through a GUI based engine.

If the applications created by scientists are perceived to have value to the wider scientific community providers may want to sell access to consumers. In this thesis we term this approach Software as a Tradable Service (SaaTS). For example if other biologists may benefit from using the transposon workflow (exposed as a service), the research institution may wish to charge consumers to offset operational costs and recoup some of the investment made researching and constructing the service. It is possible providers may also be motivated by profit. By offering the service in a DRIVE market consumers can dynamically provision the service for a price based on current supply and demand. The provider institution could also deploy the service on a scalable distributed infrastructure to account for increased usage.

The difficulty with SaaTS is that many (legacy) applications are not web enabled and converting applications to services is a difficult task due to the range of technologies, tools, and platforms involved. Service wrapping toolkits can be used to reduce the burden on developers therefore simplifying the creation of service based architectures. However developers must still implement economic functionality in the service to participate in a market. To simplify this task these same toolkits could potentially be extended to create economically enabled services which automate the development of tradable services. The gRAVI toolkit presented in Chapter 7 is the first example of a service wrapping toolkit that has been designed to create economically enabled services.

3.5 Utility Computing Wholesale Markets

The final use case considers a potential utility computing scenario. In a utility computing model computation and storage is viewed as a commodity in which users pay for the resources they use much the same as we currently pay for electricity. Although a global utility computing infrastructure remains unrealised, Cloud providers can be viewed as disconnected utility providers. One might imagine that a global computing utility could be implemented using a tiered market based approach similar to the tiered electricity market used in New Zealand [114]. In this model four classes of entities would be involved in the sale of global computing resources: generators, wholesalers, retailers, and consumers. At the highest level is the source of the computation, individual resource providers (*generators*) such as Grid or Cloud providers host computational resources. Generators sell computation to a select group of *wholesalers* in large quantities (in many situations generators may also be wholesalers). Wholesalers then divide their available allocation and on sell it in smaller blocks to *retailers* (or distributors). Retailers in turn directly sell small units of computation to *consumers* (individuals or organisations). Like the power Grid, consumers could also "feedback" computation into the Grid following a micro producer model. This is similar to volunteer computing and could potentially be realised using volunteer middleware deployed to consumer resources.

In a global utility model a separate market may exist at every tier: a generator computation market, a wholesale computation market, a retail computation market and an optional organisational market. The requirements at each of these market tiers are different and can be separately met by different scheduling and trust arrangements:

- Generator to Wholesale and Wholesale to retail: the requirements of the generator and wholesale markets are similar due to the allocation size and commercial implications. These markets represent a very competitive environment in which large quantities of resources are allocated for large sums of money. Therefore, ensuring the correct allocations are made without any commercially sensitive information being released is imperative. Due to the large quantities of resources being reserved and exposure to risk, it is essential to use a secure allocation process. The cost of which may be more than a simple allocation mechanism but in a large scale case this represents a small fraction of the total resources provisioned. For example a sealed bid second price auction protocol (or tender) can be used to dynamically establish the market price for a set of resources without revealing any entities true valuation. The allocation process should therefore be secure and verifiable whilst also keeping commercially sensitive information private.

- Retail to Consumer: the requirements at the retail to consumer tier are more relaxed than the wholesale market as allocations are of a fine to medium grained nature with a large number of consumers. The requirements are subtly different depending on the consumer; in the case of an organisation, individual arrangements may be made between retailer and organisation for example discounts may be applied to advertised rates. While the economic protocol used may be based on an efficient linear model privacy must still be maintained. In the case of individual end users pricing information is commonly advertised. Resources may therefore be allocated using more efficient mechanisms than those used at the wholesale level, for example a provider could use a simple posted price model as price is no longer commercially sensitive. There is little requirement for privacy or verifiability as all users in the system are aware of the posted price.

- Organisation to user: the requirements at the organisation to user level are very fine grained with little to no risk. The organisation level tier would be commonly seen within an organisation but is not an essential part of the overall utility computing model. The main concern of the allocation mechanism within the organisation is efficient allocation with minimal overhead. As the allocation takes place within a trusted domain a simple protocol could be used with no regard for trust or privacy. Specialised sub schedulers such as Falkon [31] may also be used to allocate resources within the organisation.

3.6 Summary

The five use cases presented in this chapter demonstrate a range of requirements placed on the meta-scheduling architecture. The allocation infrastructure required to meet the needs of these scenarios must scale to support dynamic multi-organisation collaborations whilst also providing efficiency in small scale institutional Grid scenarios. In order to do this, mechanisms are required to provide trust, security, and privacy whilst also offering the ability to reserve resources and enforce contracts between participants. The range of use cases shows that no one mechanism will satisfy every scenario, in order to provide the required flexibility DRIVE must have a modular architecture that is capable of supporting different allocation protocols. Generally when provisioning large quantities of resources the overhead associated with secure allocation protocols is tolerable considering the benefits gained. However for smaller allocations this overhead could be a significant percentage of the total job and therefore a more efficient, less secure protocol is more suitable. In general, DRIVE is not designed for very fine grained allocations. In order to create dynamic service trading environments, tools such as gRAVI are required to automate the design, implementation, and deployment of economically enabled services.

Chapter 4

DRIVE Architecture

DRIVE is based on a decentralised distributed architecture composed of a group of market and participant services. Figure 4.1 presents an overview of the major components within DRIVE. The DRIVE marketplace which facilitates open trading is at the core of the architecture. The market is made up of several independent services and mechanisms which collectively perform standard meta-scheduling tasks such as discovery, allocation, and management. Service providers are represented by DRIVE Agents when trading resources or services in the DRIVE market. Consumers are represented by a client-side broker designed to abstract economic negotiation.

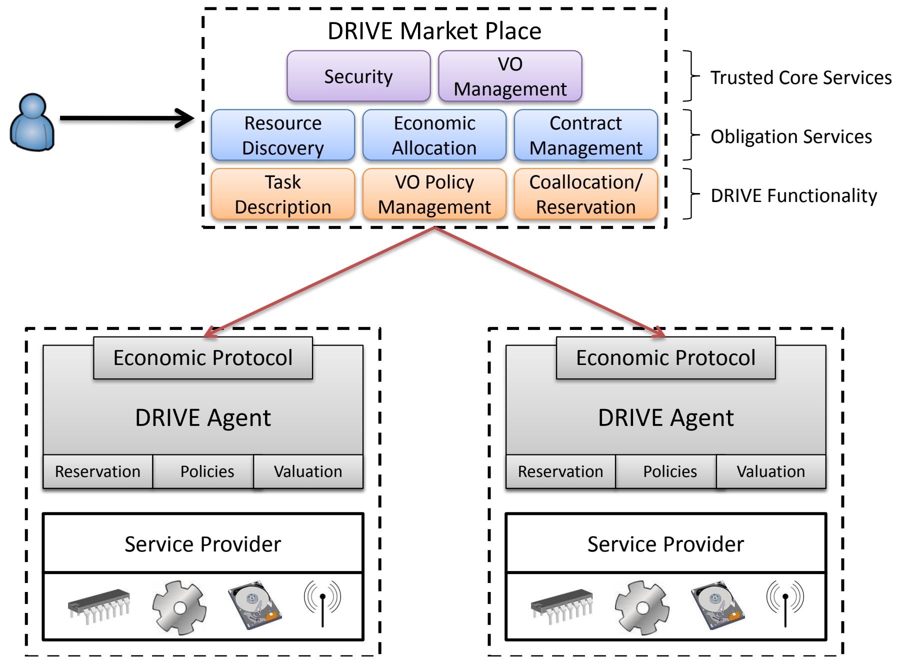


Figure 4.1: Overview of the high level DRIVE Architecture.

DRIVE is built upon a scalable distributed service-based architecture in which services can potentially be hosted on participating providers. Due to this lack of infrastructural trust, DRIVE services are defined according to a trust model composed of three categories: Participation, Obligation and Trusted Core. *Participation* services are required by service providers in order to interact with DRIVE components, for example DRIVE Agents are used to represent service providers. *Obligation* services are contributed by providers and collectively provide the core functionality of the meta-scheduler including (economic) resource allocation, contract negotiation, advertising, and management. *Trusted core* services are services which must be always available or which cannot be hosted by participating resource providers, for example security, VO management, and authoritative discovery services. The particular trust model used is implementation and deployment specific.

The DRIVE marketplace provides various functionality such as the ability to map task descriptions, implement fine grained and VO policies, co-allocate tasks, and reserve resources using a flexible reservation window. DRIVE Agents use policies to determine negotiation strategies and valuation functions to price resources. Individual providers may choose to use a generic DRIVE Reservation Service in order to facilitate domain-independent advanced reservation. The DRIVE market is designed to be open, therefore negotiations can take place using any pre-defined protocol. As the result of a negotiation a SLA based contract is created to represent the obligations of participants.

The lifecycle of interaction with DRIVE is shown in Figure 4.2. Users submit tasks for allocation using a specified economic protocol. An initial agreement is created as the result of a negotiation between consumer and providers (phase 1), this agreement is then *hardened* into a binding contract (phase 2) before the resources are allocated to the consumer. This two phase contract model is used to mitigate the effects of allocation latency and therefore encourage participation in the market.

While the DRIVE architecture is not bound to a particular hosting environment, the design utilises several concepts provided by stateful Grid and Web services. The reference Web service implementation used is Globus Toolkit 4 (GT4) [115] compliant WSRF [22] Web services. The design also attempts to leverage existing services, mechanisms, and standards where possible so as to maximise interoperability and encourage standardisation.

4.1 Design Goals

DRIVE has been designed with several clear goals in mind which influence a number of the design decisions presented in this chapter. The core design goals and their implications on the architecture are:

1. **Provider and task independent:** DRIVE is designed to be independent from a single provider or class of providers such that different types of provider can participate in a federated market. For example, DRIVE can be used to submit jobs in a Grid environment and VMs (or

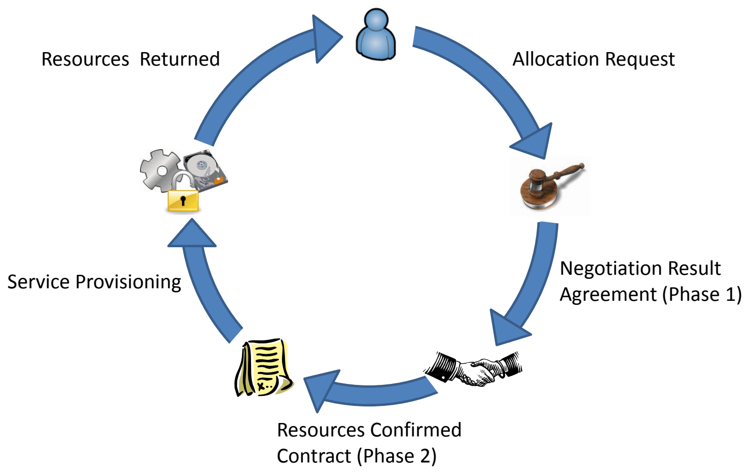


Figure 4.2: DRIVE Allocation Lifecycle.

service requests) in a Cloud environment. The important aspects of the architecture that provide this flexibility are the task representation, contract representation, and task execution mechanisms. The task definition is not bound to a specific domain, therefore users can describe a task in any domain specific representation. The allocation procedure is independent of this description assuming providers and users understand the task description and are able to compute valuations. The resulting contract is formed using the task description as the term language. Task execution is outside the scope of DRIVE, however due to the flexibility in design any suitable execution mechanism could be used, for example Grid schedulers, Web/Grid services, or VMs.

2. **Support efficient flexible allocation mechanisms:**

   (a) **Efficient allocation mechanisms:** The core responsibility of a meta-scheduler is providing efficient allocation. DRIVE focuses on efficient allocation using computational (or service) economies for two reasons: first, economic allocation principles are necessitated as commercial providers may be included in the VO. Secondly, economies have been shown to provide effective decentralised resource allocation in distributed systems [24].

   (b) **Protocol independent:** DRIVE is designed to be deployed in a wide range of scenarios (such as those presented in Chapter 3) which have vastly different allocation requirements. For instance, within an organisation there is a level of internal trust, therefore users can achieve maximum allocation performance by using an efficient protocol which does not utilise expensive trust or privacy preserving mechanisms. In a global federated environment no such trust exists. Users therefore require a secure protocol to ensure resource allocation is carried out fairly and financial details are kept private. DRIVE provides protocol independence through a plug-in protocol architecture exposed by all market services.

3. **Secure:**

   (a) **General VO security:** Standardised security mechanisms are required to authenticate and authorise consumers and providers for accountability. DRIVE is intended to interact with both commercial and non-commercial providers. In a commercial environment security implications are magnified due to the motivation for malicious or subversive behaviour.

   (b) **Fine grained security:** Due to the distributed nature of DRIVE participants, protocols and services must be secured such that only permitted actions can be performed and communications are kept private. This is achieved through fine grained security policies and secure communication channels.

   (c) **Secure privacy preserving allocation:** The design of DRIVE has considered the requirements of various secure, privacy preserving protocols. These protocols allow trustworthy resource allocation in commercial environments by providing guarantees over allocation results and financial privacy in untrusted environments. Many of the protocols considered have strict threshold requirements which necessitate the development of generic distributed Allocation Components.

4. **Scalable:**

   (a) **Scalable architecture:** DRIVE has been designed to represent a range of scenarios from small scale single organisation distributed systems through to global federated environments. In order to support large federated environments, DRIVE must scale in terms of the number of consumers and providers represented, concurrent allocations supported, and contracts created. In order to do this DRIVE itself is designed as a decentralised distributed architecture where no single service is required to coordinate access.

   (b) **Autonomous and self managing:** To reduce the management requirements DRIVE (and market participants) must be autonomous and self managing. This is accomplished through the definition and use of a strong policy framework throughout the architecture.

   (c) **Minimal infrastructure :** One of the major motivating factors for implementing secure protocols is the goal of reducing infrastructural requirements for hosting DRIVE. Current meta-scheduling architectures require dedicated resources on which to host the meta-scheduler. Scaling these architectures to large scale federated environments or global Grids would require large scale investment in dedicated resources, the problem with this approach is determining who should provide such resources. Due to the complexity (and distribution) of secure economic protocols a dedicated meta-scheduler will not scale without large scale investment in dedicated resources. DRIVE overcomes these limitations by using a novel co-op architecture in which meta-scheduler services are hosted on participating providers.

5. **Provide a strong contract management framework:** With the development of utility computing models there is a requirement to define, monitor and enforce agreed upon levels of service. Consumers will not be prepared to pay if negotiated service levels are not delivered. DRIVE creates standardised binding contracts as a result of allocation such that consumers and providers clearly define their respective requirements and obligations. Contract monitoring and enforcement is outside the scope of this thesis, however both have been considered to future-proof the architecture.

6. **Open, interoperable, and standardised:** Where possible DRIVE has been designed to leverage existing standardised services, mechanisms and protocols for aspects such as contract creation, resource discovery, and service design. This is intended to maximise the interoperability of the DRIVE architecture and therefore create an open market infrastructure.

4.2 High Level Architecture

Figure 4.3 shows a typical DRIVE deployment, highlighting the co-op service-based infrastructure used. In this figure four service providers (SPs) are existing members of the VO, each exposes a set of services that facilitate resource discovery, allocation, contract negotiation, and task execution within the VO. SP3 is shown offering only participation services, SP1 and SP4 offer common obligation services, SP2 reflects a typical provider hosting a combination of participation and obligation services. The registration and discovery services in the trusted core are shown in the centre of the diagram to indicate they are hosted in a trusted environment. Each service provider in a DRIVE VO may represent an independent distributed environment (Grid, Cloud, Cluster). In this diagram SP 5 is attempting to join the VO by registering itself with the VO registration service, and its participation and obligation services in a VO discovery service. The other providers show a simple negotiation. DRIVE Agents acting on behalf of SP2 and SP3 are negotiating for the rights to host a task submitted by a user through a DRIVE broker, negotiation takes place using a contributed Allocation Manager hosted on SP1.

The remainder of this chapter presents the DRIVE architecture including descriptions of the components and protocols used. Specifically the chapter focuses on the following areas:

- **Resource Allocation (Section 4.3):** DRIVE is designed to create a market in which a range of economic protocols can be used concurrently to trade resources. Auction protocols in particular are well suited to distributed resource allocation as they can dynamically and efficiently establish market prices. However, these protocols are susceptible to cheating and

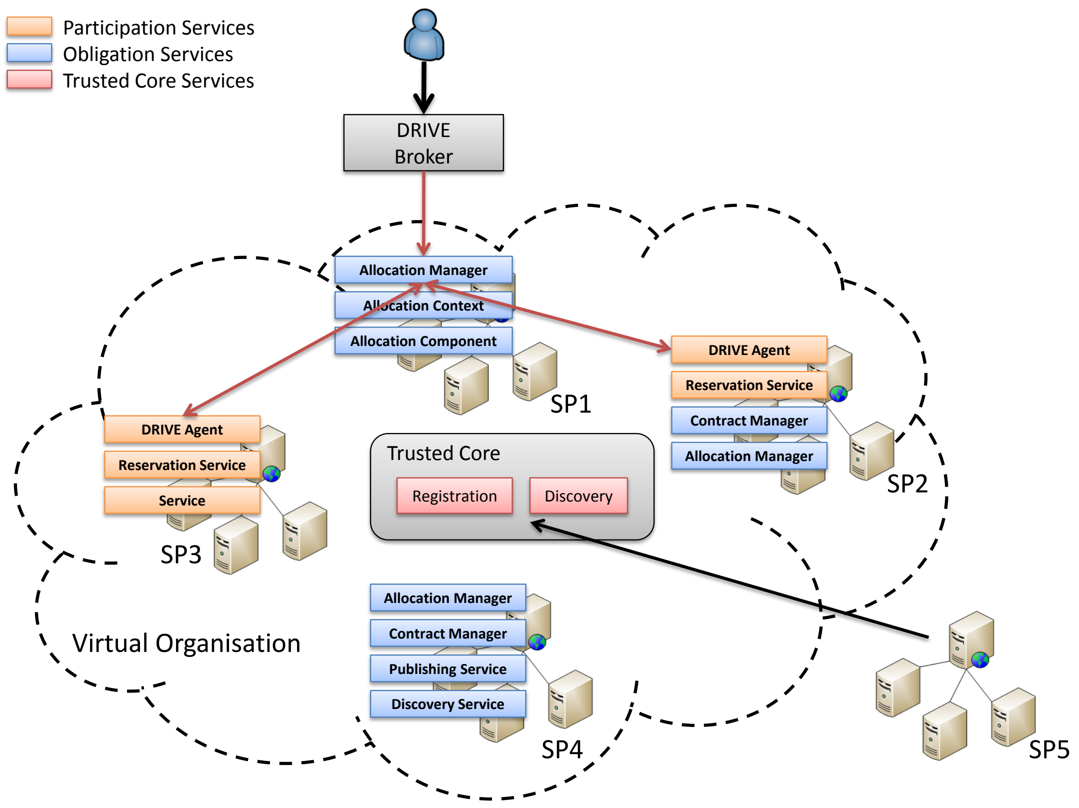


Figure 4.3: DRIVE Service based topology. Several Service Providers (SP) are members of a DRIVE VO. A simple negotiation is taking place between a user and two DRIVE Agents (SP2 and SP3) using an Allocation Manager hosted on SP1. SP5 is attempting to join the VO by registering with the registration and discovery components of DRIVE.

release of commercially sensitive information in untrusted distributed environments. Secure auction protocols can be used to overcome some of these limitations through encryption and threshold trust. In DRIVE an Allocation Manager is responsible for managing the allocation process, storing state and determining matches between consumer and provider for a given task following an arbitrary economic protocol. In this process a number of Allocation Components may be instantiated and used to meet the requirements of a given distributed economic protocol (as is commonly seen in secure auction protocols).

- **Service Provider Interaction (Section 4.4):** DRIVE Agents represent service providers in the DRIVE market, abstracting the complexities of valuation and economic negotiation. Administrators are able to define high level policies regarding negotiation participation and resource valuation. A proprietary DRIVE Reservation Service is used to support provider-independent advanced reservation. Use of the Reservation Service is optional, however it provides standardised support for advanced reservations and can be used to implement scheduling algorithms to optimise allocation.

- **Contract Management (Section 4.5):** Like real-world economies, DRIVE uses contracts to represent negotiated terms between consumers and providers. A contract defines a SLA specifying the agreed upon levels of service for a particular provision. As parties in DRIVE are assumed to be self interested, there may be motivation to violate contracts. For this reason DRIVE relies on incentives to encourage honouring contractual obligations. The contract creation process in DRIVE is abstracted through a Contract Manager which is responsible for creating and distributing agreements to providers and consumers. If any providers reject a possible agreement the Contract Manager (depending on the protocol) is able to transparently create agreements with substitute providers to meet the requirements of the task.

- **VO Management (Section 4.6):** DRIVE relies on a VO model to represent the highly dynamic nature of distributed Grid and Cloud environments. To support this model VO management systems are used to register and authenticate users, and define fine grained and VO authorisation policies.

- **Registration and Discovery (Section 4.7):** DRIVE supports two levels of provider registration: Firstly, providers periodically register metadata describing their capabilities which is used by Allocation Managers to target advertisements. Secondly, providers also register the address of any obligation and participation services so that they can be used by other DRIVE services. Unlike traditional Grid systems, provider discovery is not the basis for matchmaking in the DRIVE architecture, rather resource requirements are evaluated by providers during valuation of a resource request. Essentially registered metadata acts as a "hint" providing a partial picture of the system, reverse discovery is used to complete the discovery process allowing providers to discover potential allocations.

4.3 Resource Allocation

In traditional computing models, resource management is undertaken by a single entity (the operating system) which provides absolute control of resource allocation. The operating system is responsible for providing optimal resource allocation both for the system as a whole (globally) and for individual applications (locally) [28]. In many situations however, these goals conflict, for this reason operating systems separate global and local optimality, instead relying on resource negotiation between applications and the operating system to determine allocation. In distributed environments, it is not feasible (or appropriate) for a single entity to manage resources across the entire virtualised system. Like individual computers, goals and strategies may differ between consumers (applications), VOs, and resource providers. A more thorough negotiation mechanism is required such that each party can adequately express their respective goals and reach a mutually agreeable outcome.

Negotiation mechanisms are composed of three components: a protocol that facilitates the process, a communication language to express negotiation parameters, and policies describing relationships between parties (what is negotiated and why). Negotiation is generally categorised according to the number of parties involved: *bipartite* negotiation involves two parties whereas *multipartite* negotiation involves more than two (this may be either multiple buyers or multiple sellers). The difference is important as it defines where competition exists, bipartite negotiation involves competition between the buyer and seller, whereas multipartite negotiation involves competition between buyers (or sellers).

Distributed resource allocation is tasked with allocating a finite pool of resources across a population of potential consumers. This task is inherent in human negotiation and forms the basis of modern economics. As such the use of economies in computational resource management has long been touted as a solution for efficient resource allocation. In fact computational economies were first used in 1968 to allocate compute time on a departmental PDP-1 [23]. In Grid and Cloud systems providers have differing policies, cost models, and resource availability, consumers on the other hand have different resource requirements and budgets. In addition, both consumers and providers have different goals, strategies, and valuation functions. Economies provides a way to represent these diverse perspectives by defining a range of protocols that support both bipartite and multipartite negotiation between participants [25].

Economies have a number of advantages that make them well suited to distributed resource allocation. They are typically scalable and adaptable to rapidly changing market conditions. In general they provide effective decentralised decision making with minimal communication overhead [27]. They provide a well understood class of mechanisms with an extensive body of research outlining aspects such as dominant strategies, revenue management, and entity behaviour. Economic models allow providers to specify what is shared, with whom, while also providing incentives for providers and consumers to participate in a market. In addition to these advantages, federated environments containing commercial Cloud providers necessitate the use of economically aware allocation mechanisms driven by the underlying allocation principles used by commercial providers. In DRIVE there may be multiple self interested commercial and non-commercial entities acting as competitive or cooperative consumers and/or providers, each serving different clients and following independent goals [116].

The remainder of this section provides an overview of economic negotiation protocols applied in distributed domains, in particular it focuses on secure combinatorial auction protocols for which DRIVE is in part designed. The DRIVE allocation architecture is then described based upon the requirements generated by the protocols presented.

4.3.1 Economic Models

Market abstractions can be used to efficiently determine the true price for a good (or service) within an economy, balancing supply and demand to reach equilibrium. When determining appropriate economic models it is important to consider the interactions between entities within the system, entities are generally classified as *cooperative*, *hostile*, or *competitive* (self-interested). Cooperative entities aim to increase global efficiency thus maximising global gain at the expensive of possible local loss. Hostile entities aim to reduce global efficiency in an effort to ensure global loss with no concern for local gain (or loss). Competitive entities are self-interested, that is they are only concerned with local gain and are indifferent in terms of global gain or loss. In a federated environment composed of autonomous commercial and non-commercial providers, it is assumed providers are most often self interested and therefore competitive.

There are a number of competitive economic models that can be applied to distributed resource management including auctions, bargaining, commodity markets, posted price, and tendering [25]:

- **Auctions:** An auction facilitates multipartite negotiation to establish a market price for a good or service. Auctioneers control the auction, setting rules, and following the auction protocol. Auctions are used extensively for online sales of goods through sites such as eBay[1] as they are an efficient way to determine the price for good. Auctions are discussed in detail in Section 4.3.3

- **Bargaining:** Bargaining is a multi step bipartite negotiation process to establish a mutually agreeable price for a good. Consumers bargain with providers for lower prices or increased usage, negotiation finishes when one party is no longer willing to negotiate.

- **Commodity Markets:** Commodity markets facilitate raw resource exchange, providers set prices and charge users for the amount of resource consumed. Pricing is generally based on a flat fee or variable function altered to reach supply and demand equilibrium.

- **Posted Price:** Resource offers are posted in the market by providers, consumers then discover offers and select the offers that best meet their needs. Posted price models are the predominant model used in Cloud computing.

- **Tendering:** A tender aims to find a suitable provider to host a given task. In a tender a task is advertised to a number of providers. Each provider computes a one time offer to provide the requested service. The lowest price is generally selected as the winner of the tender. This is a common model used in distributed environments and is essentially a reverse auction.

There are also a number of co-operative economic models in which consumers and providers work together to obtain an agreement [25]. Models such as credit sharing, bid-based proportional resource sharing, and cooperative bartering/shareholder models are often used to share resources in co-operative environments. DRIVE is focused on competitive federated environments, as such the focus of this survey is competitive economic protocols.

[1] `http://www.ebay.com/`

4.3.2 Economic measures

The goal of any economic negotiation is mutual satisfaction between consumer and provider. As a basis for analysis of economic protocols there are a number of measures of satisfaction developed in Economics, Game theory and Computational economies [28, 117, 118]. The following criteria are often used to analyse economic protocols:

- **Social Welfare:** The social welfare of a system is a global measure of the summation of individual welfare, where individual welfare is the payoff for an individual. When the social welfare of a system is maximised the resource distribution is pareto optimal.
- **Pareto Optimality/Efficiency:** A resource distribution is Pareto optimal if any redistribution that will make an entity better off is impossible without also making one or more entities worse off.
- **Individual Rationality:** Individual rationality describes the situation where entities are better off participating in the negotiation than not participating, that is, the payoff for participating is not less than the payoff for not participating.
- **Stability:** A system is stable if it is in every entities best interests to retain their current strategy.
- **Symmetry:** A system is symmetric if individuals with the same behaviour have the same expected payoff.
- **Computational Efficiency:** The computational efficiency of a system describes the overhead required to reach a solution.
- **Communication Efficiency:** The communication efficiency of a system reflects the overhead of communication between entities required to reach a solution.

Ideally protocols used in competitive distributed resource allocation should be all of the above. Ensuring that the payoffs for the system are maximised (social welfare), allocations are optimal (pareto optimal), entities are encouraged to participate by receiving profit and workload (individual rational), it is best for entities to maintain their strategies (stability), entities with the same strategies expect the same result (symmetric), and the protocol has minimal computation and communication overhead.

4.3.3 Auctions

Auction protocols facilitate multipartite negotiation between consumers and providers. Figure 4.4 depicts two multipartite resource auction scenarios reflecting competition between consumers and competition between providers respectively. Auctions are commonly considered from a consumers (buyers) perspective, in this scenario consumers bid for particular resources from a single provider. The left side of Figure 4.4 shows a standard auction. Two consumers bid 1$ and 2$ for 1 unit of computation auctioned by a single provider, the auctioneer then determines the winner as the highest bid (User 2) and allocates resources. On the right side of Figure 4.4 a reverse auction (or tender) is shown. Reverse auctions allow consideration from the providers (seller) perspective. In this example three providers bid (or tender) 1$, 2$ and 3$ for the requested user task (1 unit of computation), the winner is declared as the lowest offer (provider 1).

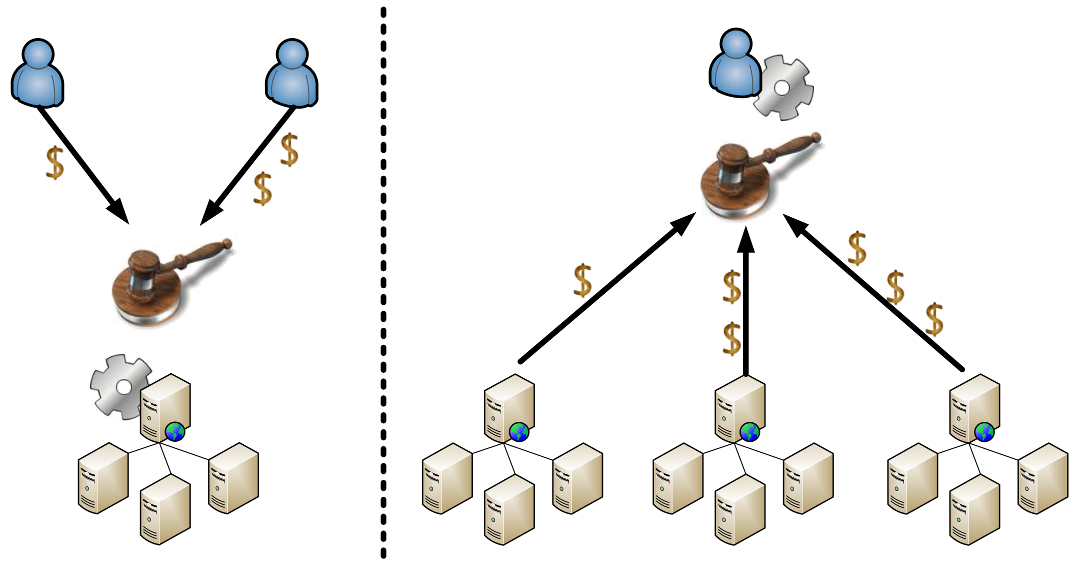


Figure 4.4: Auction for computational resources. A standard auction is shown on the left, here consumers compete for the computation auctioned by the provider. A reverse auction is shown on the right, here providers compete for the task submitted by a consumer.

Auction protocols have a number of desirable properties which make them attractive for use in a distributed environment in particular they are generally an efficient means of establishing a market price in an open market [119] and have been shown to provide Quality of Service (QoS) advantages over other methods [29]. Analysis of auction protocols relies on knowledge of how bidders value resources. There are three categories which describe bidders' values: private value, common value, and correlated value.

- **Private value auctions:** a bidders valuation is determined by the bidders preferences alone. Potential resale value is not considered.

- **Common value auctions:** a bidders valuation is based completely on their perception of other bidders values for the goods.

- **Correlated value auctions:** a combination of private and common value, a bidders valuation depends in part on its own preferences and in part on others' values.

4.3.4 Auctions Protocols

There are a five main types of auction protocols: English, Dutch, Sealed-bid, Vickrey and Double. Analysis of the different protocols often focuses on dominant bidding strategies under particular conditions. A dominant strategy is defined as a strategy that gives the bidder the highest payoff independent of the strategies employed by any other bidder.

- **English Auction (first price, open bid, ascending):** An English auction is the most common auction protocol. Bidders openly increase their bids until only a single bidder remains. The price paid is the highest price bid by the winning bidder. Bidding starts low and increases until demand falls. In private value English auctions the dominant strategy is to increase ones bid by small increments stopping when the private value is reached (incremental bidding).

- **Dutch Auction (first price, open bid, descending):** A Dutch auction is a descending version of the English auction. The auctioneer declares a starting bid and decreases the price until a bidder bids. Here bidding starts high and decreases until demand matches supply. There is no general dominant strategy for Dutch auctions, however counterspeculation and lying may be used to increase payoff at a cost.

- **Sealed bid Auction (first price, sealed bid):** In a Sealed bid auction each bidder submits a single sealed bid without knowledge of any other bids. The highest bidder wins the auction and pays their bid. There is no general dominant strategy for sealed bid auctions, however strategic underbidding and counterspeculation may increase payoff assuming there are many bidders.

- **Vickrey Auction (second price, sealed bid):** In a Vickrey auction each bidder submits a single sealed bid without knowing any other bids. The highest bidder wins the auction and pays the second highest bid. A private value Vickrey auction is strategically equivalent to an English auction, as such the dominant strategy is to bid ones true value. The advantage of the Vickrey protocol over the English protocol is reduced communication as bidders must bid their true valuation in a single bid, rather than iteratively increasing ones bid.

- **Double Auction:** A Double auction differs from the other four protocols in that multiple buyers submit their bids while sellers simultaneously submit offers to an auctioneer. The auctioneer follows a matching process between offers and bids, this process starts with the lowest price offers moving up and matches each with the highest price bids moving down. Double auctions are commonly used for trading in stock markets.

The revenue equivalence theorem states that the four standard auction protocols (English, Dutch, Sealed bid, and Vickrey) yield the same expected return in private value auctions. Assuming valuations are developed independently and bidders are risk-neutral. In Dutch and Sealed bid auctions equilibrium depends on bidders guessing other entity's strategies, whereas in English and Vickrey auctions a bidder's strategy does not depend on predicting other bidders' values which ensures no computation is spent speculating on other bidders' actions. In private value auctions the dominant strategy in both English and Vickrey auctions is truthful bidding, however in non-private value auctions the dominant strategy changes as the open nature of the English auction reveals information about other bidders' valuations and can therefore be used to determine potential resale value.

Truthful bidding has obvious advantages in computational economies as bidders do not consume resources speculating on other bidder's behaviour. Sealed bid auctions have the additional advantage of communication efficiency as bidders bid only once. For these reasons Vickrey auctions are particularly attractive for use in computational economies. In a Vickrey auction allocations are Pareto optimal and therefore provide social welfare. The protocol is stable as it is best to follow the dominant strategy of truthful bidding and symmetrical as bidders with the same behaviour can expect the same result. The lack of counterspeculation seen in other protocols provides computational efficiency. Communication efficiency is maximised as bidders bid their true value immediately in a single (sealed) bid.

Vickrey Protocol Limitations

While the Vickrey protocol has a number of desirable properties, there are also a number of limitations [120], for example lower revenue for non-private value auctions, lying in non-private value auctions, bidder collusion, lying auctioneers, and the release of potentially sensitive information.

- **Lower revenue:** In non-private value auctions with at least three bidders the English auction leads to higher revenue because the observation of other bidders prices raises a bidders valuation. However this raises a new issue, in non-private value auctions the *winners curse* states that the winning bidder has paid too much because a bidders valuation is dependent on (at least in part) other bidders values. Knowledge of the winners curse means bidders should bid less than their true valuations [121].

- **Lying:** Like many other auction protocols, it may be advantageous to lie about ones true value for a good in a non-private value auction as other bidders' values are in part based on bid values.

- **Bidder Collusion:** Bidder collusion effects all auction protocols as bidders can coordinate bids to ensure the winning price is low [122]. However, to collude bidders must coordinate before the auction.

- **Lying auctioneer:** All auction protocols place inherent trust in the auctioneer and are therefore susceptible to a lying auctioneer. In addition to these standard problems the Vickrey auction protocol also relies on the auctioneer to accurately compute the second price. Theoretically the auctioneer could issue false bids to artificially inflate the second bid, essentially using a first price protocol in place of the second price Vickrey protocol [123] as winning bidders are unaware of the second bid. In all auction protocols the auctioneer may place false bids, attempt to omit bids, or reveal potentially sensitive information.

- **Release of information:** While truthful bidding may be beneficial in computational economies it has the disadvantage of bidders revealing their true valuation, which may be sensitive in a commercial environment. Even if bid information can be kept private for the duration of the auction, it may still be valuable as an indication of past behaviour and can be used to estimate willingness to pay [124].

The problems related to lying auctioneers and release of sensitive information have been suggested as the main reasons as to why the Vickrey auction protocol is rarely used in human economies [125]. In computational economies however, these problems can both be mitigated using cryptography and secure auction protocols as discussed in Section 4.3.6.

4.3.5 Combinatorial Auctions

In many domains auctions involve the sale of a variety of distinct goods. Due to the increased utility of dependant items, bidders often have preferences for sets (or *bundles*) of items. A common example used to highlight increased utility is the sale of shoes, one shoe by itself may have limited value to bidders, however if sold in a *bundle* as a pair of shoes, their value is increased. The same applies if items are viewed as substitutes for one another as bidders may have preference for one or the other but not both (for example identical right shoes may be substituted). Allowing bidders to bid on *combinations* of items increases economic efficiency in these scenarios [126].

To meet QoS criteria in Grid and Cloud systems, an auction for a single representative resource is not normally sufficient, for example, CPU time is only one of the many resources required for execution. In addition particular resources may be preferred given deadline constraints while others may be viewed as equivalent (substitutes). These preferences can be accurately represented using combinatorial auctions therefore allowing bidders to bid on collections of goods.

Figure 4.5 shows an example of a reverse combinatorial auction, as would be commonly used in DRIVE. In this example a consumer submits a request (Auction Description) for 1 unit of computation, memory, and storage. Individual providers then deconstruct the request and bid on particular bundles. Provider 1 is shown offering computation and memory for 2\$, and computation alone for 1\$. Provider 2 is offering memory for 2\$ and computation for 1\$. Provider 3 is offering computation, memory, and storage for 4\$, and storage alone for 1\$. When determining the winner(s) the auctioneer must process each combination of bids, in this case the best way to satisfy the requirements would be to use computation and memory from provider 1 and storage from provider 3, therefore costing 3\$. This example shows a simplistic request, a more realistic combinatorial request would involve complex nesting of "and" and "or" semantics to express preferences and substitutes. For example a consumer may be willing to give up storage for twice the memory offered.

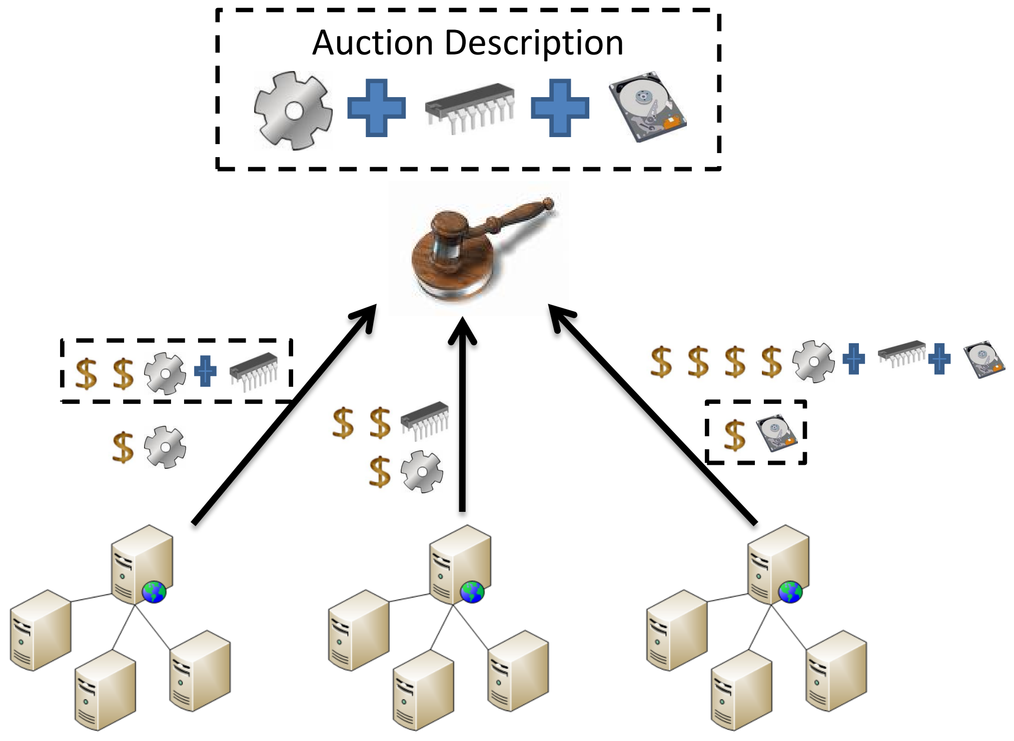


Figure 4.5: Combinatorial reverse auction for system resources. Providers bid on combinations of resources expressed in the auction description, the winning provider(s) are found by combining bundles to discover the best price.

In combinatorial auctions bidders submit a bid for every subset of objects they are interested in. This results in increased complexity both in terms of representing the bids and determining the winning set of bids. This problem is termed the Combinatorial Auction Problem (CAP). In combinatorial auctions winner determination is NP-Complete and can be formulated as an instance of the Set Packing Problem, a well studied integer program. There are a number of methods used to minimise the CAP, including approximation methods and decentralised approaches, however implementation is complicated and may not be guaranteed to produce optimal allocation. Minimisation of the CAP is outside the scope of DRIVE and is a task left to combinatorial protocol developers.

Combinatorial Representation

Representation of the different combinations, preferences and substitutes is difficult due to the complexity of combinatorial auctions. Various techniques have been explored to represent combinations such as graphs, matrices, and circuits. Graphs are the most common approach where nodes represent unallocated goods, edges represent an allocated subset of goods to bidders, and each path through the graph represents an allocation of goods to bidders.

Figure 4.6 shows a combinatorial auction graph for a reverse (descending) auction, in this example 3 goods {G1, G2, G3} are offered and 2 bidders {B1, B2} have submitted bids on various combinations of these goods. The winning path is denoted with a solid red line which represents the lowest value path through the graph. The winning path allocates {G2} to B1 for $2 and {G1,G3} to B2 for $5, resulting in a total value of $7. Clearly this graph can be used to represent first price auction protocols as the price paid is simply the sum of the paths. However, computing the second price in a Vickrey auction is difficult due to the complexity establishing a bidders impact on the auction. One example of a combinatorial Vickrey auction protocol is the Generalised Vickrey Auction.

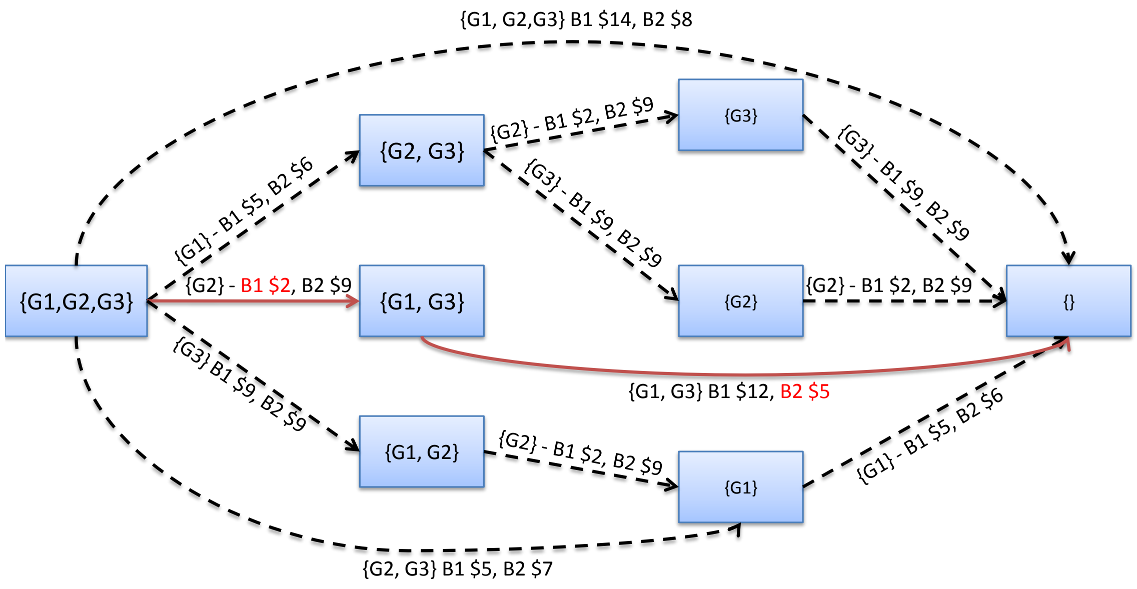


Figure 4.6: Combinatorial graph representation. This graphs shows bids from 2 bidders {B1,B2} on combinations of 3 goods {G1,G2,G3}.

Generalised Vickrey Auction

The Generalised Vickrey Auction (GVA) [127] protocol is an extension of the Vickrey auction protocol for combinatorial auctions. In a GVA bidders submit sealed bids for each combination of goods in the auction, the winning bidder then pays the second highest (or lowest) bid. The second price is established by applying a discount based on the winners contribution to the allocation [28]. In the GVA protocol the CAP is first solved by finding S^* (the maximising allocation of S) and V^* (the maximising valuation) calculated based on individual bidders valuations v. In the set S^* there are W winners. The GVA protocol then solves the CAP W times, once for each winner $i \in W$ resulting in maximising allocations S_i^* and valuations V_i^* without winner i. The difference between valuations $V^* - V_i^*$ represents an individual winners contribution to the allocation and is termed the Vickrey discount (Δ_i). The winning bidder pays their bid minus the calculated discount. In a reverse auction the discount is negative, resulting in an increased winning price.

$$P(i) = v_i(S_i^*) - \Delta_i$$

To demonstrate the GVA discount calculation the combinatorial auction represented in Figure 4.6 can again be used. Recall the total value of this auction was \$7, B1 won {G2} for \$2 and B2 won {G1,G3} for \$5. The discount is calculated by determining the effect of each bidder on the auction:

- If B1 were removed from the auction the lowest bid would now be \$8 (B2 {G1,G2,G3}), therefore the discount is -\$1 (\$7-\$8) and B1 would receive \$3 (\$2+\$1) for {G2}.
- If B2 were removed from the auction the lowest price would now be \$10 (B1 {G2,G3} \$5 + B1 {G1} \$5). The discount is therefore -\$5 (\$5-\$10) and B2 would receive \$10 (\$5+\$5) for {G1,G3}.

The total value paid to bidders in the auction has increased to \$13 as this is essentially the "second price" if both bidders were removed. Obviously this is a trivial example with only two bidders and three goods, however these calculations highlight the complexity involved in combinatorial auctions both when representing bids and determining winners (and the second price). In the GVA protocol a great deal of trust must also be placed in the auctioneer to calculate the prices correctly. These issues motivate the need for distributed secure auction protocols.

4.3.6 Secure Auction Protocols

Trust is often implicitly assumed in computing environments due to trusted infrastructure or external policies and procedures regarding administration of users. However in a large scale distributed environment this assumption is no longer valid as trust between entities is difficult to establish. Traditional auction protocols require a level of trust in the auctioneer to prevent a malicious auctioneer revealing bid information or manipulating the outcome of an auction. Developing trusted auctioneers leads to centralised system design and therefore reduced protocol scalability and performance. A centralised trusted auctioneer is infeasible in a large scale federated model due to a lack of scalability, single point of failure, and a requirement for dedicated infrastructure. To overcome these limitations trust can be established through protocols such that any entity (within reason) can be trusted to perform a well defined action (such as resource allocation).

There are three key requirements for holding trustworthy resource auctions in a federated environment [128]:

1. Avoid trusting a single auctioneer;
2. Provide strong bid privacy; and,
3. Provide verification that the auction was conducted correctly.

The first two requirements can be established through secure distributed auction protocols. Verification may occur at the conclusion of an auction, the goal being a level of assurance that the protocol was followed correctly while still maintaining privacy. In general verification techniques can detect cheating agents, ensure all bids have been considered, and validate the outcome of the auction. Verification techniques are not a focus of this thesis. Various techniques have been explored in the literature to facilitate public verification without revealing additional information [128, 129, 130, 131]. The remainder of this thesis focuses on secure protocols used to provide bid privacy and distribute trust over a group of auctioneers.

Established Trust

There are a number of ways to establish trust in an auction [132]:

- System components (Pre-existing trust)
- Reputation services (earned trust)
- Bid-encryption schemes (procedural trust)
- Threshold schemes (distributed trust)
- Threshold/bid-encryption schemes (enforced trust)

The simplest approach is to leverage *pre-existing trust* between the auctioneer and the auction participants, this may be established through dedicated infrastructure or external relationships between entities. For example an auctioneer running within an institution has a high level of pre-established trust amongst members of the same institution. Reputation services provide a way to *earn trust* through actions [91], these actions may take place within the auction environment or possibly from actions in other systems or even domains. Reputation can be aggregated from any number of sources and may facilitate reputation delegation. For example if an auctioneer has previously conducted auctions for a particular user, the user is more likely to trust the same auctioneer in future auctions. While pre-existing and earned trust can be used to gain a level of trust in the auction entities, they do not ensure the auction is conducted correctly or provide any guarantees that information is not released.

Bid-encryption can be used to restrict the release of sensitive information providing a level of *procedural trust*. Cryptographic techniques make it difficult for auction entities to discover bid information or subvert the outcome of an auction by altering bid data. Threshold schemes *distribute trust* over a group of auctioneers, thus removing the need for trust in a single auctioneer. Threshold schemes require a number of distributed non-malicious auctioneers to execute the protocol correctly, ensuring any malicious auctioneers cannot subvert the auction. Public key cryptography is often used for coordinated multi-party decryption of bids. Threshold bid-encryption schemes combine the privacy preservation properties of bid-encryption with distributed threshold mechanisms thus *enforcing trust* which ensures the auction protocol against a malicious auctioneer.

Secure Auction Protocol Examples

One example of a secure auction protocol is the Secure Generalised Vickrey Auction (SGVA) [92, 93]. The SGVA protocol preserves bid privacy through a threshold bid-encryption technique. Valuations are hidden in the bid representation while still allowing winners and the associated prices to be found without revealing bid information. Various techniques can be used to encrypt the bid including homomorphic [93, 132] or polynomial encryption [92, 133]. In the homomorphic scheme bid values are represented as bit vectors, each bit is encrypted using a homomorphic cryptographic algorithm such as Elgamel public key encryption [134]. Addition and comparison operators are defined that allow auctioneers to determine winners without revealing bid values. The comparison operation is a two step process which involves vector multiplication and then left to right decryption in order to discover the highest bid without revealing individual values. The polynomial decryption scheme encodes bid values in the degree of a polynomial. A distributed pool of evaluators each calculates and publishes their "share" of the auction using dynamic programming techniques. Evaluators collaboratively discover the optimal value using a binary search, values are then reconstructed by interpolating the shares published by each evaluator.

The Garbled Circuits [130] protocol is another example of a secure auction protocol in which bid values are encoded using a pre-generated boolean circuit. The circuit acts as a black box – evaluating a given input and revealing nothing except a single output, thus obscuring the original value and any intermediary values. The circuit is constructed from a series of boolean gates by an *Auction Issuer*, the circuit is then "garbled" by applying a random permutation to the wires (this permutation is kept secret). The garbled circuit is distributed to bidders so that they can each encode their bid values using the circuit. The permutation of wires is used to establish a mapping table, which maps the garbled output value to the real output. Upon completion of the auction the auctioneer executes the garbled circuit using the garbled input and decodes the output using the mapping table sent by the Auction Issuer. One advantage of the Garbled Circuits protocol is the circuit can be constructed offline [135], therefore improving performance.

While DRIVE is designed to host secure auction protocols and includes several implementations of secure protocols, the protocols themselves are outside the scope of this thesis. A thorough taxonomy of secure auction protocols, worked examples of several secure protocols, and performance comparisons are provided in [128]

4.3.7 Currency

The DRIVE architecture is not designed to be bound to a particular type of currency [136] (virtual or real) to provide flexibility in scenarios represented. For example, virtual currencies can be used in internal deployments, in which allocation efficiency can be gained by using economic principles. Real currency is required in commercial environments, such as a federated environment with commercial Cloud providers. Due to the simplicity of creating closed virtual currencies most market based Grid systems use virtual currency. There are however, several major limitations of virtual currencies as identified by Shneidman et al. [137]:

- **Lack of liquidity:** which makes it difficult to exchange between real and virtual currency
- **Starvation:** when heavy consumers run out of currency
- **Depletion:** when users leave the system or hoard currency which has the effect of reducing the amount of currency available
- **Inflation:** as users are added to the system and given credits or credits are assigned to existing members artificially

In the DRIVE model it is difficult to create a true virtualised economy as there are (typically) few services offered in a computational market. Users are generally either a consumer or a producer, therefore currency does not flow circularly, rather the flow is in one direction leading to consumer starvation. Policies and administrative actions have been used to address the limitations of virtual currency. Karma [138] uses mathematical functions to compute currency value in an effort to offset inflation and deflation. Self recharging currencies [139] can be used to combat starvation at the expense of inflation, essentially creating a proportional sharing model combined with a virtual economy. Generally to create a successful virtual economy, the currency must be expressive and appreciated by users [137]. Using real currencies, or binding a virtual currency to a real currency can be used to encourage participation, provide lower barriers to entry for new users, and create a self sustaining environment [137]. The choice of currency is not within the scope of DRIVE, however the DRIVE design is equally applicable to both virtual and real economies.

4.3.8 DRIVE Allocation Architecture

DRIVE is designed to facilitate service trading using *any* economic protocol. The protocols discussed in this chapter differ greatly in terms of required architecture, auction and bid representation, winner determination algorithms, and communication between components. These differences motivate some of the design decisions and goals of this architecture. Briefly, the major requirements of the DRIVE allocation architecture are:

- **Provider and Task Independent:** Any type of task can be allocated and any type of provider can participate in the market.

- **Protocol Independent:** Any economic protocol can be used simultaneously in the market.
- **Distributed:** Distribution is required for performance, fault tolerance and scalability. In addition Allocation Component distribution is required by some protocols in order to guarantee security and privacy.
- **Decentralised:** Any Allocation Manager or Allocation Components can be used to ensure there is no single point of failure.
- **Scalable:** To represent large scale federated environments the allocation architecture must scale to allocate a huge number of tasks.

DRIVE Allocation Services

In DRIVE three services are used to create a scalable, distributed, decentralised and flexible allocation architecture. The Allocation Manager manages economic negotiation, the Allocation Context stores negotiation state, and Allocation Components are used to provide distributed protocol specific functionality. Due to the distributed and decentralised nature of the DRIVE architecture any Allocation Manager or Component can be used to provide the same functionality. Figure 4.7 shows the DRIVE allocation architecture.

The Allocation Manager is the central DRIVE component, designed to solicit task requests from consumers, oversee the allocation process, and provide a level of abstraction above the virtualised economy. The Allocation Manager acts as an intermediary between consumers and providers to facilitate market interactions, it is therefore independent from a particular task or provider model. To provide protocol independence a plug-in architecture is used, thereby allowing a range of protocols to be seamlessly selected and simultaneously used in different scenarios. Due to the wide ranging requirements of privacy preserving, trustworthy and verifiable economic allocation protocols the plug-in interface is generic such that it can be used to initiate the protocol, create protocol specific advertisements, instantiate distributed architecture (Auction Components), perform negotiation in protocol dependent languages, and determine allocation results following the protocol. To assist with these tasks the Allocation Manager offers a number of protocol-independent functions relating to the specific environment that can be applied to any protocol through the plug-in interface, for example: persistent state storage and retrieval, provider and DRIVE service discovery, allocation advertising, allocation monitoring, and the ability to publish or notify participants of results.

Allocation Managers are invoked by participants (consumers/providers) with a task or resource description, a selected protocol, and allocation options relating to the negotiation or the protocol. The chosen protocol is executed using the plug-in library, passing the description and any user defined options. The plug-in is then responsible for following the protocol and may use exposed Allocation Manager functionality to perform necessary tasks, for example to advertise the negotiation. The plug-in can instantiate and use required Allocation Components. Any messages received by the Allocation Manager relating to a particular instance are forwarded to

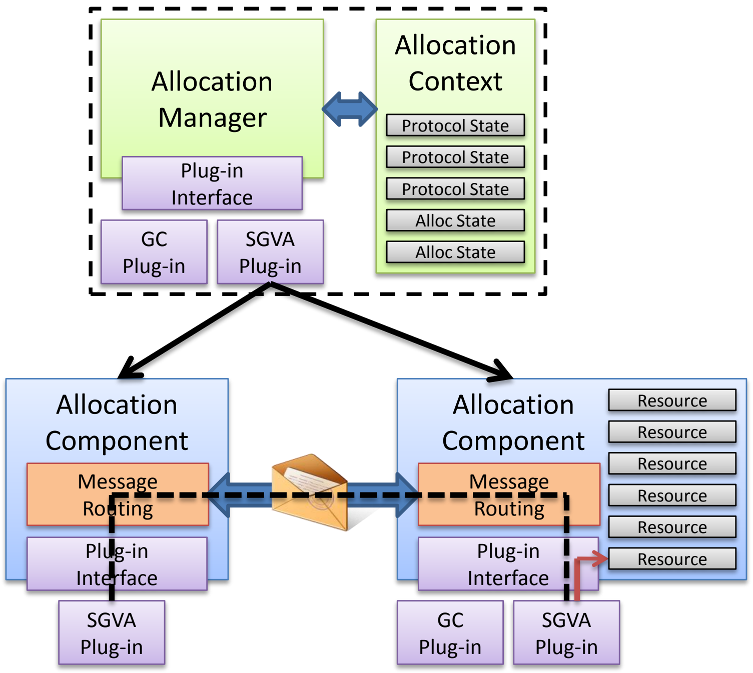


Figure 4.7: DRIVE Allocation architecture. Allocation Managers supervise the auction process. Allocation Context services store auction state and interact directly with a single Allocation Manager. Allocation Components are general purpose services that wrap protocol-specific functionality and are used to provide required protocol distribution.

the plug-in through the message passing interface, for example bids in an auction scenario. The Allocation Manager periodically monitors the allocation through the plug-in interface to ensure any user specified conditions are maintained, for example negotiation duration.

An Allocation Context service is associated with each Allocation Manager to manage both allocation state and protocol specific state. This separation of state management from allocation mechanisms follows standard design principles and provides separation from a particular hosting infrastructure. The Allocation Context abstracts the underlying stateful mechanisms, making the representation of state transparent to the Allocation Manager and exposing interfaces to store and retrieve state. While the DRIVE architecture assumes stateful web services, the Context service could store state in any way, for example on the file system or in a database. In the reference WSRF model, a factory pattern is used to create a *resource* for the duration of an individual invocation, the context service provides transparent access to this resource by embedding a resource key in the WS-addressing invocation. The Context service is also designed to provide standardised persistence and notifications.

As outlined in the previous section economic protocols have different requirements in terms of representation, winner computation, and topology. In some cases the topology forms the basis for providing trust guarantees, for example protocols relying on threshold trust require a number of components to collectively determine the winner. In DRIVE, Allocation Components are used to “wrap” protocol specific functionality with a service interface. For a single protocol multiple Auction components can be instantiated as any particular entity in the allocation process (for example, Auction Evaluators in the SGVA protocol or Auction Issuers in the Garbled Circuits protocol). To support this extensibility the Auction Component exposes a protocol plug-in interface allowing any protocol library to be dynamically loaded and used. A generic message passing interface is exposed through a simple service interface, the appropriate plug-in can then be executed to perform the required action based on the protocol specific message. The message passing interface removes the need to pre-define a typed interface for all possible Allocation Components. Auction Components also maintain state about each economic negotiation they are involved in, the plug-in interface includes functionality to store and retrieve state from stateful resources stored in the Auction Component.

In addition to these allocation services DRIVE Agents represent service providers in economic negotiation using a similar plug-in interface. Service providers and the associated DRIVE Agents are discussed in Section 4.4.

4.4 Service Providers

Service providers offer functionality to consumers in exchange for payment in the DRIVE marketplace. DRIVE places no requirement on the type of service offered. Theoretically a service could represent anything, ranging from low level resource abstraction through to specific application functionality. There is also no requirement for a particular type of interface, for example a service could be offered through a Web service, API, or specialised client. In a computational Grid, Globus GRAM is considered a service provider, offering a transparent interface (WS or pre-ws) for users to access distributed computational resources. In a SOA environment, a service may offer a particular function, for example producing a seismic map (in CyberShake) or executing a bioinformatics application like BLAST (Basic Local Alignment Search Tool) to compare biological sequences in the transposon workflow.

To participate in a DRIVE market service providers must be represented in negotiation and contract validation before service invocation. It is unrealistic to assume every service provider will alter their implementation to interact with DRIVE, rather it is our view that this functionality can be somewhat detached from the service provider. In DRIVE, a specialised DRIVE Agent is used to act on behalf of the service provider, this Agent acts as a “gate-keeper” for the service provider by participating in negotiation and authorising service invocation. Unlike the service provider the DRIVE Agent must be implemented following the defined DRIVE interfaces.

4.4.1 DRIVE Agent

The DRIVE agent is designed to represent providers in the DRIVE market by following protocols and exposing an interface for invocation from the market. The agent handles provider registration, DRIVE service discovery, economic negotiation, and contract management in the marketplace. The major goals of the DRIVE Agent are task, protocol, and provider independence so that a DRIVE Agent can represent any type of provider using any task model and interact with the market using arbitrary economic protocols. It is also important for the DRIVE Agent to be highly configurable and autonomous with respect to negotiation and valuation policies so that administrators can define requirements and behaviour in different scenarios. Service providers need to be able to update capacity, determine with whom to negotiate (trust, reputation, VO membership), and calculate appropriate service or resource valuations based on requests and available capacity. The DRIVE Agent therefore exposes interfaces allowing administrators to express such requirements.

Figure 4.8 shows the architecture of the DRIVE Agent. Communication between the DRIVE Agent and market occurs through a plug-in economic protocol. Like the Allocation Manager plug-in protocols are loaded dynamically and can be used simultaneously to participate in negotiations. The DRIVE Agent includes functionality to receive and respond to allocation advertisements or negotiation requests and also advertise resource profiles describing real-time capacity. The DRIVE Agent supports user defined plug-in policies (Section 4.4.3) and pricing functions (Section 4.4.4) to determine risk and value requests. A generic service monitoring interface is also defined so that the DRIVE Agent can obtain real time service capacity, the implementation of which is domain specific. For example in a Grid the system monitor may report resource usage (CPU, Memory, I/O) in a Storage Cloud the system monitor is more likely to focus on available storage. During the negotiation process the DRIVE Agent considers all of these aspects when determining valuing resources, this process is detailed in Section 4.4.4.

The DRIVE Agent has been designed as a separate entity to reduce the burden of entering a DRIVE market. However, DRIVE Agent functionality could also be integrated into the service provider, this has the advantage of providing a single contact point for consumers and DRIVE entities alike, while also allowing tightly coupled valuation and participation policies. An approach for creating DRIVE-enabled services is outlined in following section.

4.4.2 DRIVE-enabled Services

One difficulty faced when creating service based architectures is Web enabling existing applications. Often the task of "wrapping" an existing application with a Web or Grid service interface is more difficult than re-writing the entire application due to the complexities involving underlying technologies. Several toolkits have been created with the aim of simplifying this wrapping process, for example gRAVI (Chapter 7), Opal [140], and the Generic Service Toolkit (GST) [141]. These toolkits are based on a requirement to expose distributed applications through Web services for use in workflows or to facilitate scientific collaboration. As such the services produced

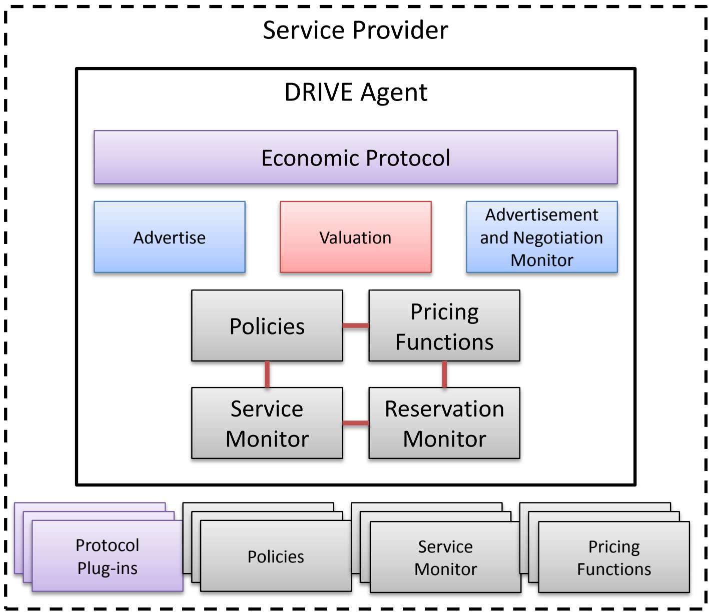


Figure 4.8: DRIVE Agent architecture. The DRIVE agent communicates through plug-in economic protocols. Valuation of requests considers a range of data including policies, service capacity, and pricing functions. The Agent is also responsible for advertising capabilities and discovering existing negotiations.

are “bare-boned” without much of the functionality expressed in custom built services.

In an economic oriented environment web enablement is the first step in exposing an application in the marketplace. The previously mentioned toolkits could all be used to deploy an application service which can be invoked in an economic environment, however the service also requires mechanisms to interact with the market in order to negotiate usage in exchange for payment. In DRIVE this can be accomplished by deploying a parallel DRIVE Agent alongside the generated service, however this separation requires additional complexity to enable interaction between the two services (security, APIs). A better solution is integrating DRIVE Agent functionality during service generation to create an economically enabled service. gRAVI provides the first example of a wrapping toolkit able to generate an economic (DRIVE) compliant Web/-Grid service. gRAVI services contain all the interfaces and artefacts required to participate in a DRIVE marketplace abstracting technical implementation details from providers and administrators. The generated service exposes a plug-in interface allowing provider specific implementation of valuation functionality and a plug-in protocol interface to support arbitrary economic protocols. In addition customisable DRIVE Agent policies are provided which can be used to control service interactions in the marketplace. The design and implementation of gRAVI is presented in Chapter 7.

4.4.3 Service Provider Policies

The process of deciding to participate in negotiation and valuing resources are major complexities in economic resource allocation. In a reverse auction protocol the decision making process includes a number of considerations from the perspective of the provider. Initially the provider must decide whether or not to participate, this involves deciding if the client and DRIVE components are trustworthy, determining if there is sufficient capacity, and ensuring the protocol is supported. The provider must then determine a value for the goods in order to negotiate. Similar processes are undertaken on the part of the consumer. In computational domains policies provide an extensible, adaptable and standardised way of representing such decisions.

In DRIVE a range of user defined policies can be expressed. For example, policies regarding reputation of entities can be used to determine the trust in DRIVE components or other users. Policies regarding current load, projected load, overbooking and reservations can be used collectively to determine capacity and contribute to the assessment of risk. This information can then be combined with pricing functions to determine the value (and risk) to a provider (or consumer) of a given provision. The range of policies that could potentially be applied in DRIVE is endless as DRIVE is designed to model real world economies composed of competitive self-interested agents. For example agents could decide to intentionally violate existing agreements in an attempt to increase revenue from hosting another task. This scenario could be represented by policies regarding selective violations.

Policy Language

Policies are commonly used in a number of areas, most notably security - administrators define a set of rules and practices for how an organisation manages and protects information or resources. Many of these policy languages are not limited to use in the particular domain for which they are designed and can be extended or used directly to make decisions in other areas. Due to the amount of research into policy languages there are various different implementations available ranging from formal policy languages to rule-based languages using if semantics.

In Web services standards there has been an effort to create XML-based languages governing access control, alternatively ontological languages such as the Web Ontology Language (OWL) [142] have gathered support as they are more powerful in terms of semantic expressiveness and provide simple reasoning capabilities. Rule based engines such as Jess [143] are also commonly used to express policies in Java applications. The policy description languages considered for DRIVE were eXtensible Access Control Markup Language (XACML) [144], Enterprise Privacy Authorisation Language (EPAL) [145], and Ponder [146]. Semantic languages KAoS [147], RuleML [148], and Rei [149] were also briefly examined to complete the comparison of policy languages.

XACML is an OASIS standard that defines an XML based architecture providing functionality to create, evaluate and enforce policies. In addition to the access control policy language it also defines a request/response language. XACML enables the use of a decision point to store and evaluate user defined policies. XACML policies are customisable and therefore facilitate user defined functions, making it well suited for DRIVE as policies must be highly customisable and able to be applied in different domains (provider types). In XACML, policy rules describe desired behaviour in abstract terms. The Policy Decision Point (PDP) makes the decision whether or not to authorise a request and the Policy Enforcement Point (PEP) enforces the policy decisions. The major issue with XACML like many other XML based languages is the verbose nature of the policies, however in many situations this is also beneficial as policies are human understandable.

EPAL is an XML based functional language developed by IBM to represent enterprise security policies. EPAL is very similar to XACML in most respects including representation, functionality and policy enforcement model. A comparison of features of the two found that EPAL supports a subset of almost all features included in XACML [150]. Ponder is a widely used policy language designed specifically for management and security of distributed systems. Ponder policies are described using an extensible object oriented model which allows definition and multiple instantiations of policies. The language however is designed to specify general policies with little interdependence and the language itself is somewhat complex.

The DRIVE extensible policy architecture is based on XACML policies for a number of reasons. Most importantly XACML is standardised by OASIS and under continual development. It is also one of the most widely used and mature policy languages available with multiple implementations. XACML is generic (not bound to a particular domain), platform independent, flexible in how it is used, and extensible in terms of its XML representation and the ability to define new data types, functions and rules. The XML representation is easily understood allowing non-technical users a simple way to define complex policies. Policies in XACML can also be distributed and may refer to other policies therefore creating complex policy structures involving extension of distributed sub policies.

4.4.4 Valuation

DRIVE attempts to model the real world when valuing a resource (or service). To do this entities assess the risk of a negotiation (using policies) and apply a pricing function to calculate their true value. In any economic environment there is an inherent level of risk in the actions of participants. Techniques such as risk assessment and risk management [151, 152] are commonly used to consider the possibility and consequences of unforeseen events. In DRIVE providers may enter into a contract without full knowledge of the likelihood of being able to meet contractual requirements. For example, it is hard to predict resource or network failure, overbooking issues, or possibly even natural disaster. Risk management considers the possibility that future events might negatively affect an entity in the system and attempts to quantify the impact of such an event. In DRIVE, pricing functions are used to determine a specific price for a set of goods, maintaining relativity between similar goods given the current conditions (supply/demand). Both consumers and providers use pricing functions to determine market prices, for example in an auction each party's valuation is reflected in its reserve price or bid.

Risk Assessment

In a market based environment like DRIVE, resource providers incorporate risk in their cost model. Like the real world, providers individually evaluate risk and amortise this cost over all clients by adding an overhead to calculated prices. The more consumers a provider has and the more negotiations entered, the lower the overhead becomes.

Risk assessment is based on the probability of an event occurring and its impact calculated for all possible circumstances [151]. DRIVE uses the notion of a *risk factor* allowing participants to assign a risk value to each piece of information considered. This risk value is calculated by using local policies to determine a level of risk for a specific information source. For example, the reputation of a client may be considered high risk if previous interactions have resulted in contract breaches, or low risk if previous interactions have been favourable. The computed risk factor is then applied to the participants valuation to incorporate the potential for unforeseen events.

In DRIVE there is potential to extended the ability of a provider to amortise risk by adding third-party insurance [153]. This approach would allow external insurance providers to cover individual resource provider risk. An insurance model is well suited to an environment like DRIVE as resource providers could pay a single fee that insures the provider against the cost of broken obligations, thus reducing the impact of a single disastrous event. Insurance providers also benefit as profit can be made because claim distribution is more reliable with a large number of customers.

Pricing Functions

Pricing (or Bidding) functions are used to determine a price for a set of goods (resources) based on market conditions. Pricing functions define how different metrics are combined to determine the private value of a good. Pricing is one of the biggest difficulties in market oriented distributed computing as there is a lack of standardisation, for example providers have different resource representations and there are no well defined "units". Cloud computing has attempted to standardise computational units with the introduction of VMs representing defined capacity, although the posted price mechanisms are static. Dynamic pricing schemes (as supported by DRIVE) can provide financial gains for participants based on supply and demand analysis, however these approaches are more complex and may include both functional (VMs, CPU, Memory) and non-functional (software availability) resources. In the past there have been a number of different approaches to Grid resource pricing such as linear functions, shared market services, and adaptive pricing techniques.

Most economic systems make use of generic linear pricing schemes. For example Cloud providers typically employ a linear utility model in which prices are based on the amount of resource usage. Priority and weighting time based mechanisms have been used to increase prices based on faster service, for example the pricing schemes used by Distributed Grid Accounting System (DGAS) [154] in EGEE. Collective bartering approaches have also been used allowing consumers and producers to collectively determine an appropriate price [155], however both parties must still specify a valuation range. In [156] prices are determined using a projected load curve to establish a utilisation value, current market information and policies are then applied to determine the final price. In this model an eagerness factor is included to specify how urgent the request is, this adds priority based pricing to the model. In [157] real options are applied to price grid commodities (for example CPU and Memory). In this model commodity availability fluctuates over time between the present time and an arbitrary point in the future which reflects the option to wait. Real option theory is applied by formulating the problem as a real option pricing problem. Real option pricing models originate in investment domains and are used to determine stock option prices. A fuzzy algorithm is used to define a price variant factor, which determines a price aimed at maximising Grid usage.

Global market services have also been used to determine global prices for goods, however these approaches violate the competitive supply and demand properties of open economies. For example DGAS includes Price Authority services that use different pricing functions to assign prices to global resources. DGAS pricing algorithms include static pricing functions and dynamic pricing functions that can be used to increase prices for jobs requiring lower wait times. Caracas et al. [158] present the design and implementation of a generic pricing service capable of representing different pricing schemes. The pricing mechanisms used are capable of determining dynamic prices for information and computational elements by applying functions related to resource utilisation. This approach lacks autonomy and would require a level of trust in the pricing service.

Techniques such as k-pricing have been proposed in situations where both consumer and provider have individual value ranges (or reserve prices) [159, 160, 161]. Essentially k-pricing acts to distribute social welfare across participants according to a predefined factor, this factor is used to balance the prices of each party in an auction. K-pricing techniques have also been applied to SLA penalties [162], in this work prices are determined using functions of the quality of service, k-pricing is used to assign penalties based on potential service levels. When the provision is completed the delivered service level is compared against the predefined levels and an appropriate penalty is invoked.

At present, most pricing research relates to Grid environments, however research regarding pricing in Cloud providers is beginning to be published. For example, Anandasivam et al. [163] apply Revenue Management concepts to service requests in a Cloud environment. In this work providers are presented with consumer offers, prices are then developed based on an approximation of the opportunity cost of reducing available capacity.

Pricing functions are an open research area in economic distributed computing. In DRIVE, pricing is viewed as a responsibility of the entities within the system as such it is not a focus of this thesis. DRIVE has been designed such that DRIVE Agents and consumers can implement arbitrary pricing functions to establish values for resources. To demonstrate this design several pricing functions have been implemented and evaluated in Chapter 6.

4.4.5 Auction Bidding Strategies

As discussed in Section 4.3.4 many auction protocols have dominant strategies under particular market conditions. For example in a private value Vickrey auction truthful bidding is the dominant strategy. However, for non-private value auctions other strategies may provide better payoffs under particular market conditions or when using different protocols. For example in non-private value auctions the value of a good is dependent on others valuations, therefore determining another providers value for a good can be beneficial when bidding. In open outcry auctions there is motivation to withhold a bid until the last possible moment to avoid price increases due to incremental bidding. Bidding strategies describe payoff maximising approaches which can be used during or after determining a local value for a set of resources. The design of the DRIVE Agent is such that any strategy could be implemented through the plug-in valuation interface. The following section summarises potential valuation strategies and protocol bidding strategies that can be employed in an auction scenario.

Valuation Bidding Strategies

Valuation bidding strategies describe techniques for determining a bid value, based on other entities actions or market conditions. The base bidding strategy against which all other strategies are compared is truth telling. In a truthful strategy bidders bid their own valuation based on the current situation without consideration of other agents actions.

Zero Intelligence Plus (ZIP) strategies can be used to create bids based on projected profit margins. ZIP strategies are widely used in real world Continuous Double Auctions (CDA) and have been shown to out perform human traders in these markets [164]. They have also been successfully applied to sealed-bid auctions [165] even though the information released is substantially less than a CDA (only winning bidder and price). In [166] ZIP agents use public market information to alter bid values. ZIP agents calculate profit margins based on the difference between a valuation and a bid, they then increase or decrease the projected profit margin based on if the last action was a bid or an offer (in a CDA protocol).

Q-Strategy [167, 168] makes use of reinforcement learning to create bidding rules based on previous market interactions. Q-Strategy takes a Q-Learning approach [169] with an e-greedy selection policy. In this strategy market information is collected in an *exploration phase*, collected information is then used to generate more intelligent bids in the *exploitation phase*. The exploration phase involves generating random bids for different job types to establish market conditions. When the job is executed on the host the outcome is calculated (reward/penalty) and stored. A table of bids, executed jobs, and payoffs are maintained in the Q-Table. The Q-Table contains Q-Learning dimensions (state, action, payoff) where the state represents the job type, action represents bids, and payoff is an aggregate function calculated from similar jobs and their bids. The Q-Rule then defines an aggregate payoff function based on this data. In the exploitation phase, bidders select a similar job type from the Q-table and choose the bid price which resulted in the highest aggregated past payoff, therefore maximising their expected payoff. One issue with Q-Strategy is the expensive startup exploration phase in which information is obtained through random bidding. In DRIVE this could be optimised as information could in part be obtained through Publishing Services thus allowing more efficient exploration.

Protocol Bidding Strategies

Auction protocols have additional bidding dimensions that can be exploited by profit maximising strategies. One of the most common strategies employed is to determine the best time to place a bid. Both early bidding and late bidding have been shown to provide advantages in different domains. If bidders have a fixed willingness to pay there should be no effect of when a bid is placed (except based on time based tie-breaking rules).

Various research has looked at Last minute bidding (or “sniping”) in the context of open outcry online auctions [170, 171, 172, 173] and proxy bidding scenarios. Typical motivation for last minute bidding is to combat “shill bidders” (fake bidders raising the price) and incremental bidding (bidding in increments rather than bidding ones true value or proxy bidding). In addition sniping provides a way for a bidder to win an auction with a bid less than another bidder’s true value, if other bidders do not have time to react to the snipe. Two environmental properties encourage late bidding in online auctions: First there are typically a large number of similar goods which gives bidders an opportunity to bid late and risk losing, secondly there is an opportunity to determine an items common value from observing other bids. Therefore a bidder has more knowledge about an items value after all bidders have bid [173]. Some online auction sites (for example TradeMe[2]) minimise the effect of shill bidding by automatically extending auctions, this has the effect of increasing the revenue generated and could potentially be applied in a DRIVE auction.

Conversely, in some environments bidding early can have the effect of scaring off other bidders, therefore resulting in a lower price. Late bidding in this scenario results in higher prices due to increased competition. Bramsen [174] presents a study of 17,000 online furniture auctions, showing that early bidding results in significantly lower prices. This fact is established through comparison between the ratio of final price to third party valuation. In this experiment, the goods on sale are more unique than those seen in other online domains which may account for some of

[2]http://www.trademe.co.nz

the advantages seen when bidding early.

It has been thought time based bidding strategies are only valuable in open-outcry auctions, due to the information learned over time. However, as outlined in Section 5.7.4 Just-In-Time (JIT) bidding in sealed bid auctions can be used as a means of reducing the effect of auction latency in distributed auctions. Chapter 6 provides the first quantifiable analysis of JIT bidding in a sealed bid context, showing the potential utilisation advantages gained by reducing effective latency and therefore establishing more accurate capacity predictions.

4.4.6 Advanced Reservation

Advanced reservation support in distributed systems is desirable for performance predictability, meeting resource requirements, and providing QoS guarantees [175, 79, 176, 177, 81]. As Grid and Cloud systems evolve the task scheduling requirements become more complex, requiring fine grained coordination of interdependent jobs in order to achieve larger goals. Often tasks require particular resources (or services) to be used at certain times in order to run efficiently. For example, a task may require temporary data storage while executing and more permanent storage after completion. Service orchestration in a workflow may express dependencies between service invocation. In an economic framework, users can take advantage of scheduling opportunities to provision services during "cheaper" usage times when resources are potentially underutilised.

While these advantages relate specifically to consumers, the use of flexible reservation windows has also been shown to provide scheduling benefits to service providers [178]. Flexible advanced reservations in particular allow providers the opportunity to dynamically schedule independent tasks therefore increasing occupancy and utilisation. Flexible reservation is analogous to deadline constrained allocation [41] as deadlines have an associated cost function allowing providers the flexibility of scheduling based on cost.

DRIVE is designed with a flexible reservation model, allowing consumer definition of a "reservation window" associated with a task. This window defines the duration of the task within a given time period (start and end time) leaving specific execution time to the provider. To support advanced reservation by providers, DRIVE includes a Reservation Service to store and schedule commitments in a secure manner. The Reservation Service allows future commitments to be considered by providers prior to negotiating for subsequent allocation, it also supports arbitrary advanced scheduling techniques to optimise task execution. While there are countless examples of reservation enabled schedulers, brokers, and resource managers they are all bound to the domain for which they are designed. The DRIVE Reservation Service has been designed to be domain independent, allowing reservations to be stored for any class of task supported.

4.4.7 DRIVE Reservation Service

The DRIVE Reservation Service has been designed to meet a number of goals, each of which has influenced architectural decisions. The goals of the DRIVE Reservation Service are:

- **Provider and Task Independent:** As per the goals of DRIVE, the Reservation Service must support arbitrary task models and provider types; therefore reservations must be flexible and extensible to include an arbitrary term language.
- **Synchronised Reservation Management:** If providers have existing reservation systems it is important to provide a synchronised view of the resources, this requires mechanisms to propagate reservations to LRMs or schedulers and also update DRIVE reservations based on local reservations.
- **Secure:** Reservations may contain commercially sensitive information (pricing, clients, QoS) which must be kept private.
- **Standardised:** All parties in a DRIVE market must be able to understand the structure of a reservation.
- **Fast and Efficient:** A major goal of the reservation service is fast efficient reservation retrieval so that DRIVE Agents can obtain current and projected utilisation immediately.
- **Persistence:** Reservation information is crucial to provider operation, therefore reservations must be maintained and reloaded if the Reservation Service fails.
- **Flexible Scheduling Support:** The Reservation Service must support user defined scheduling algorithms such that administrators can easily optimise operation of the provider in different scenarios.

The Reservation Service is a provider level service that manages reservations for a single provider or group of providers. The Reservation Service is designed to store information regarding all aspects of negotiation in DRIVE, as such, there are different types of reservations stored representing the multi phase negotiation mechanisms used (negotiations, agreements, and contracts). Ongoing negotiations and tentative agreements can be just as important as contracts when determining risk. Local policies determine what reservation types are stored and how they are interpreted during valuation. Use of the Reservation Service in DRIVE is not mandatory, in fact providers may choose to implement their own reservation functionality or exploit existing reservation capabilities. The Reservation Service however provides an efficient mechanism to store and manage negotiation state and contractual obligations in DRIVE. There are several reasons why there has not been widespread adoption of advanced reservations in the Grid community [175]. In particular, advanced reservations have been shown to cause performance degradation [179], many mechanisms lack flexibility and scalability [180], and often they do not address user needs and system requirements. There are also a number of reasons as to why existing reservation architectures are not suitable for use in DRIVE. Most importantly existing reservation support is limited and is not standardised amongst schedulers and resource managers. DRIVE is designed to not be bound to a single domain and therefore it must support reservations as a first class entity for all domains, such as Cloud and SOAs.

In DRIVE, it is important to interact with existing infrastructure, for this reason the Reservation Service is designed to synchronise with existing reservation compliant LRMs. The Globus Advanced Reservation Service[3] (GARS) can be used to provide this functionality, as it provides a transparent interface to existing reservation-enabled schedulers to create, manage, and synchronise reservations with underlying resource providers. GARs can be configured to obtain reservation information from a range of commonly used LRMs and schedulers.

Reservations in DRIVE are kept securely so as not to release potentially sensitive reservation information to external parties. A reservation may store detailed information relating to clients and their jobs, as well as pricing information and QoS agreements established between parties. Fine grained security policies may be implemented to allow clients (or brokers) the ability to monitor the status of their reservations through a publicly exposed interface. Importance in the design has been placed on providing efficient reservation creation and retrieval, this is achieved by using an indexed schedule structure. The schedule is indexed by time so that providers can efficiently determine real time and predicted capacity. In addition scheduling algorithms can act on this schedule to optimise execution times. Arbitrary user defined scheduling algorithms are supported through a plug-in interface.

4.4.8 Reservation Scheduling

Advanced reservations open up many possibilities in terms of using advanced scheduling techniques to optimise resource utilisation. Fragmentation can occur when tasks are added to the schedule without optimising utilisation by planning execution times. In DRIVE advanced reservation requests are flexible in that they specify a window of time in which they require execution. Without planning, large gaps can appear in the schedule resulting in underutilisation of resources and tasks being turned away as there is no suitable slot within the defined window. Figure 4.9 illustrates this problem, in this example the first diagram shows an unoptimised schedule with six tasks scheduled to be executed over a period of time. The second diagram shows a task being submitted with a small execution window, without rearranging the schedule the agent would not be able to consider hosting the task as it conflicts with an existing reservation. The third diagram shows an optimised schedule with the same tasks arranged differently (within their individual reservation windows), the new task can now be hosted at the designated time and there is greater utilisation of resources over a shorter period of time which may result in higher revenue for the provider.

This type of schedule optimisation has resulted in an extensive body of research ranging from scheduling tasks on supercomputers [181] to complex allocation in clusters [182] and co-allocation reservations in Grid systems [183, 184]. Scheduling approaches include algorithms such as first come first served (FCFS), shortest job first, best fit, priority based, backfilling, greedy filling, and heuristic based approaches. Many of these algorithms originate in batch or queue based systems and are focused on parallel job execution. In traditional Grid environments scheduling algo-

[3] `http://confluence.globus.org/display/gars`

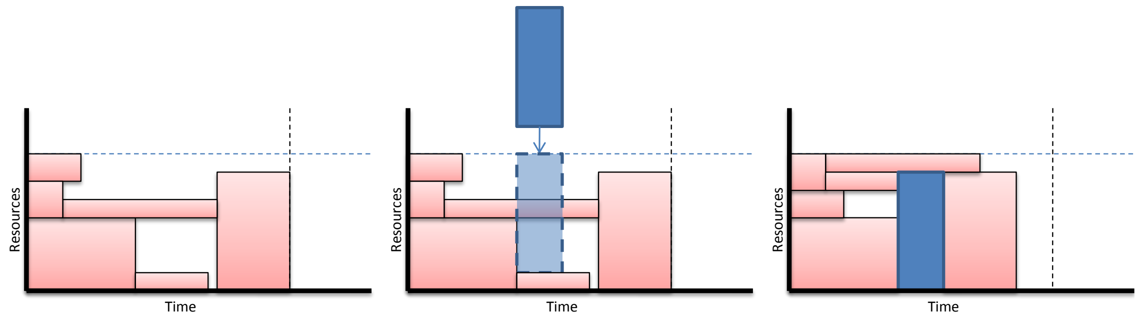


Figure 4.9: Optimising a resource schedule using scheduling techniques. Without intelligent scheduling the submitted task cannot be run even though there is available capacity, however using advanced scheduling techniques the other tasks can be scheduled in such a way as to optimise the overall schedule.

rithms have been extended to consider deadlines and QoS constraints placed on tasks. However in a DRIVE scenario deadline and QoS analysis is conducted during valuation and therefore algorithms can be used to organise the schedule at any time. The simplistic approach of allocating tasks by the order of arrival (FCFS) leads to issues of underutilisation and fragmentation as is seen in Figure 4.9 . Conservative and aggressive Backfilling [185, 186, 187, 188] approaches improve this situation by moving small jobs forward to fill holes in the schedule. Heuristic approaches [189, 190, 191] can be used to select the best task to move into the fragmented hole in the schedule.

In DRIVE Agents act on behalf of resource providers which in turn act on behalf of groups of resources, in this environment the scheduling problem is made even more complex when considering multiple co-allocation requests that require synchronised execution over a pool of resources. Scheduling techniques, such as those outlined in [184], are optimised to take into consideration distributed co-allocation and could be applied in DRIVE.

Scheduling algorithms are not within the scope of this thesis. However the Reservation Service has been designed to be extensible so that arbitrary scheduling algorithms can be implemented. This scheduling can take place when tasks are added to the schedule or when tasks are considered for hosting. Polices regarding negotiations, tentative agreements, and hardened agreements can be used to determine the relevance of each to the scheduling process.

4.5 Contract Management

Having negotiated terms of service provision through an economic protocol the participating entities (providers and consumers) are inherently expected to honour their obligations. However due to the self interested and competitive nature of participants there may be motivation to violate agreements for economic gain or malicious reasons. As in real world economies contractual arrangements can be established to ensure each party honours their obligations, allowing consumers and providers to define and enforce specific terms of service usage and associated incentives (rewards/penalties) for honouring or violating agreements.

In DRIVE contracts describe SLAs between consumers and providers. A SLA contains a collection of specific requirements and QoS metrics to be delivered. Standardised representation is required as SLA descriptions may differ depending on the perspective of the entity, for example they may be task or service requirements in the case of a consumer or individual service levels for providers.

4.5.1 Service Level Agreements

A SLA represents an agreement between a service consumer and service provider for a particular provision, as such it is a favoured model of providing assurances in distributed systems. The agreement may be a 1-1 mapping between consumer and provider or possibly n-m between multiple consumers and/or multiple providers. A SLA is made up of a number of Service level Objectives (SLO) that define particular QoS requirements that must be maintained by the provider. These individual SLOs form the measurable parameters of the SLA and can be monitored during the provision to ensure service levels are met. SLAs contain information regarding the parties involved, time periods resources are used, and incentives for honouring obligations. There are a number of results at the conclusion of a SLA - there could be financial penalties/rewards enforced, reputation information formed for use in future transactions, or mutual resolution between participants such as re-execution of the agreement.

There is much literature regarding creation of SLAs in terms of advertising and posted price markets. However, there is much less research into dynamic market based mechanisms such as auctions. Typically in existing systems resource allocation and SLA negotiation are simultaneous - the negotiation is effectively the allocation. In DRIVE, SLA negotiation takes place after initial allocation. This initial allocation process considers loosely defined QoS parameters and results in a tentative agreement between parties. The parties then participate in a more concrete negotiation process to define the specific QoS constraints in the form of an SLA.

There have been many systems designed to define SLAs. The *Contract Net Protocol* [104, 192] was one of the first protocols for negotiating electronic service contracts. Alternative languages for specifying SLAs include: *WSLA* [193], *SLAng* [194], *WS-Agreement* [78] and *RBSLA* [195]. However, none of these approaches explicitly consider an economic term language [72]. WS-Agreement provides a flexible framework in which a domain-specific term language can be defined and used. WS-Agreement has become the de facto standard due to its flexible nature and OGF support. This is demonstrated by many examples of its use in differing architectures, such as Cremona [196], SWAPS [197], and several projects; including SORMA, AssessGrid[4], and SLA@SOI[5]. The main issue with WS-Agreement is that its protocol stack is not designed to sup-

[4]http://www.assessgrid.eu/
[5]http://www.sla-at-soi.eu/

port auction-orientated SLAs, as it is designed for bipartite negotiation [198]. Despite this, it has been successfully used with auction mechanisms in projects such as SORMA using a proprietary economic term language.

WS-agreement has been chosen as the contract representation language for DRIVE due to its flexibility, extensibility and widespread adoption. WS-agreement has essentially become a de-facto standard for SLA representation in the wider Grid community. WS-Agreement also fits with the design of DRIVE in that it is standardised and task independent as any task description language can be used to form the description terms of the agreement. The economic representation limitations can be overcome by defining additional specialised term languages as is the case with SORMA. However, a slightly different approach has been taken in DRIVE, rather than extending an existing language a separate economic language has been defined to represent the economic aspects of negotiation. There are three major reasons for this approach, first an important goal in the design of DRIVE is independence from a specific class of task, a separate economic language separates the economic properties from the task description rather than binding the description to a specific term language (such as JSDL). Secondly there may be a requirement for an extended (or different) economic language depending on the protocol used, this is easier using a separate language which may be extended or replaced. Finally in a federated environment, providers may have existing economic representations (and possibly contract/agreement architectures), it therefore makes sense to use a standardised representation like WS-agreement with the ability to map term representations rather than requiring providers implement proprietary non-standardised representations.

4.5.2 Contract Incentives

Enforcement of contracts is difficult in any environment, not least of which distributed architectures as providers are under independent administration. Rather than enforcing appropriate behaviour DRIVE relies on incentives to encourage participants to act in a desired manner. Incentives are generally thought of in two categories: positive and negative, positive incentives motivate entities to act a particular way by rewarding them for their actions, negative incentives take the opposite approach by punishing entities that do not act appropriately. Incentives can be further categorised as *remunerative*, *moral* or *coercive* depending on the way in which agents are motivated. Remunerative incentives are based on an expected reward (or penalty) in exchange for their actions, typically remunerative incentives are financial. Moral incentives exist when it is generally accepted that one course of action is the correct behaviour, entities acting against a moral incentive expect condemnation from the community while those that follow the moral incentive are looked at favourably. Moral incentives are similar to proportional share models. Coercive incentives are based on a physical force being applied to entities that fail to act in an appropriate manner, for example actions such as imprisonment or confiscation. In DRIVE, financial rewards and penalties are considered remunerative incentives, reputation can be thought of as a moral incentive, and VO suspension or expulsion is a coercive penalty.

In economic systems financial penalties are the most commonly used means of providing both positive and negative incentives. Financial incentives are central to economics both in individual decision making and in co-operative or competitive reasoning and can therefore be considered in economic analysis of the system. Positive financial incentives are inherent in economic systems such as DRIVE, due to the fact providers are rewarded for services provided through payments by consumers. In this situation if a consumer is unhappy with the service received they may choose not to pay. However, if a service is half fulfilled it is difficult to establish the value of the violation. Penalties provide a way of categorising breaches and punishing poorly behaving users, in DRIVE a set of penalties can be outlined for differing levels of contract breaches.

The DRIVE architecture focuses on financial incentives to encourage agreement compliance, other approaches to enforcement are outside the scope of this thesis and are considered future work. DRIVE contracts define a set of rewards and penalties for each contract term. A binary measure of fulfilment is used - if a service level is delivered then a reward is paid, if the term is not fulfilled then a penalty is invoked. This approach provides a degree of flexibility allowing fine grained terms (and matching rewards or penalties) to be specified.

4.5.3 DRIVE Contracts

DRIVE contracts are expressed using WS-agreement with individual terms described in a domain-specific task description language. A unique two phase progressive contract creation process is used to mitigate the effects of allocation latency by *hardening* tentative agreements into contracts [199].

Progressive Contract Creation

There is potential for considerable latency in the DRIVE allocation process (auctions, tendering) between providers participating in economic negotiation and winner determination/confirmation. This latency may restrict providers participating in future negotiations due to lack of knowledge of the outcome of ongoing or previous negotiations. Providers have two approaches to this issue, they can reserve resources for the duration of the negotiation or they can wait until the result of the allocation before resource reservation. Neither situation is ideal; initial reservation leads to underutilisation as a negotiation typically has one winner and multiple losers, late reservation results in contract violations as resource state may have changed between negotiation and reservation. To mitigate the effect of latency DRIVE implements a progressive two phase contract mechanism to reflect the various stages of negotiation. Providers can therefore take into account the possibility of allocation when considering future negotiations.

The two phase contract structure is shown in Figure 4.10. As the result of an allocation a tentative agreement is created between the user and winning provider(s) (phase 1), before redemption this agreement must be *hardened* into a binding agreement (or contract) that defines particular levels of service to be delivered along with a set of rewards and penalties for honouring or breaking the agreement (phase 2). Typically the tentative agreement is not strictly binding and penalties for violations are not as harsh as for breaking a binding contract. The motivation for this separation is twofold, first it encourages providers to participate in allocation in the knowledge they will not be penalised as harshly for pulling out at an earlier stage. Secondly it facilitates overbooking (Section 5.7.1) which has been shown to increase revenue as some percentage of offers made will not result in allocations [200].

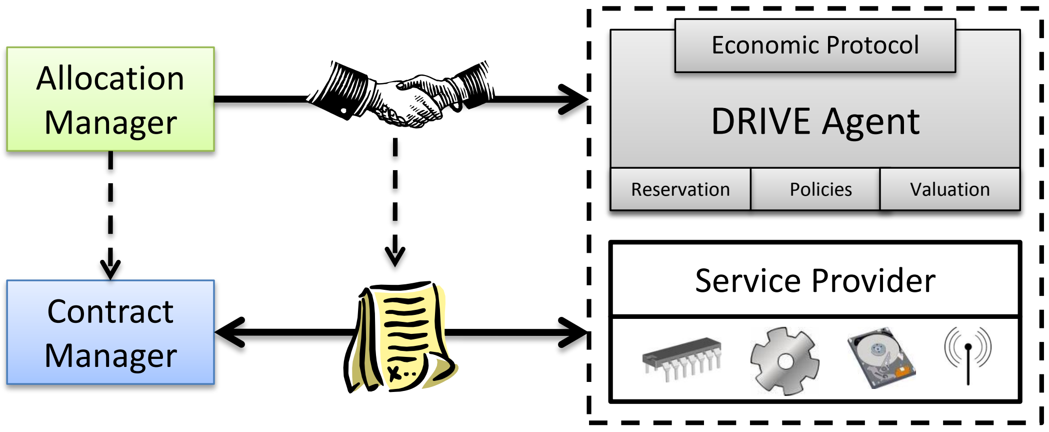
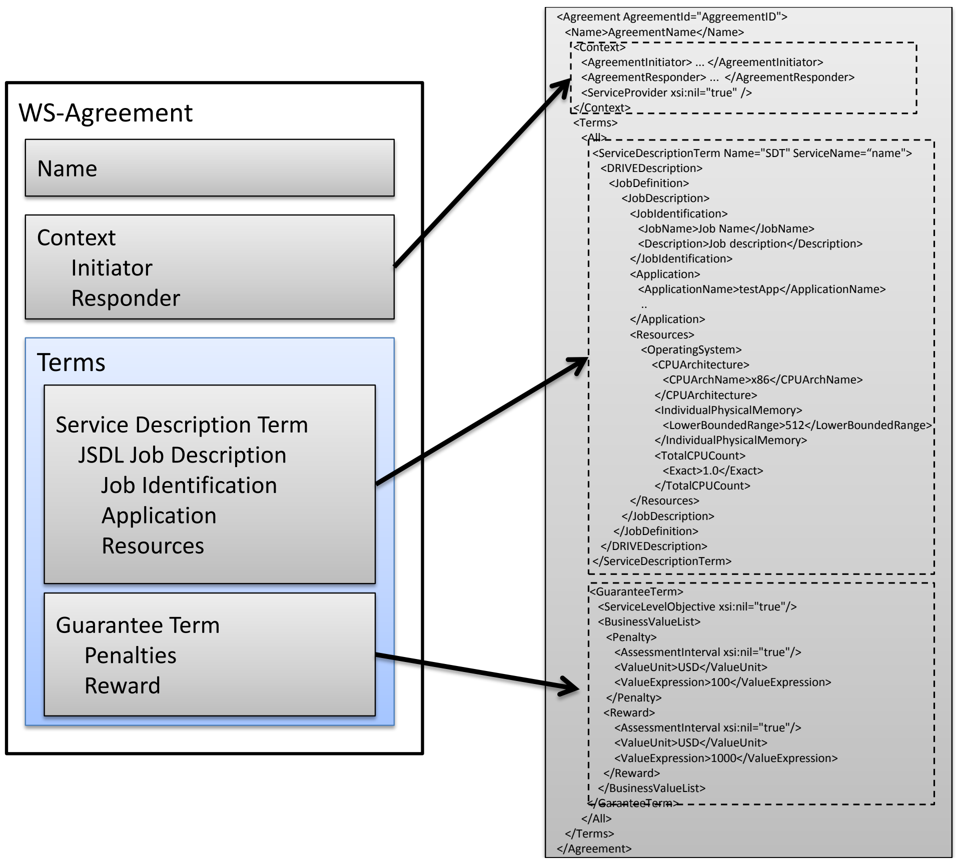


Figure 4.10: Two phase contract creation. The Allocation Manager creates a tentative (soft) agreement as the result of an allocation, the Contract Manager hardens the agreement into a binding contract.

Contract Representation

DRIVE Contracts contain contextual information relating to the participating parties, description terms defining what has been agreed upon, and guarantee terms outlining financial incentives (penalties/rewards). An example agreement is shown in Figure 4.11, for brevity some details have been left out. This agreement represents a Grid job expressed using JSDL as the term language. The service description term contains the (JSDL) task description which outlines the application and trivial resource requirements (CPU architecture, CPU Count, and Minimum Memory). The guarantee terms include a penalty ($100 US) if the agreement is violated and a reward ($1000 US) if the agreement is honoured - this reward is the negotiated price between consumer and provider. In this example there is a single guarantee term encapsulating all of the individual service levels, if a single resource requirement is not met then the contract is violated and the penalty term is invoked. Often users place more importance on particular aspects of the agreement (for example not receiving all the required CPUs may impact the provision more than the speed of CPU delivered), as such agreements may have multiple guarantee terms each relating to an individual SLO. Figure 4.11 presents a trivial DRIVE agreement, a full agreement produced by DRIVE is outlined in Appendix A.



```xml
<Agreement AgreementId="AggreementID">
  <Name>AgreementName</Name>
   <Context>
    <AgreementInitiator> ... </AgreementInitiator>
    <AgreementResponder> ...  </AgreementResponder>
    <ServiceProvider xsi:nil="true" />
   </Context>
   <Terms>
    <All>
     <ServiceDescriptionTerm Name="SDT" ServiceName="name">
      <DRIVEDescription>
       <JobDefinition>
         <JobDescription>
          <JobIdentification>
           <JobName>Job Name</JobName>
           <Description>Job description</Description>
          </JobIdentification>
          <Application>
           <ApplicationName>testApp</ApplicationName>
            ..
          </Application>
          <Resources>
           <OperatingSystem>
            <CPUArchitecture>
             <CPUArchName>x86</CPUArchName>
            </CPUArchitecture>
            <IndividualPhysicalMemory>
             <LowerBoundedRange>512</LowerBoundedRange>
            </IndividualPhysicalMemory>
            <TotalCPUCount>
             <Exact>1.0</Exact>
            </TotalCPUCount>
          </Resources>
         </JobDescription>
       </JobDefinition>
      </DRIVEDescription>
     </ServiceDescriptionTerm>
     <GuaranteeTerm>
      <ServiceLevelObjective xsi:nil="true"/>
      <BusinessValueList>
       <Penalty>
        <AssessmentInterval xsi:nil="true"/>
        <ValueUnit>USD</ValueUnit>
        <ValueExpression>100</ValueExpression>
       </Penalty>
       <Reward>
        <AssessmentInterval xsi:nil="true"/>
        <ValueUnit>USD</ValueUnit>
        <ValueExpression>1000</ValueExpression>
       </Reward>
      </BusinessValueList>
     </GaranteeTerm>
    </All>
  </Terms>
</Agreement>
```

Figure 4.11: DRIVE Contract using WS-Agreement.

4.5.4 DRIVE Contract Manager

The DRIVE Contract Manager is designed to abstract the complexities of the contract hardening process from users and providers. At the conclusion of an allocation the Contract Manager is responsible for negotiating contracts with the winning provider(s). The resulting contract is also stored following the WS-Agreement specification, terms of the contract are stored individually so that they can be monitored in the future. All participants are presented with copies of the resulting contract, they can also access the stored contract or subscribe for notifications of term state through the Contract Manager.

Figure 4.12 depicts the contract creation process. The Contract Manager iteratively negotiates a contract with each winning provider based on the tentative agreement created by the Allocation Manager. A provider may reject an agreement for any number of reasons (for example lack of capacity or consumer reputation). In this case the Contract Manager can (depending on the protocol) request second chance substitute providers from the Allocation Manager. These second chance providers are found by re-computing the negotiation without the defaulting provider.

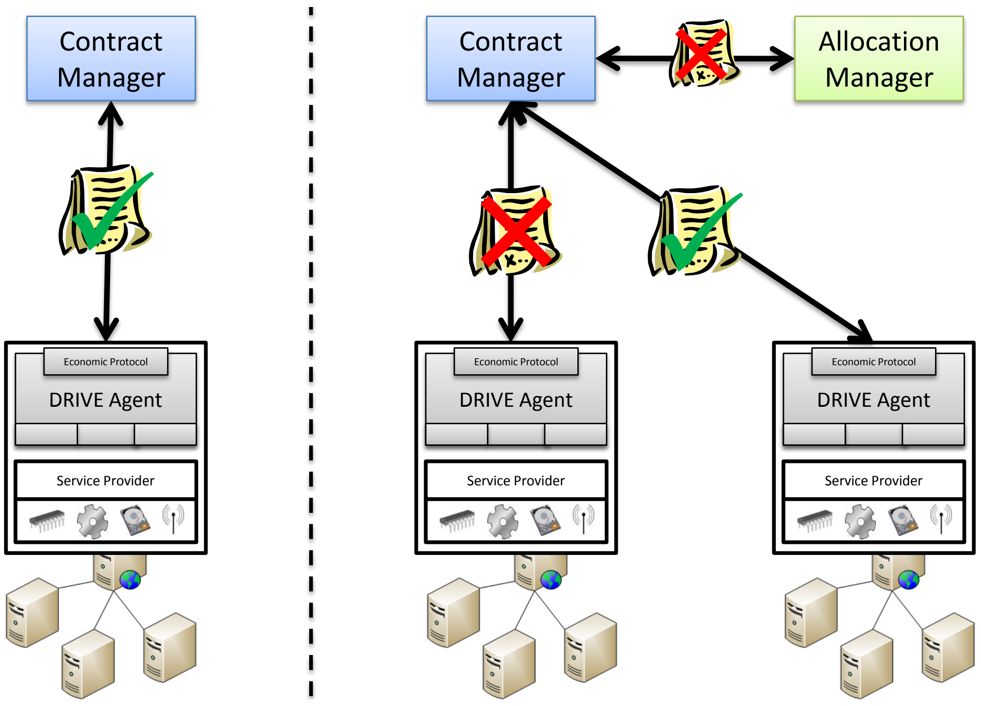


Figure 4.12: DRIVE Contract creation. The Contract Manager creates and confirms contracts with providers. In the case the provider rejects the agreement (right side) second chance substitute providers are used to confirm the contract.

The Contract Manager does not need to be deployed as a trusted entity in the VO as it has been designed to be verifiable rather than secure. This means that while a Contract Manager could potentially subvert the contract creation process, entities in the system would be able to detect malicious behaviour. All information specified in a contract is publicly available. Providers send signed messages when confirming or rejecting agreements, in addition the rejection reason is included in the signed message so that substitutes are not computed in the event a contract has been altered. Penalties and VO properties are declared in the task description (or defined statically) and all parties are aware of these properties when negotiating or joining the VO. The task description is available to all parties in the allocation and the winning results are also published and available to each entity in the VO.

DRIVE does not explicitly consider monitoring and enforcement of agreements as it is outside the scope of this project. However, the Contract Manager has been designed to be extensible such that in the future agreements can be monitored in a fine grained manner. Contracts are stored by the Contract Manager in such a way that individual guarantee terms can be monitored, to do this guarantee terms are represented separately from the description terms. Rather than enforcing contracts DRIVE relies on penalties and rewards invoked when a monitor determines if terms have been satisfied. General approaches to penalties include depositing default fees with a trusted third party in advance, exchanging fees directly as the result of an agreement, exclusion from future allocation, feedback (reputation based), or re-execution of the agreement. These mechanisms are also outside the scope of this thesis.

The Contract Manager is designed as a separate entity in the DRIVE architecture for a number of reasons. Most importantly the process of contract establishment (and monitoring/enforcement) is expensive, adding this functionality to the Allocation Manager would add additional computation and communication overhead. While Contract negotiation could be performed by clients and providers, the negotiation protocol is complicated and the Contract Manager abstracts these complexities. The Contract Manager also maintains contract state for the duration of the provision and in the future it is designed to be used for fine grained monitoring and enforcement of agreements.

4.5.5 Co-allocation

Co-allocation [201] is the process of simultaneously allocating resources in predetermined capacities over a group of resource providers. Co-allocation is often used in distributed computing to satisfy requirements for QoS, replication, and parallelism. Take for example, a large scale scientific application that has data sites worldwide, overall efficiency can be greatly improved by running an application in parallel using multiple processing resources close to the individual data sites. Co-allocation in DRIVE is defined as replicated task execution over a group of service providers. If task capabilities differ, for example having different resource requirements for different aspects of the task, multiple related tasks are submitted for allocation relying on advanced reservation to coordinate execution

Co-allocation is supported through the task description schema and by the allocation process. Co-allocation allocations in DRIVE are performed in the same way as conventional single allocations, but rather than a single contract, a number of contracts are created for each co-allocated task. Co-allocated tasks are listed in the allocation description which is taken into account during the negotiation phase and multiple winners are accepted for the allocation. Co-allocation may refer to more than just simply requiring several resources to provide fault tolerance it is possible a user may want x instances provided by the same provider, this type of parameter is included as a "count" parameter in the allocation description and subsequent contract. The task description includes co-allocation information to determine if one negotiation should be run (x instances on 1 provider) or multiple negotiations (x instances over y providers). Both of these cases are represented in the subsequent contract(s).

4.6 Virtual Organisation

DRIVE uses a Virtual Organisation (VO) model to represent the highly dynamic nature of distributed environments (federated and non federated). A VO is a dynamic grouping model in which a set of users, institutions and resources are defined based on a set of shared policies (security, sharing, resource management). The concept of a VO forms the basis of Grid computing. Foster [13] defines Grid computing as *"coordinated resource sharing and problem solving in dynamic, multi-institutional virtual organisations"*. The sharing described is highly regulated, with rules governing what is shared, with whom, and under what conditions. The communities of individuals and groups defined by these rules is termed a VO. Figure 4.13 shows the grouping of users, institutions and resources into two VOs, this diagram highlights the fact that VO membership is not mutually exclusive.

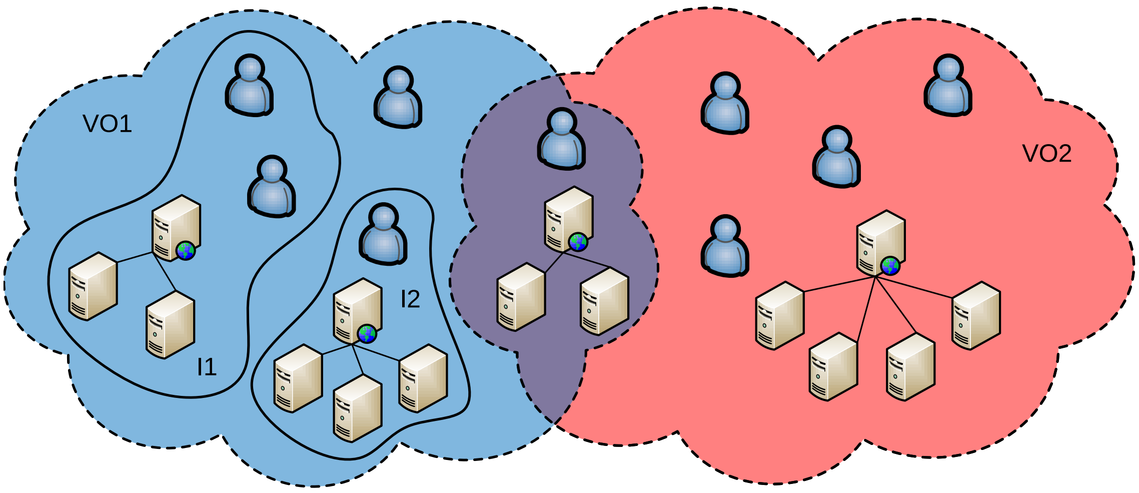


Figure 4.13: Two overlapping VOs (VO1, VO2) composed of institutions (I1, I2), individuals and resources.

Federated environments facilitate interaction between heterogeneous distributed resource providers, essentially adding another layer to the Grid VO model. However, the same VO requirements apply in this model to establish communities involving resources, individuals, institutions, and Grid/Cloud communities governed by rules defining sharing and access between members. Membership services are used to manage VOs by authenticating users and authorising actions according to VO defined policies. In large scale environments it is infeasible to maintain permissions for every individual to perform every possible action, rather VO groups and techniques for membership aggregation are used.

4.6.1 Virtual Organisation Management

The nature of a VO requires a specialised management infrastructure to control membership and, define and enforce sharing policies between members. There are two main tasks performed by VO management components, firstly, *registration* and *authentication* of users and providers to ensure their identity and VO membership, and secondly *authorisation* of users' actions based on previously defined sharing rules (policies). There are a number of VO membership systems used in Grid computing to model dynamic multi-organisation environments with heterogeneous resources and user groups. Many of these membership services are built upon existing Grid security architectures.

The Grid Security Infrastructure (GSI)[202] is the de facto standard Grid security architecture developed as part of the Globus Toolkit. Authorisation in GSI is based on user credentials (certificates) and provider access control lists (grid-map files). Authorisation decisions are made by mapping grid entities to local user accounts on specific resources. This approach lacks scalability and requires separate user accounts on hosting providers for each user or group represented. GSI does not model dynamic user groups well and it is difficult to implement fine grained authorisation policies. Several projects have been designed specifically to provide VO management on top of GSI, for example the Community Authorisation Service (CAS) [86], PRivilege Management and Authorisation (PRIMA) [87] and the Virtual Organisation Membership Service (VOMS)[88, 89].

CAS supports aggregation of VO policies (what VO members are allowed to do) and local policies (what a VO is allowed to do), by embedding signed SAML policy assertions in GSI proxy credentials. A CAS server maintains VO policies and a list of VO members. Users request capabilities using standard community credentials with restrictions (limited proxy credentials), the server then embeds signed policy assertions in the new credential relating to the VO. This allows administrators the ability to easily grant rights to a VO, and also allows community administrators to, in turn, grant subsets of these rights to individual members.

PRIMA provides fine grained privilege management at the user level, allowing users to hold and delegate access privileges to resources. XACML [144] privilege statements are embedded in GSI proxy credentials, these statements are then compared with XACML resource policies to determine access control at the resource provider.

VOMS is similar to CAS in that it embeds attribute certificates in GSI proxy credentials. The extended credentials specify user, group and VO membership which is used to determine access rights. Resource providers maintain access control lists relating users, groups, or VOs to authorise user actions. Unlike CAS, VOMS provides access rights directly and does not require mapping or interpretation by resource providers. Also, users authenticate with their own credentials rather than using limited group credentials making the architecture more flexible for individual users.

The design of DRIVE does not limit the implementation to a single VO management entity, this is due in part to the flexibility required in DRIVE to represent different communities (which may have existing VO management infrastructures). The core DRIVE architecture assumes the use of VOMS as it is the most widely used VO management architecture and has desirable properties such as extensibility, backward compatibility of certificates, and a comprehensive architecture defining groups and roles. However, CAS and PRIMA are similar enough in nature that they could be used without major modifications. The two DRIVE based implementations presented in Chapters 5 and 8 highlight this architectural flexibility in DRIVE by using VOMS and Facebook respectively to provide VO management.

4.7 Registration and Discovery

There are two distinct discovery mechanisms used in DRIVE, the first is discovery of DRIVE services to be used for a particular process (for example allocation), secondly Allocation Managers (or Components) discover suitable providers to initiate economic negotiation. Figure 4.14 shows a high level view of the information stored and the entities that access this information.

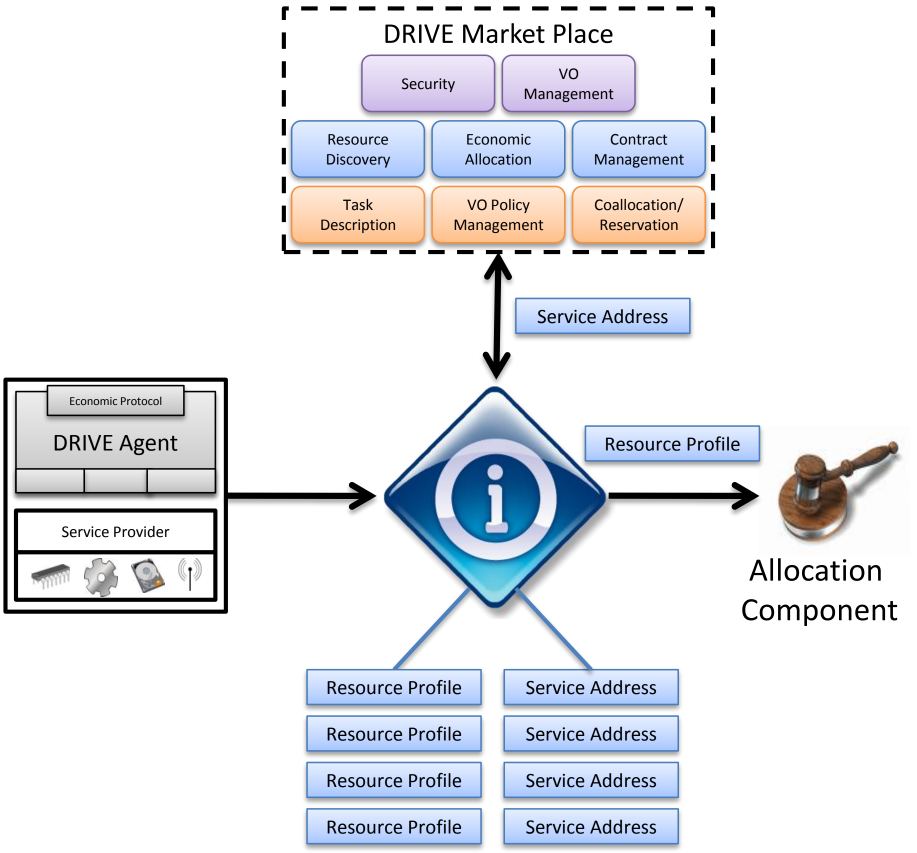


Figure 4.14: DRIVE Information Service. DRIVE Agents register resource profiles (metadata) describing their capabilities which is used by Allocation Components to target advertisements. Individual service addresses are also registered so that they can be discovered and used on demand.

Due to the distributed collaborative nature of DRIVE, service discovery is required by many of the meta-scheduling processes (invocation, allocation, contract creation). For example, clients locate Allocation Managers to facilitate the allocation process, Allocation Managers locate Allocation Components to conduct the allocation according to the chosen protocol and entities also discover Contract Managers to establish contracts as the result of an allocation. This type of service discovery is common in Grid environments and as such DRIVE utilises existing Grid service registration and discovery mechanisms.

In traditional Grid models resource (provider) discovery forms the basis for matchmaking. Providers advertise capacity by publishing resource information, discovery mechanisms are then used to match resource requirements with suitable provider capacity essentially acting as a global allocation mechanism. However, in DRIVE matchmaking occurs through economic negotiation where resource requirements are evaluated against provider capacity during independent valuation of a user request, allocation decisions are then made through the economic protocol. For this reason traditional discovery mechanisms are not crucial to the operation of DRIVE.

In DRIVE providers register metadata advertising their general capabilities in a *resource profile*. During allocation it is not vitally important to discover every provider in the VO, in fact, providers may discover current negotiations through reverse discovery mechanisms (Publishing Services). Discovery in DRIVE is therefore loss tolerant (or hint based) in that any number of satisfactory resources can be discovered to form a partial picture, reverse discovery complements this technique to provide absolute knowledge of the system. This approach has analogies to human economies, consumers may receive explicit sales offers through traditional targeted advertising, or they may discover the same sale by checking the store website or even going directly to the store.

In a Grid environment Grid Resource Information Services (GRIS) are used to register and discover information. The Globus Monitoring and Discovery System (MDS) is one such implementation providing an extensible hierarchical architecture from which information can be registered and retrieved. MDS exposes mechanisms by which XPath query statements are evaluated against registered metadata and filtered results are returned. DRIVE is designed to use the MDS Index Service to store and retrieve information in a structured manner. However this is not a necessity. Any Index Service model could be trivially substituted assuming it provides mechanisms to register and discover XML-based metadata.

The DRIVE metadata schema has been designed to provide information required by Allocation Managers. Included is a proof of concept resource profile describing the capabilities of a provider and categorisation of task types for low level resource providers. The resource profile has been designed for job based Grids however, it could be extended to other task domains. The Social Cloud implementation presented in Chapter 8 includes a modified resource profile describing the storage capabilities of a provider, this implementation demonstrates the versatility of the DRIVE registration architecture. In addition to the resource profile a modified form of the CaGrid metadata schema is used to provide detail about service providers, including information such as their location, contact details and administrators. This information is registered when a provider joins a DRIVE VO, DRIVE Agents are configured to periodically refresh this information to reflect current capacity.

4.7.1 Discovery Service Configuration

Unlike traditional Grid schedulers DRIVE does not rely on GRIS as the sole means of resource discovery. The burden of discovery and scheduling is in part shared amongst participants, using reverse discovery techniques. Never the less the discovery system plays an important role in targeting allocation advertisements at appropriate resources and discovering DRIVE services.

Early information service architectures, such as MDS-1 [48] used centralised services for resource discovery. In this model users and schedulers send information queries to a single centralised index service and resource providers constantly update the Index Service on resource usage. This approach limits scalability and the centralised service may become a bottleneck, it is also clearly a single point of failure. MDS-3 [49] allows information services to be organised in a hierarchy which improves the scalability problems seen in the centralised model. In a hierarchical model information flows in a stream like manner where individual services pass information upstream. Figure 4.15 shows a hierarchical discovery configuration in DRIVE. Service metadata propagates around the system between MDS services. Multiple information sources may register with individual Index Services. Users and other Index Services query directory services to discover resources and services. One can think of this system like DNS in which information can be found by progressively going up the tree. However this approach may still lead to scalability issues when attempting to discover large numbers of registered entities and it may be susceptible to failure if authoritative nodes fail. As the size of the Grid increases centralised and hierarchical models do not guarantee scalability and fault tolerance.

The most widely proposed scalable discovery model for Grid based systems uses distributed hash table (DHT) P2P overlay networks [203, 69, 204, 205]. P2P models have an extensive body of research and multiple implementations are available that can be used to create distributed, scalable, and fault tolerant discovery infrastructures. These models typically use decentralised P2P overlay networks to share resource information between VOs. Talia et al. [204] use a combination of a VO level P2P network and a VO level hierarchy of Index Services, thus combining the advantages of each method. Generally, in P2P discovery models, each node in the system stores information relating to one or more resources. In order to discover suitable resources a user sends a request to any node in the system, the node then returns matching resource descriptions if they are stored locally, or forwards the request to another node which in turn does the same. While these systems have a number of positive aspects including distribution, scalability, autonomy, and fault tolerance there are however limitations with P2P based approaches including efficiency, security, trust, and reputation [69].

The advantage of the MDS approach is each piece of information is stored in XML resources facilitating standardised access and filtering based on XPath. A new distributed MDS service could be created by leverage existing work on P2P based WSRF containers [206], in which resources can be discovered using the P2P network. This is in effect the same representation of information as in MDS and provides similar resource queries following the proposed P2P protocol. The DRIVE components have been designed with MDS as the discovery architecture due to its widespread

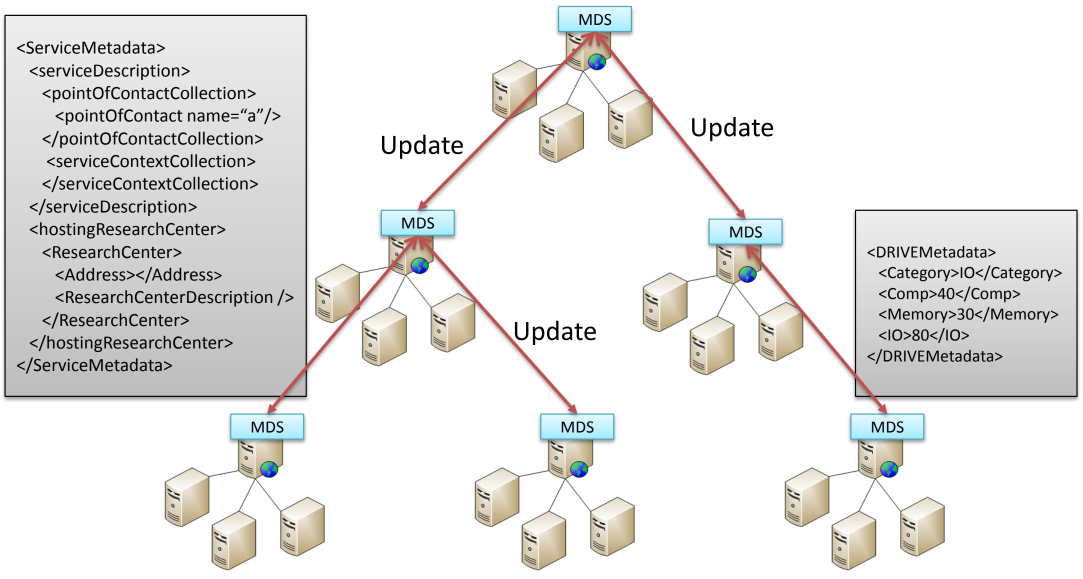


Figure 4.15: DRIVE Discovery Topology. DRIVE Metadata and Service Metadata propagates around the hierarchical MDS topology.

use in Grid environments and standardised nature. However the components that perform discovery are easily adaptable to other discovery models assuming the nodes in the network expose similar interfaces (registration and discovery) to that of MDS.

4.7.2 Reverse Discovery and Publishing

The use of a hint based discovery mechanism helps to improve auction efficiency at the expense of absolute knowledge. Reverse discovery mechanisms are used in DRIVE to complement traditional provider discovery and ensure all entities have the opportunity to participate in resource negotiation. Allocation advertising in DRIVE is only targeted at suitable providers based on registered profiles as it is not feasible for Allocation Managers to individually notify all members of a VO, instead, allocations are also published in VO wide Publishing Services to allow other providers the chance to discover ongoing negotiations. These discovery mechanisms are illustrated in Figure 4.16, two providers are notified by explicit advertising (left) and two providers discover the allocation through a Publishing Service (right).

The DRIVE Publishing Service is essentially a bulletin board for the VO, allowing Allocation Managers the ability to share market information with members without explicitly notifying all entities. Publishing Services describe current negotiations and also allocation results to share market conditions throughout the VO. The data stored in the Publishing Service could be registered

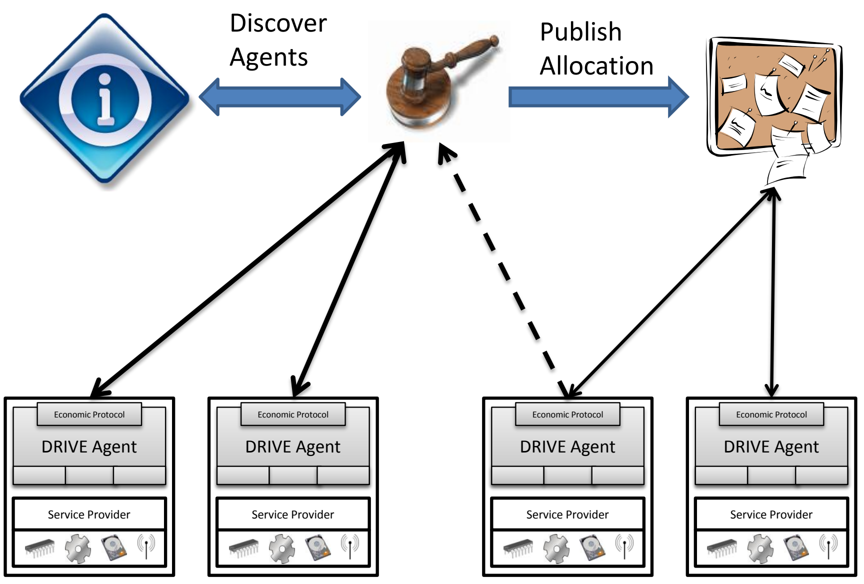


Figure 4.16: DRIVE Reverse Discovery. The Allocation Manager discovers suitable providers from the information service and then advertises the allocation directly. The providers on the right discover allocations through a Publishing Service and may choose to participate in the allocation.

in the MDS Index service alongside provider metadata, however the use of a separate service allows fine grained control over publishing, extensibility in published data, and it can be implemented as a simple obligation service without requiring communication, aggregation or complex topologies. In addition if publishing to MDS was required, the resources stored in the publishing services could be trivially configured to register with MDS.

4.8 Summary

This chapter has presented the DRIVE architecture, including detailed descriptions of the core DRIVE components and the protocols used to facilitate discovery, allocation, and SLA creation in a distributed market. The architecture is based on a co-op model in which a group of distributed DRIVE services provide decentralised meta-scheduling functionality. These services can potentially be hosted on participating providers through the use of secure allocation protocols. The architecture is designed to create an open market through the use of arbitrary economic protocols. Auction protocols in particular are well suited to distributed resource allocation as they can dynamically and efficiently establish market prices. In an untrusted environment secure auction protocols can be used to establish trust and privacy in the allocations. To facilitate the requirements of secure protocols DRIVE includes a plug-in protocol interface and a generic Auction

Component. DRIVE uses a novel two phase contract creation architecture to formally describe participant obligations. An incentive model is used to encourage participants to honour established contracts. One of the major goals of the DRIVE architecture is providing independence from a particular provider type, task model, and economic protocol. This independence has influenced many of the design decisions made, for example the plug-in protocol model, generic task description and term language, and separation from a particular execution model. To verify the architecture a prototype implementation of DRIVE used to create a federated Grid marketplace is presented in the next chapter.

Chapter 5

DRIVE Implementation

This chapter describes the prototype implementation of the DRIVE architecture. The prototype facilitates a federated Grid marketplace representing multiple resource providers. In the market reverse auction protocols [30] are used for economic resource allocation. The prototype includes functional implementations of the major components in the DRIVE architecture including the Auction Manager (Allocation Manager), Auction Component (Allocation Component), Contract Manager, Reservation Service, Publishing Service, and Bidding Agent (DRIVE Agent). Each component is implemented in Java as a Globus Toolkit 4 WSRF Web service. The prototype itself does not include banking functionality, however a prototype banking service is described within the context of the DRIVE based Social Cloud presented in Chapter 8.

Figure 5.1 shows the prototype implementation of DRIVE. The important auction components are the Auction Manger, Auction Context and Auction Component which collectively conduct reverse resource auctions. The DRIVE Agents in the prototype are called Bidding Agents responsible for valuing resource requests according to local policy and producing bids adhering to the appropriate protocol specification. Contract Managers abstract the DRIVE contract architecture by hardening agreements on behalf of users. Standard Grid VO management (VOMS) and discovery (MDS) mechanisms are also used in the prototype. The services depicted are defined according to the trust model defined in Section 5.1.

In the prototype consumers submit jobs to an Auction Manager which hosts a competitive reverse auction. Providers may choose to bid for the right to host a consumer job. The DRIVE prototype supports three distributed auction protocols, a standard sealed-bid first-price auction, a sealed-bid second-price (Vickery) auction and a privacy preserving, verifiable Garbled Circuit [207, 130]. A comparison between these protocols in DRIVE is presented in Chapter 6. Jobs are described using the Job Submission Description Language (JSDL) [71]. Globus GRAM [46, 47] is used to execute jobs on heterogeneous resource providers. Contracts are described using WS-Agreement with JSDL used as the term language. Following the DRIVE architecture a modified WS-agreement protocol is used due to the initial agreement created through auctioning. A proof of concept bidding policy framework is implemented using the eXtensible Access Control

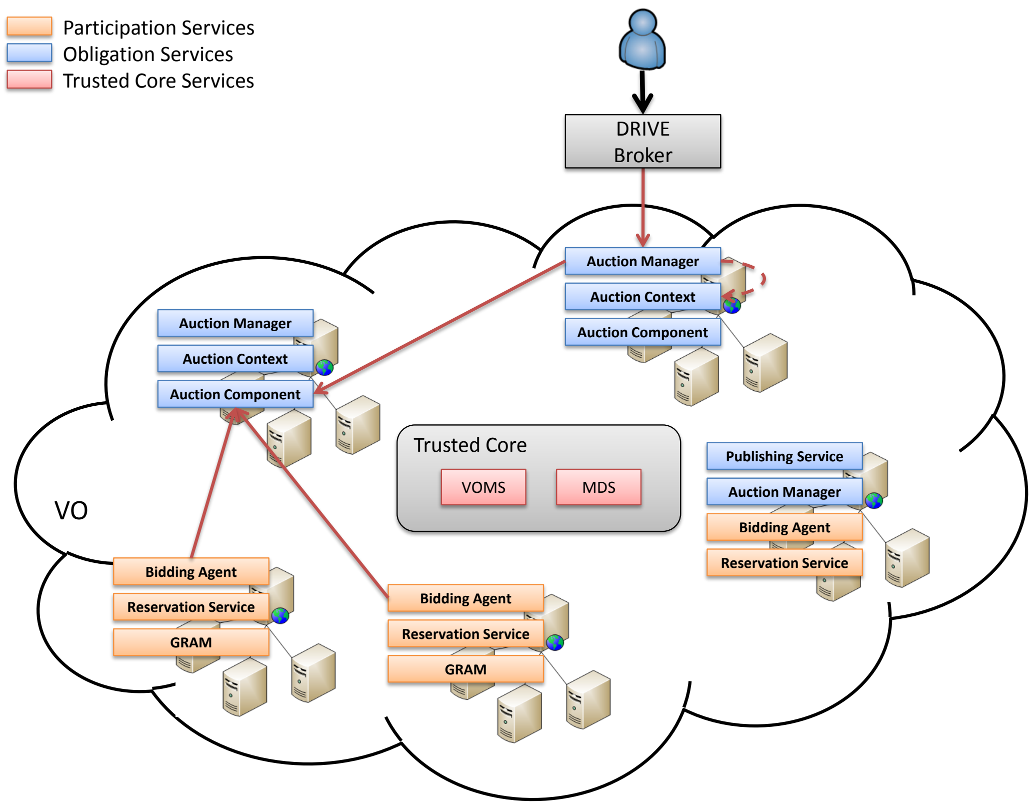


Figure 5.1: The prototype DRIVE architecture. In this diagram there are five service providers that are members of a DRIVE VO. Clients submit jobs through a client side broker to an Auction Manager for resource allocation. An Auction Context is used to store auction state and an Auction Component is used to conduct the auction following the specific protocol. Bidding Agents act on behalf of service providers to compute valuations and submits bids.

Markup Language (XACML) [144].

The remainder of this chapter presents the DRIVE prototype implementation outlining the major components used and describing interactions between components. The chapter begins with an overview of the prototype trust model and the services in each layer of the model. The structure of this chapter reflects the architecture presented in Chapter 4. In particular focus is given to allocation, bidding, valuation, advanced reservation, job execution, contract management, VO management, security, registration, and discovery. The chapter concludes with a discussion of potential high utilisation strategies that can be used to increase occupancy and utilisation in auction-based systems. Due to the Grid focus of this prototype service providers are in this case resource providers, the two terms are therefore used interchangeably.

5.1 DRIVE Trust Model

The services in the DRIVE prototype have vastly different trust requirements. To represent these requirements the core services are categorised according to the layered trust model shown in Figure 5.2. Participation services form the lowest layer of the model as the services are controlled by, and represent, the hosting provider. Obligation services "belong" to the DRIVE meta-scheduler, however as they are hosted on potentially untrusted providers they are prone to subversion and therefore form the second layer in the trust model. Finally trusted core services require trusted environments in which to run to ensure integrity and availability. This section presents an overview of the core prototype DRIVE services according to the trust model.

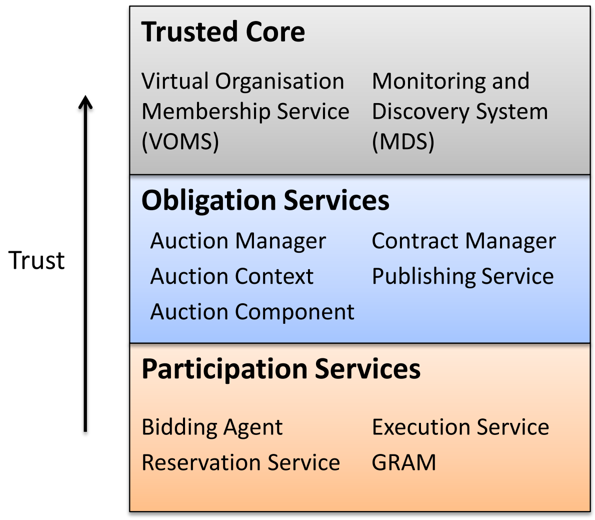


Figure 5.2: DRIVE Trust Model.

5.1.1 Participation Services

Participation services are required in order to participate in the DRIVE VO. These services are not contributed to the meta-scheduler rather they provide an interface with which core DRIVE services interact with providers. There is no need for these services to be trusted externally as they are hosted and controlled locally by resource providers. The most important participation services are used for bidding, reservations, and execution of user tasks.

The Bidding Agent is the prototype instantiation of the DRIVE Agent designed to represent a resource provider in economic negotiation. The Bidding Agent exposes interfaces for receiving advertisements, discovering auctions, bidding and creating contracts. Bidding Agents respond to auction events by computing bids for tasks they wish to host using valuation functions, local policies, and consideration of future commitments. They are also responsible for following the chosen protocol and encoding bids in the required bidding language.

The Reservation Service contains an indexed schedule that provides a detailed picture of future commitments made by the provider. It stores any potential tasks the provider has negotiated for, including auctions that have been bid on, tentative agreements, and hardened contracts. This service is the key component for supporting flexible advanced reservations in DRIVE.

Execution of tasks in a job based Grid is difficult as resource providers may utilise differing LRMs with individual description languages and protocols. Globus GRAM provides transparency over heterogeneous resource managers allowing users to submit, monitor and control jobs using a standard interface and language. The prototype implementation of DRIVE makes use of GRAM as it is one of the most widely used execution components available and is part of the de facto standard Globus toolkit. It also provides extensive functionality and has been shown to provide high throughput execution, scalability, and availability [47]. To interact with DRIVE, the prototype includes a DRIVE Execution service that wraps GRAM, this allows contract enforcement and mapping between job descriptions.

5.1.2 Obligation Services

Obligation services collectively form the core of the DRIVE meta-scheduler providing resource allocation, contract creation, and general management. Individual services communicate with one another to provide these functions, following the selected protocols and adhering to local and VO policy. In particular the Auction Manager, Auction Context, Auction Component, Contract Manager and Publishing Service are implemented as obligation services in the DRIVE prototype.

The central component of the allocation mechanism is the Auction Manager, it is the first point of contact by external clients and is responsible for managing the auction process. The Auction Manager stores individual auction state in context resources and utilises an Auction Context service to manage these resources. In order to create an auction instance a client starts an auction by passing the task description to the Auction Manager which then creates the appropriate auction resources and selects the services required for the particular auction protocol. The Auction Manager advertises auctions by querying appropriate bidders, creates protocol specific representations of the auction, monitors the auction, terminates the auction (time, number of bids), determines winners, and notifies parties of results.

The prototype auction protocol plug-in architecture is designed to support the implementation and use of arbitrary protocols without prior knowledge of the specific services required to implement a particular protocol. However, secure auction protocols may require distributed instances of protocol specific services in order to maintain the validity and security of the protocol. Auction Components form the basis of the plug-in protocol architecture allowing run-time configuration of services to behave as required by arbitrary protocols. Effectively an Auction Component is a shell that wraps a protocol specific service, it uses a message based system to communicate through a typed Web service interface. The Auction Component takes different forms depending on the protocol, for example in a Secure Generalised Vickery Auction (SGVA) the Auction Component is an Auction Evaluator used to compute shares of the winner of the auction. In the Garbled Circuit protocol the Auction Component is an Auction Issuer used to garble the circuit and is a requirement of the VPOT protocol[207].

The Contract Manager is designed to abstract contract creation, monitoring and enforcement. At the completion of an auction participants contact a Contract Manager to confirm agreements with winning providers and create binding contracts. The Contract Manager iteratively confirms agreements and consults the Auction Manager for second chance substitute providers in the case providers reject agreements.

Publishing Services are used to publish auction advertisements and auction results. Publishing Services aid reverse auction discovery allowing bidders the opportunity to discover published auctions that were not directly advertised to them. Auction results are also published to share market information with bidders and provide a public record of results, which may be monitored by any entity in the system and used for confirm results. The information stored is not authoritative and is only used to sample market conditions. Authoritative results may be obtained from Auction Managers.

5.1.3 Trusted Core Services

In the DRIVE prototype some services cannot be hosted on participating resources due to the nature of the service. There are a number of reasons why this may be the case, for example services may require data privacy, data integrity, certificate chains for authorisation, a trust anchor, or they may be directly responsible for security. Trusted core services also include services that require dedicated hardware to ensure they are always available as they are essential components of the system without which the meta-scheduler would not function. In the DRIVE prototype MDS and VOMS are hosted in the trusted core. Both services must be operational at all times and rely on a trust anchor to ensure correct functionality. VOMS also requires data privacy as it manages authorisation information.

Globus MDS is a standard information system used in Globus Grid environments. In DRIVE, resource providers register service addresses, metadata, and resource profiles when joining the VO. The service addresses must be discoverable (or published) so the services can be used to provide the functionality of the meta-scheduler. While it is not a necessity to implement MDS as a trusted core component, functionality would be severely limited if the discovery services were unavailable or corrupted. Reverse discovery mechanisms could still be used for negotiations and published service addresses could be used to discover DRIVE components. However this approach limits scalability and will reduce the dynamic nature of the system. In the prototype only authoritative MDS services are required to be in the trusted core, much like DNS, nodes further downstream could be less secure as discovery can be performed at the top of the tree. Using alternate implementations of discovery systems could also reduce the requirement for these services to be trusted, for example a P2P configuration [69] where peers are responsible for completing information following discovery protocols.

VOMS is a database-based virtual organisation membership service which has mechanisms to manage authorisation information within multi-institutional collaborations. Users join a VO by contacting the VOMS service to obtain VO credentials and resources check local access control lists to determine if these credentials are valid. VOMS forms the basis for authentication and authorisation in the DRIVE prototype and as such it must be a trusted component.

5.2 Resource Allocation

The DRIVE prototype includes three reverse auction protocol implementations that can be used simultaneously in the market. In these protocols providers bid (or tender) for the rights to host a user submitted job, the lowest price "wins" the job. Reverse auction protocols are commonly used in Grid systems as they model the submission process. In a reverse auction model the task of resource valuation is left to providers, which provides a degree of economic transparency to consumers. Providers essentially compete to maximise local utilisation and profit. In DRIVE resource providers are represented by a Bidding Agent that values requests and submits bids according to the chosen protocol. Consumers interactions with the DRIVE market are also transparent through the use of a specialised DRIVE client broker.

5.2.1 Auction Sequence Diagram

Figure 5.3 outlines the auction based allocation process in DRIVE. Auctions are conducted using a subset of the contributed obligation services. The process of auctioning is transparent to the user. Before starting an auction the client or broker must authenticate with VOMS, a working proxy is created and subsequent access to DRIVE services is provided via this proxy. The broker selects a viable Auction Manager by discovering available Auction Managers from the MDS Index Service. The choice of Auction Manager is left up to the user and can be based on any user defined requirements, for example protocol support, reputation (previous interactions), location, or price (if users are charged for service use). Having selected a suitable Auction Manager all auction correspondence is performed through this service. An auction request is submitted to the chosen Auction Manager to start the auction process. The job is described in the appropriate description language (JSDL) and any other auction properties are also specified at this point, such as auction duration, number of bidders, and protocol specific properties (for example encryption size). The Auction Manager creates an instance of the auction setting up stateful resources to maintain auction state through the Auction Context. Depending on the auction protocol a range of Auction Components will be instantiated and used to perform protocol specific computation, for example for the Garbled Circuit protocol an Auction Issuer is instantiated, for the homomorphic and polynomial auction protocols a group of Auction Evaluators will be used.

The Auction Manager advertises the auction and solicits bids from providers. In order to target interested bidders the Auction Manager retrieves a list of suitable providers from MDS,

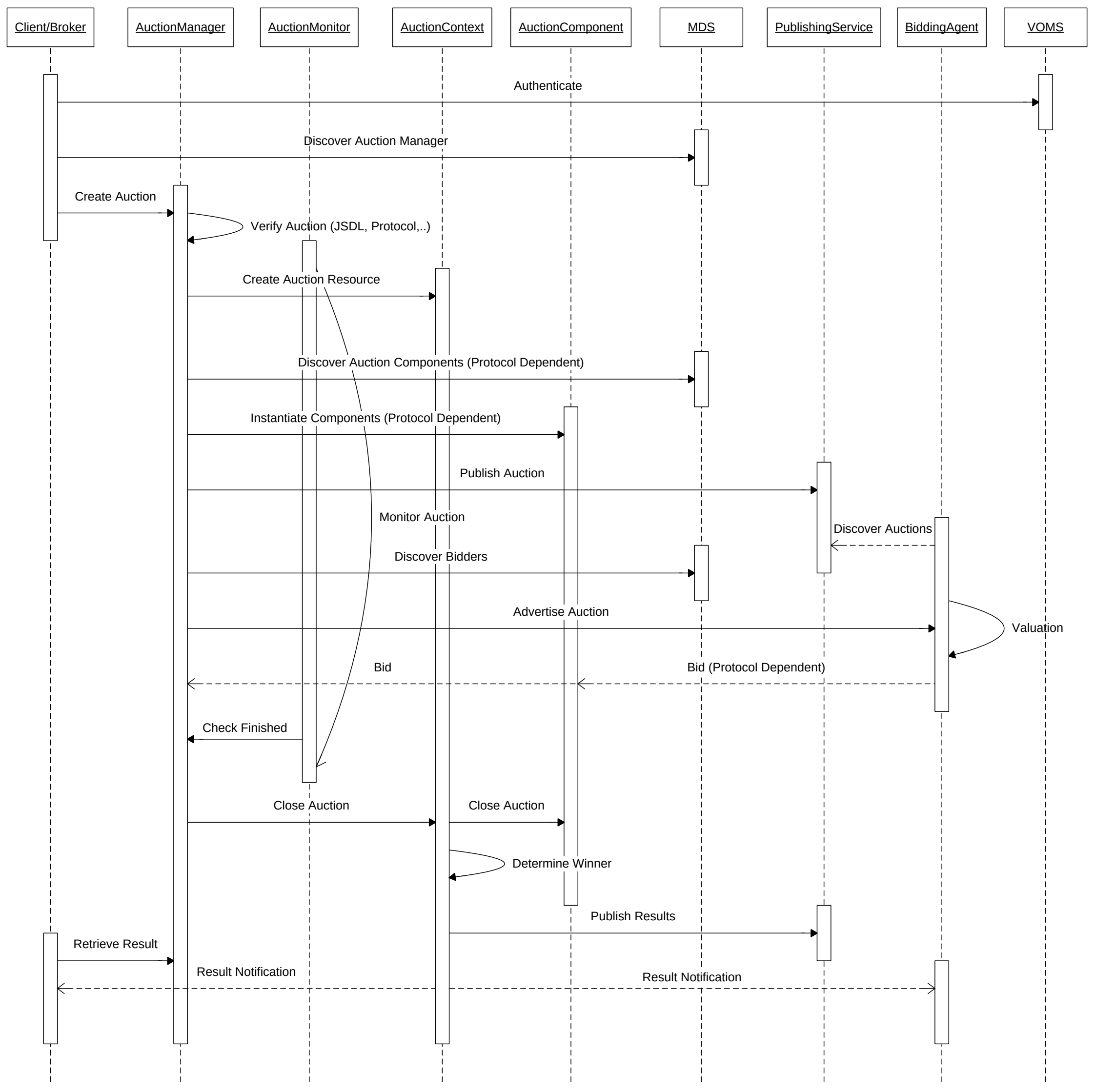


Figure 5.3: Auction based resource allocation in DRIVE.

filtered based on specific job requirements. These selected resource providers are then notified of the auction through “push” advertisements. The auction is also published to defined Publishing Services for reverse discovery.

Individual Bidding Agents are invoked by Auction Manager (or Auction Component depending on the protocol) advertisements, or through reverse discovery using a Publishing Service. The Bidding Agent computes a bid and contacts the Auction Manager (or Auction Component) to submit the bid.

Through the duration of the auction the Auction Monitor manages auction constraints and closes the auction when end conditions are met. When the auction is complete a winner is determined and a tentative agreement is created, in some protocols a list of alternative resource providers is maintained in case the agreement cannot be honoured. The Auction Manager and Auction Context notify participants (either explicitly or through subscribed WS-Notifications) and publish results at specified Publishing Services. Participating resource providers record the successful allocation and subsequent tentative agreement through their Reservation Service.

The remainder of this section provides detailed descriptions of the components and protocols used in the auction-based allocation process.

5.2.2 Auction Request

Due to the complexity of DRIVE and the associated auction protocols there are a number of configurable properties which may be specified when starting an auction, for example particular auction services, auction protocols, protocol properties, auction options, auction components to be used in the allocation process, and the job description to name just a few. DRIVE wraps each of these configurable properties in an Auction Request XML document, the schema for which is outlined in Appendix C (Listing C.1).

DRIVE allows consumers (or providers) to select DRIVE services for all aspects of the allocation including the Auction Manager, Contract Manager, Publishing Services and Auction Components (for example Evaluators for the SGVA protocol). This choice is important for commercial entities as they may select components based on prior experience, trust, cost, or location. In a way selection is be based on brand recognition as users may select components which have favourable reputation.

When starting an auction the consumer may specify the type of auction protocol to be used, associated protocol specific properties, and generic properties of the auction. For example, in a homomorphic SGVA auction protocol specific parameters include the encryption strength (bits) and the number of goods represented. Auction options are independent of the chosen protocol and relate to management aspects of the auction, for example in the prototype auction options are used to determine the end of an auction by restricting the auction duration or limiting the number of participants.

The design of DRIVE enforces task independence, therefore in the prototype the job description can be specified in any task dependent language. For this reason attributes such as the reservation job window, coallocation parameters, and economic parameters (such as reserve price) are included in the auction request to ensure they are independent from the task representation. Economic (and auction) parameters are separated from the job description to ensure flexibility and provide extensibility. This differs from approaches taken in other systems (for example SORMA) where job description languages are extended to encode economic parameters and reservation windows. The DRIVE prototype does not follow this approach as it is inflexible and binds the system to a single description language. In the job based DRIVE prototype the job description is a serialised JSDL document describing the requirements of the job.

5.2.3 Job Description

Job description languages are designed to express the requirements of a job in a well defined manner such that the clients submitting jobs and the providers hosting jobs are aware of specific requirements. In the wider Grid community there are many different job representation languages used, this is in part due to the lack of Grid standardisation and the fact most resource managers use proprietary languages. Of the common Grid-like systems; Condor uses ClassAds [208], Globus uses RSL [73] in Pre-WS GRAM and GRAM Job Description [209] in GT4, Job Description Language (JDL) is used by Enabling Grids for e-Science (EGEE) Workload Management [210], and Abstract Job Objects (AJO) are used in Uniform Interface to Computing Resources (UNICORE5) [211]. The lack of a standardised job description language limits scalability as users must understand the description language used by each resource provider, scheduler and broker they use. Some effort has been made developing tools and translators to map differing representations, however these mechanisms are by no way complete and mapping between languages is often lossy.

A standard description language facilitates interoperability amongst schedulers, resource managers, and brokers alike. It also simplifies mappings as proprietary languages can be mapped to a single common language rather than producing mappings for every permutation of existing languages. The Job Description Markup Language (JDML) [212] was an initial attempt at standardisation. JDML was developed for the EU DataGrid project and based on Condor ClassAds. In JDML job requirements are expressed using XML based attribute value pairs. WS-JDML [213] provides a web service based implementation acting on JDML documents. The Job Submission Definition Language (JSDL) [71] is an official Global Grid Forum recommendation for standardising job description and utilises some of the concepts developed in JDML. As JSDL has become a defined Open Grid Forum standard multiple Grid schedulers have added support including GridSAM, Gridway, GRIA, GridBus, gLite, and UNICORE6 [214]. Globus has also developed a prototype GRAM service supporting JSDL however as yet this has not been released.

JSDL is an XML based language designed primarily for Grid computing and as such it is able to describe submission requirements for individual jobs. Three categories of requirements are expressed in JSDL: job identification requirements, resource requirements, and data requirements. JSDL is designed to be extensible, allowing extension by third-party developers to add additional elements. JSDL was chosen as the submission language for the DRIVE prototype due to its flexibility, standardisation, extensibility and its focus on Grid job representation. To demonstrate the flexibility and extensibility of JSDL it has been extended for use in the DRIVE based Cloud presented in Chapter 8.

5.2.4 Auction Services

The three main services used in DRIVE to facilitate the auction process are shown in Figure 5.4. The Auction Manager is the central component which creates new auctions, oversees the bidding process, monitors the status of auctions and determines the result. An Auction Context service is associated with a single Auction Manager and is responsible for storing general allocation and protocol state (auction description, options, protocol properties, bids). Auction Components are protocol specific entities which implement the required protocol functionality using a plug-in interface and a message passing interface for protocol specific communication.

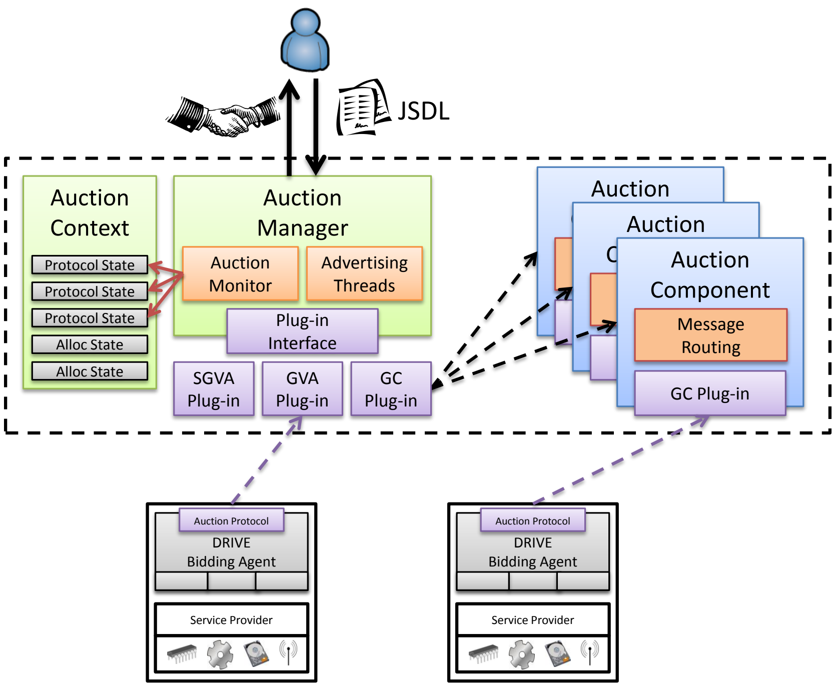


Figure 5.4: DRIVE Auction Services.

Auction Manager

The Auction Manager instantiates an auction from a client request. The auction request is parsed and the job description is validated according to the given schema (in this case JSDL). Assuming the chosen auction protocol is supported by the Auction Manager an auction resource is created in the associated Auction Context. The appropriate plug-in protocol is then dynamically loaded and executed to conduct the auction. The Auction Manager has an associated advertisement thread pool which can be used by the auction protocol to advertise the auction to bidders. Suitable bidders are found by composing an MDS query for providers which meet certain requirements related to the job description. The Auction Manager exposes bid submission methods to facilitate bidding, result retrieval methods for discovering the outcome of auctions, and substitute computation methods for re-executing auctions. Most of these methods are mapped to the context service referencing the appropriate auction resource (state). The Auction Manager also maintains various state to monitor auctions (both running and completed auctions) and uses a threaded auction monitor to actively ensure auction conditions are met. The auction monitor ensures running auctions are following specified auction options and properties, in particular it monitors conditions to ensure the auction finalises when its end conditions are met.

Auction Context

Following the factory pattern the Auction Context creates and stores auction resources to represent auction state. The context resource stores all information relating to the auction, exposing much of the lifecycle of an auction. At all points in the auction lifecycle state is represented in a *ResourceProperty*, accessible to entities within the system. Auction State follows the schema shown in Appendix C (Listing C.2). The Auction Monitor updates state throughout the auction process. Depending on the protocol the context resource may collect bids, and determine the winning bidder and price to be paid according to the selected protocol. In distributed protocols, Auction Components instantiated for the auction are able to access shared information in the context resource to compute the winner. Finalising the auction is generally invoked by the auction monitor, however it may also be triggered by an auction event such as a bid. Upon winner determination the results are published to any Publishing Services specified by any party, optionally a contract negotiation mechanism can also be started at this point if requested by the client. In the case a bidder cannot honour their agreement, the context service (in collaboration with Auction Components) offers the ability to compute second chance substitute providers. Depending on the protocol used this process will generally be conducted using the state stored in the context resource to re-execute the protocol plug-in. For the remainder of this chapter the Auction Context is considered to be part of the Auction Manager due to the 1-1 mapping between them.

Auction Component

The Auction Component provides a service based wrapper for protocol specific code. Auction services instantiate one or more Auction Components as a given protocol specific entity before an auction. The Auction Component dynamically loads a registered plug-in protocol library and creates a stateful representation of the instance. The protocol instance is stored in a local Auction Component resource and communication occurs using protocol specific messages. The message passing interface provides generic communication through the typed Auction Component interface. A protocol or instance id is attached to each message, this facilitates routing to the appropriate protocol plug-in. The plug-in protocol is expected to parse the message and perform the appropriate action. A response can be sent in the form of a synchronous message returned through the Auction Component service. In most cases however, this process is asynchronous and the plug-in protocol will make additional calls to other entities within the system. The protocol plug-in also stores a reference to a stateful resource controlled by the Auction Context service. This resource allows the plug-in to store and retrieve protocol specific state using a stack based approach. The advantage of this approach is state may be persisted such that the protocol will not lose information between invocations or if the container is restarted. The interface implemented by all protocol plug-ins defines the required instantiation, deletion, message passing, state management, and resource access methods. The Auction Component serves to provide protocol independence and extensibility allowing protocol developers the opportunity to create new distributed auction protocols without worrying about the communication interface or distributed services required.

5.2.5 Broker

The DRIVE prototype includes a Java broker to provide transparent user interaction with the DRIVE architecture. The broker is designed to abstract the complexities of the Web service based architecture providing a Java API and command line interface for performing common actions such as submitting a task, creating and monitoring an auction, creating and retrieving a contract, and submitting a task for execution. The broker is also configured to handle WS-notifications generated from the Auction Manager and Contract Manager.

Every DRIVE service has a client API capable of invoking each method while also encoding resource identification in WS-addressing headers and using credential management to provide transparent secure access to the (stateful) service. The broker uses these APIs to provide end to end support allowing a client to invoke the broker using default, or individually customised, properties. The broker can encode requirements in a JSDL job description, initiate an auction, retrieve the result, create a contract, and submit the job at the specified time without requiring any explicit user interaction. Currently the broker does not include credit management, due to the fact banking and book balancing is not included in the prototype.

The broker supports batch submission and dependant task submission over a period of time using a multi threaded architecture for submitting and monitoring multiple auctions simultaneously. Credential management is abstracted by loading standard Grid credentials (with possible VOMS extensions) for service invocation. Experiments on the DRIVE architecture utilise the multi threaded broker architecture to submit large scale Grid traces containing thousands of jobs.

5.2.6 Auction Protocols

The DRIVE prototype includes implementations of three auction protocols: a single good sealed-bid first price auction protocol, a single good sealed-bid second price Vickery auction protocol, and a distributed secure combinatorial Garbled Circuit auction protocol. Three protocols were implemented in order to satisfy the framework development "rule of three" [215]. The rule of three states that three protocols should be implemented in a proposed framework to ensure that all the common components required for the domain have been considered.

These different protocol implementations also serve to demonstrate the flexibility of the DRIVE architecture, allowing protocols to be seamlessly selected based upon the scenario. The protocol implementations differ greatly in terms of security, complexity, and overhead and are therefore most suited to different scenarios. The sealed-bid first price and Vickery protocols are simple and efficient protocols requiring only a single auctioneer and capable of representing only a single good. The Garbled Circuits protocol is an example of a more complex secure distributed protocol requiring two auction parties that together conduct the auction. This implementation is capable of hosting combinatorial auctions representing multiple goods.

By default Auction Managers are configured to use the Vickery protocol as it is efficient and has several advantages such as truthful bidding and bid privacy (discussed in Chapter 4). Users may request the use of different protocols in the auction request. Auction Managers maintain a list of protocols supported and mappings to the appropriate library for dynamic class loading. The two sealed bid protocols do not require instantiation of any additional Auction Components as all process can be performed by a single auctioneer. Auction descriptions are communicated to bidders using the initial JSDL job description, bidders then bid using sealed (encoded) string representations. In the prototype public keys can be retrieved through the service, in the future this information could be added to registered metadata or distributed offline.

The Garbled Circuit protocol uses a single Auction Component instantiated as an Auction Issuer, this component is responsible for creating the circuit and producing the garbled mapping table. Communication between entities uses serialised *GateTables* and *MappingTables*. Bidding Agents also implement Garbled Circuit functionality, computing values for each of the bundles, according to the properties of the protocol (goods, bits in price, and bundles). The bids are encoded in a three dimensional array representing bids on each of the combinations. This bid object is serialised and submitted to the Auction Manager. The Auction Manager Garbled Circuit plug-in iterates through the bids retrieving the garbled input for each bid from the Auction Issuer and determines the winner(s) of the auction.

In the prototype the mapping process from JSDL to combinatorial representation in the Garbled Circuits protocol is a simple proof of concept implementation. JSDL resource parameters are mapped arbitrarily to form different combinations in the Garbled Circuits representation. This approach is suitable for evaluating the performance of the protocol, however, further analysis is required to determine the best way to represent combinatorial options in JSDL and develop appropriate mapping techniques from JSDL to protocol specific combinatorial representations.

5.2.7 Job Profiling

While matchmaking is not the basis for allocation in DRIVE, providers and consumers must still be able to quantify the requirements of a job to negotiate allocation. The inability of clients to accurately determine resource requirements is commonly cited as one of the biggest difficulties facing Grid systems [216]. Poor resource estimation has been shown to be common in Grid environments, for example analysis of a production European Grid workload showed differences of up to two orders of magnitude between requested memory capacity and actual memory usage [217]. Accurate resource estimation in an economic utility environment like DRIVE is even more important as users pay for the resources provisioned based on the estimated requirements of the job. Poor estimations lead to overutilisation or underutilisation of the provider, which in turn will result in overpayment by consumers or penalties imposed on consumers respectively. Poor resource estimation has been suggested as one of the major motivating factors for resource overbooking [200].

Most Grid and Cloud architectures implicitly assume resource requirements and execution times are known in advance or take the approach that resource prediction is a task for clients. In a commercial environment this approach makes sense as clients pay for usage and generally they are in the best position to estimate the requirements of jobs through sampling, application models, or past history of similar jobs. In addition the alternative approach of provider estimation is difficult and can only be applied in simple environments. Predicting job run time in distributed environments is more difficult as providers are heterogeneous by nature. In Cloud computing users provision VMs which essentially provide fixed pre-defined resource capabilities, it is then up to the consumer to run jobs within the scope of the VM.

The DRIVE prototype takes a similar approach to job profiling as other Grid systems. Consumers are expected to estimate and quantify job requirements in defined units and then encode them in the job description (JSDL). Execution time per-se is not considered due to the heterogeneity of providers, however reservation windows combined with defined units are used to provision resources with particular deadlines. To reduce the burden on consumers tools and algorithms are required to estimate resource requirements.

Estimation Techniques

There is little research related to predicting resource requirements and execution times of jobs in parallel and distributed environments. Most existing research in this area has focused on creating job run time prediction models [218], genetic algorithms [219], and machine learning algorithms [220]. Sonmez et al. [221] present analysis of various parallel estimation techniques on Grid workload traces, their results show that using available information (site, user, application, job size) and classifications over time, accurate job duration predictions can be made by providers. One of the most well known and earliest estimation projects is the Network Weather Service [222] which uses sensors deployed to Grid resources and networks to create a model of current utilisation. A "forecasting" process predicts future utilisation by applying a set of forecast models to a time series of these measurements. The focus of this work is on load balancing rather than estimation of individual jobs, however data obtained from the NWS could be used to develop job profiles.

EMPEROR [58] is a meta-scheduler designed to analyse scheduling algorithms and estimation models. EMPEROR includes several predictor models to predict the requirements of a task. For example individual resource load and memory utilisation are monitored and models are dynamically created based on statistical characteristics of usage sequences. A prediction model is used to calculate an estimated run time on loaded resources using reference times based on unloaded hosts [102]. The results shown in [58] show highly accurate execution time and memory usage estimations using both stationary and non stationary traces on unloaded, lightly loaded and heavy loaded resources.

Another way to determine job requirements is through job (or application) benchmarking. This involves executing the job (or part of a job) in a sandboxed environment where resource requirements can be determined. One example is Parallel Assessment Window System (PAWS) [223] which is designed to benchmark applications for parallel supercomputers by generating a set of metrics relating to the performance of an application. Interestingly these metrics are developed relative to a particular host environment and may therefore be suitable in a federated Grid/Cloud model.

Resource profiling and prediction is currently an open research problem. Further work is required to apply these profiling approaches to obtain accurate resource requirement estimations. The most promising approaches are based on historical data which can be maintained by clients and used to estimate requirements. Tooling could be used to obtain benchmark information about jobs when there is no historical data. Job samples or samples could also be used to tune resource requirements. Profiling is not within the scope of this thesis and is therefore not considered in the DRIVE prototype, however the approaches presented in this section could be applied by consumers and providers to estimate requirements in a DRIVE market.

5.3 Resource Provider Components

In the DRIVE prototype resource providers bid for the right to host a given job using a plug-in reverse auction protocol. Each provider or group of providers is represented by a DRIVE Bidding Agent which is responsible for valuation and bidding on behalf of the provider(s).

5.3.1 Bidding Agent

The DRIVE Bidding Agent is implemented as a single service which advertises capacity, values resources, bids in auctions, and negotiates contracts. Figure 5.5 shows the major components in the Bidding Agent. The blue components (Bidding Thread, Publish Metadata, Advertisement Monitor) signify threaded entities which are always running. The dashed blue box indicates a thread pool of bidding threads. The core Bidding Agent relies on a plug-in architecture to load auction protocols, system utilisation libraries, policies and pricing functions. The plug-in architecture provides extensibility and customisability along with simplifying administration.

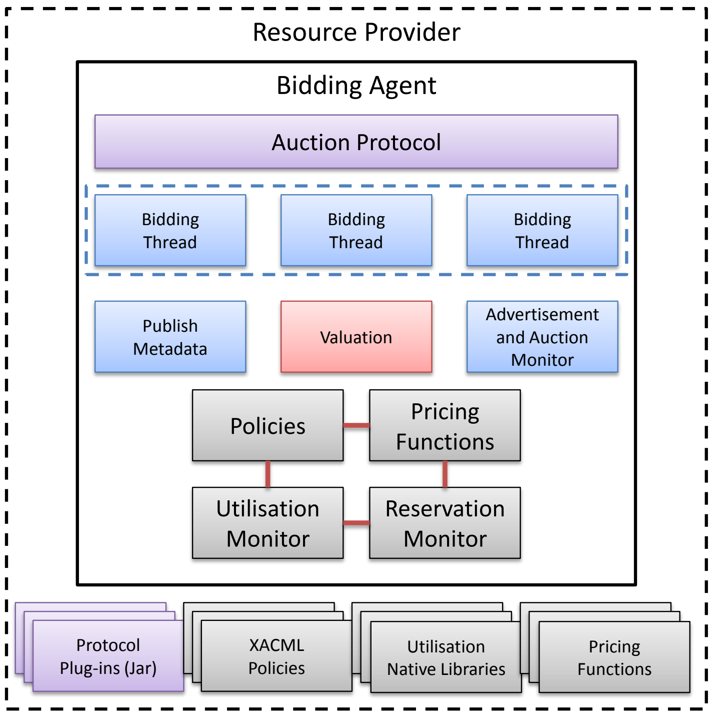


Figure 5.5: DRIVE Bidding Agent.

Any number of auction protocols may be supported simultaneously by a Bidding Agent, each protocol supported is defined in the Bidding Agent properties file and is dynamically loaded when required through the plug-in interface. Plug-in protocols must follow the Bidding Protocol interface. Protocol developers distribute jar files to bidders, who in turn store the plug-in locally and specify the path in properties files. In the future protocols could be retrieved from an online repository. The plug-in architecture makes use of Java's dynamic class loader.

The Utilisation Monitor is a plug-in component that allows host dependent resource monitoring. The prototype loads a monitoring jar file at run time which discovers the host architecture and dynamically loads the appropriate C library to monitor system resources. Interactions between the specific system monitor and the Java monitor use the Java Native Interface (JNI). In the prototype two C utilisation monitor plug-ins are provided one for Windows and the other for Fedora Core as these are the architectures used in the experimental testbed. Each plug-in monitors real time system information (CPU, memory, I/O) periodically or when requested.

The Reservation Monitor, provides an interface to the local Reservation Service which is used to query and update resource commitments. The valuation component computes bid values for a given resource by consulting current utilisation, projected utilisation, local policies, bidding strategies, and using the appropriate pricing functions. Valuation processes and bidding policies are covered in more detail in the following sections.

As described in Section 5.5.2 each resource provider periodically publishes a resource profile (metadata) describing the current capacity of the provider and the type of jobs they are interested in hosting. The metadata component of the Bidding Agent probes the utilisation monitor and reservation monitor to obtain current and projected utilisation, this data is used to alter the registered resource profile and periodically update the information stored in the Index Service. The Advertisement Monitor responds to registered Publishing Service events in an attempt to discover auction advertisements that are not explicitly advertised to the given provider, upon discovery the same bid valuation process occurs to determine if the bidder participates in the auction.

A push/pull model is used to start the bidding process as the Bidding Agent may be invoked by an auction advertisement or respond to an auction description discovered through the Advertisement Monitor. In both cases, the auction description is passed to a Bidding Agent request queue, bidding threads in the thread pool monitor the queue and process individual tasks when an event is triggered. The thread pool makes the bidding process asynchronous as there may be various delays in the bidding process. Without a threaded implementation bid requests would tie up the service until the bid has been finalised and submitted.

Having retrieved an auction advertisement from the queue the bidding thread ensures the auction protocol is supported, it then begins the valuation process. Valuation determines a value for the resource(s) (this may happen multiple times in a single combinatorial auction for each set of goods) consulting current and projected capacity and using local policies to determine the associated risk to the provider. Pricing functions determine the final bid value to be submitted. The Bidding Thread calls the appropriate protocol plug-in to submit the bid to the advertised Auction Manager or Auction Component. The final bid is submitted using an appropriate representation called a bidding language, generally this is protocol specific. For example in the polynomial SGVA protocol bids are encoded in the degree of the polynomial, in the Garbled Circuits protocol bids are encoded in a Boolean circuit protocols.

5.3.2 Policies

The prototype Bidding Agent includes a proof of concept implementation of the DRIVE policy architecture. To verify the design three policies representing risk, reputation, and utilisation have been implemented. The reputation and utilisation policies are used to establish individual risk for aspects of a negotiation from a bidders perspective. The risk policy aggregates individual risk values to determine the overall risk factor. In a production environment many other policies need to be considered, for example predicted load, job types, and overbooking ratio. The prototype only includes provider policies, however client polices could also be defined to incorporate client side risk when determining a reserve price, VO level policies may also be used to define membership and security requirements. An example Bidding Agent reputation policy used in the DRIVE prototype is presented in Appendix B

In DRIVE policies are stored in a repository co-located with the Bidding Agent. Administrators are responsible for defining the CML based policies and listing the paths in the Bidding Agent properties file. To simplify policy creation a Java application has been developed to generate sample policies based on user defined values and associated actions. The program creates a series of policies mapping values to one of three risk levels (low, medium, or high). The Bidding Agent has a generic bid computation component that includes a PEP to enforce risk decisions, customisable PDPs are used to evaluate reputation and utilisation values against the relevant policies and determine a risk value. The risk policy is then applied to the calculated risk value to determine the overall risk. In the prototype reputation values are generated randomly to test the implementation, projected utilisation is calculated based on the current capacity (from the utilisation monitor) and the requirements of the job. As projected utilisation increases (up to the capacity of the provider) the risk increases, similarly the risk value increases as reputation values decrease – entities with poor reputations are more risky to interact with.

The DRIVE policy architecture is extensible in that administrators can define additional XACML policies without altering any code. Generic PDPs are used to determine the level of risk based on the defined policies which are loaded at run time. Administrators can also customise the decision making process by implementing their own PDPs.

5.3.3 Valuation

Bidding Agents value resource requests based on the calculated risk value and local pricing functions. The Bidding Agent supports user defined plug-in pricing functions to determine a bid price based on a job request. The plug-in interface simply passes through the auction advertisement (including the job description) and generated risk assessment, the plug-in library then calculates and returns a bid value. At present the plug-in pricing interface is implemented for non-combinatorial auctions, however this could be extended to return a combinatorial bid for each bundle. Application of risk in combinatorial values would need to be applied to each value in the combinatorial bid which may be difficult depending on the representation.

The DRIVE prototype includes several standard pricing functions that use both linear and proportional models to calculate a base price for a resource or set of resources. These models include price decay functions based on time, available capacity, market conditions, and previous auction results. The models are explained and evaluated in Chapter 6.

Bidding Example

The following equations demonstrate bid calculation in DRIVE. For the purpose of this example reputation is used to determine the risk of participating in the auction. Reputation values are assumed to be generated based on previous interactions with entities in the system. While reputation is outside the scope of this thesis it presents a good demonstration of how risk can be incorporated in a bid. For the purpose of the prototype reputation values are randomly generated for each interaction. It is assumed in a production environment every entity involved in the auction would have an associated reputation value (stored locally or obtained from a reputation service).

An individual risk value is first calculated for each entity involved in the auction based on their respective reputation value (5.1). A total risk value for reputation is computed as the average risk associated with each entity, an exponential decay function is applied to model the relevance of newer information (5.2). In this function the decay quantity (λ) is a user defined parameter that models how quickly information becomes obsolete, and t is the time at which the value is calculated. The overall risk factor is then calculated by determining a weighted average of all risk values (for example utilisation, reputation) according to pre-defined user weightings (5.3). Local risk polices are used to determine if the overall risk factor is suitable to participate in the auction. The risk factor directly corresponds to the valuation multiplier which increases the valuation based on the assessed risk value and the perceived importance of risk by the provider (5.4). DRIVE allows any number of risk categories to be defined and participation can be cancelled if any single piece of information exceeds defined levels. A linear valuation of the requested resources is obtained by applying pricing functions to each resource or set of resources (5.5). The pricing function may value resources individually or collectively. The final bid value is then adjusted to incorporate risk in the cost model by applying the valuation multiplier (5.6).

$$ReputationRisk_i = \begin{cases} 1 & : Reputation_i > 80 \\ 2 & : Reputation_i \leq 80 \\ 3 & : Reputation_i \leq 40 \end{cases} \tag{5.1}$$

$$ReputationRisk = \frac{\sum_{i=0}^{n} (Reputation_i) e^{-\lambda t}}{n} \tag{5.2}$$

$$RiskFactor = \frac{\sum_{i=0}^{n} Risk_i * RiskWeighting_i}{n} \tag{5.3}$$

$$ValuationMultiplier = RiskFactor * RiskImportance \tag{5.4}$$

$$TrueValuation = \sum_{i=0}^{n} PricingFunction_i * Resource_i \tag{5.5}$$

$$BidValue = TrueValuation * ValuationMultiplier \tag{5.6}$$

5.3.4 Advanced Reservation

The DRIVE Reservation Service records and manages provider commitments. Bidding Agents interact with the Reservation Service to store and retrieve bids, tentative agreements, and contracts. In addition the reservation service supports implementation of user defined scheduling algorithms to optimise task execution. Reservation requests in the prototype are submitted in the auction request using a predefined reservation schema as shown in Appendix C (Listing C.3). A proprietary schema is used to provide independence from the task description used. This generic reservation schema allows all reservation requests to be standardised across task domains. DRIVE implements a flexible reservation window allowing reservations to be specified with a start time, end time and duration which provides flexibility when the task is actually run.

Reservation Service

The Reservation Service is implemented as two WSRF Web services. Figure 5.6 shows the high level architecture of the Reservation Service. The main service exposes a general interface to add and manipulate reservations, the secondary Reservation Context Service implements the factory pattern to create and manage reservation resources. The main service supports creation, removal, modification, and scheduling of reservations. Additionally it is able to retrieve a list of reservations for specific periods of time and print comprehensive schedules to files for auditing.

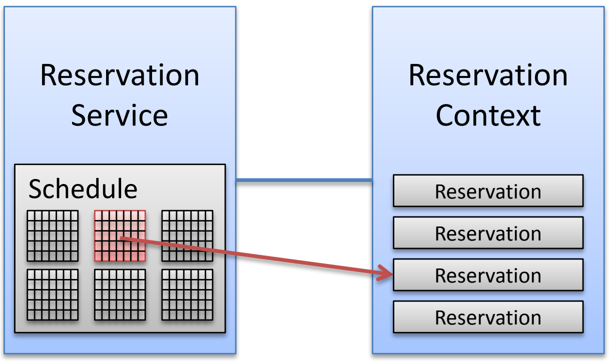


Figure 5.6: Reservation Architecture.

A major goal of the Reservation Service is to provide efficient retrieval of reservations, to accomplish this the Reservation Service maintains a schedule mapping index. The schedule indexes reservations according to reservation time using a calendar format to quickly retrieve reservations for a given time period. To reduce storage cost and retrieval time, tasks are stored according to their granularity in the largest available time slot. Reservations are identified by a unique ID with respect to the service. Upon creation of a reservation all further correspondence is conducted using the reservation ID to uniquely identify the particular reservation.

The use of WSRF resources allows clients to query the state of a reservation via a number of mechanisms including polling and WS notifications. All reservations are persistent so that if the container fails reservations are recovered when the container is restarted. There is a one time cost of iterating through all reservation objects and recreating the reservation schedule, but this is acceptable considering the performance gained by indexing according to time.

Reservations are stored as XML resources following the DRIVE reservation schema shown in Appendix C (Listing C.4). Each reservation is uniquely identified in the service by its ReservationID. The reservation type defines the current stage of negotiation and the job window defines the reservation period. As the service could potentially represent multiple resource providers resource identification and resource addresses are stored with the reservation. The task description and any tentative agreements or hardened contracts are also stored depending on the stage of negotiation. Finally contact points for the Auction Manager and Contract Manager used are stored as the provider may require this information, for example to withdraw from an auction.

5.3.5 Job Execution

The DRIVE prototype focuses on resource allocation rather than job execution. However to complete the job lifecycle Globus GRAM is used to execute jobs on heterogeneous hosts. A DRIVE specific Execution Service has been implemented to wrap GRAM due to the different job representations used and the requirement for interactions between the Execution Service and DRIVE components (bidding, reservations, contracts). The design of DRIVE implicitly assumes service providers (execution components) interact with DRIVE services to ensure clients have permission to invoke the service or execute tasks. This is done by verifying contracts and/or reservations. While this approach is appropriate when developing DRIVE-enabled services (or using gRAVI to create services), altering existing services like GRAM is not feasible.

DRIVE Execution Service

The DRIVE/GRAM Execution Service is designed to check the DRIVE reservation or contract, convert the JSDL job description to a GT4 job description, and submit the job to GRAM using the GramJob client API. At the conclusion of an auction a contract is established to specify the job requirements and reservation details. The client is expected to submit the task to the DRIVE Execution Service at the specified time. The user is pre-authenticated to ensure their identity and the Execution Service is responsible for checking reservations to ensure the contract exists. Assuming the contract is valid, resources are provisioned using GRAM to submit the job to a physical resource. At this point if there is not sufficient capacity available the Execution Service may choose to selectively violate another contract depending on local policies.

The Execution Service implements a mapping process to parse the DRIVE prototype JSDL description and create a GT4 GRAM job description. For example the JSDL element *IndividualPhysicalMemory* is mapped to GT4 GRAM job *MaxMemory* and *MinMemory* for the upper and lower range respectively. In the prototype only the job executable and system resources are mapped to the GT4 job description. Data staging is not supported as this is not a focus of the DRIVE prototype. Due to the limited number of JSDL resources expressed in the DRIVE prototype the mapping process from JSDL to GRAM is somewhat trivial. Extending the mapping functionality is outside the scope of this research, however a complete mapping could be created by porting any one of the available scripts and applications that map JSDL to RSL/GT4 XML, for example jsdlproc included with AstroGrid-D[1].

The GramJob API is a Java API for submitting jobs to a GT4 GRAM service allowing support for sequential and batch job submission, credential delegation, job monitoring, and result retrieval. Having created a GRAM job description and set the appropriate job attributes (id, duration, termination time) the Execution Service submits the job to the local GRAM service using the GramJob API. Due to the flexibility of the GramJob API a single Execution Service can represent and submit tasks to a number of GRAM services, however in the DRIVE prototype there is a one to one mapping between Execution Service and GRAM service.

5.3.6 Service Generation with gRAVI

The DRIVE architecture assumes a degree of separation between the traded service and DRIVE middleware layer. This separation presents a substantial barrier preventing exploitation of DRIVE markets by application providers as custom Bidding Agents and Execution Services must be deployed. To overcome this limitation gRAVI has been developed to automatically generate services with integrated DRIVE capabilities, therefore supporting transparent participation in DRIVE markets. This integration removes the need to create an additional middleware layer such as the Execution Service used in the DRIVE prototype and provides tight coupling between the service and the DRIVE components.

gRAVI (Grid Remote Application Virtualisation Interface) is a Web and Grid service wrapping toolkit which can be used to generate DRIVE-enabled services for any application. gRAVI makes the DRIVE infrastructure more readily available as users unfamiliar with DRIVE or computational economies can rapidly create and "sell" a service in a DRIVE market. The generated gRAVI service includes all DRIVE Bidding Agent functionality and interfaces, exposes plug-in protocol and valuation interfaces, and provides the ability to define bidding policies.

Using gRAVI developers are guided through a series of dialog windows to customise their service, including options for DRIVE-enablement, Grid scheduling support, HTML/Ajax Web interface, and file staging. gRAVI creates a fully independent WSRF service containing the application (or application path), all required libraries, configurable properties files, and deployment scripts for several different containers. In order to participate in a DRIVE market the service must be deployed and configured. Deployment is automated through gRAVI. DRIVE customisation requires implementing one or more DRIVE valuation plug-ins (the same as the Bidding Agent) which are responsible for parsing requests and determining a price. When an advertisement is received the plug-in valuation code is called to determine a bid. Metadata can be specified statically

[1]`http://www.gac-grid.org/project-products/Software/wg5-software-jsdlproc.html`

in the properties file or the user can alter the service to dynamically update metadata according to current conditions in the service code.

Chapter 7 describes the gRAVI toolkit in detail.

5.4 Contract Management

The DRIVE prototype includes a prototy[e implementation of the DRIVE contract architecture, including the Contract Manager and functional WS-Agreement based contracts. The Contract Manager establishes contracts representing the requirements and obligations of consumer and provider, and stores the contract for the duration of the provision.

5.4.1 WS-Agreement Contracts

The Contract Manager creates agreements conforming to the WS-Agreement schema. Listing 5.1 highlights some of the important elements included in a DRIVE contract. The context outlines the agreement participants, in DRIVE participants are identified through their address and unique ID assigned when joining the VO. The service provider describes which of the entities is providing the service (the default is AgreementResponder). Service Description Terms (SDTs) describe the functionality to be delivered by the agreement, in the prototype implementation the SDT is the JSDL job description. Service properties specify the domain specific measurable aspects associated with the agreement, in the case of DRIVE these are relative metrics from the job description. The location element references a specific field in the SDT irrespective of the domain. In this agreement an XPath expression maps individual disk space from the JSDL job description in the SDT to a service property variable.

The guarantee terms match up with each of the defined service property variables. Guarantee terms outline agreed upon levels of service between the consumer and the provider in the form of Service Level Objectives (SLOs). In this example agreement the obligated party is the service provider (Agreement Responder). Key Performance Indicator (KPI) targets express a SLO by defining a target level for a given key, in this case specific bounds are placed on the amount of storage provisioned (ranging between 10-100 bytes). The Business Value List outlines the business importance associated with the SLO - it is this mechanism that defines the rewards and penalties based on the outcome of the agreement. In this case the importance is arbitrarily defined as 1, the importance can act as a quantifier to interpret the proportion of the payment or penalty related to a particular guarantee term (for instance 0.5 would mean half the reward is paid if this term is satisfied). The default behaviour of the DRIVE prototype is to divide the defined reward and penalty of the job equally across the guarantee terms. However this mechanism could be extended to partition the reward or penalty unevenly across a group of guarantee terms depending on the relative importance.

A full WS-Agreement contract generated by DRIVE for the service based Social Cloud is shown in Appendix A, this agreement differs in that it uses EJSDL to describe the task, economic properties, and requirements of a storage Web service, however the structure and much of the content is similar to the job based prototype presented in this chapter.

Listing 5.1: Example DRIVE WS-Agreement.

```
<Context>
  <AgreementInitiator>Consumer</AgreementInitiator>
  <AgreementResponder>Provider</AgreementResponder>
  <ServiceProvider>AgreementResponder</ServiceProvider>
</Context>

<ServiceDescriptionTerm Name="JobDescription" ServiceName="ServiceName">
  <DRIVEDescription xmlns="">
    <jsdl:JobDefinition></jsdl:JobDefinition>
  </DRIVEDescription>
</ServiceDescriptionTerm>

<ServiceProperties ServiceName="ServiceName">
  <Variable Name="IndividualDiskSpace" Metric="jsdl:IndividualDiskSpace">
    <Location>/JobDescription/Resources/IndividualDiskSpace</Location>
  </Variable>
</ServiceProperties>

<GuaranteeTerm Name="IndividualDiskSpace" Obligated="ServiceProvider">
  <ServiceScope ServiceName="jobId"/>
    <ServiceLevelObjective>
      <KPITarget>
        <KPIName>jsdl:IndividualDiskSpace</KPIName>
        <CustomServiceLevel xsi:type="jsdl:RangeValue_Type">
          <jsdl:UpperBoundedRange>100.0</jsdl:UpperBoundedRange>
          <jsdl:LowerBoundedRange>10.0</jsdl:LowerBoundedRange>
        </CustomServiceLevel>
      </KPITarget>
    </ServiceLevelObjective>
  <BusinessValueList>
    <Importance>1</Importance>
      <Penalty>
        <AssessmentInterval xsi:nil="true"></AssessmentInterval>
        <ValueUnit>USD</ValueUnit>
        <ValueExpression>10</ValueExpression>
      </Penalty>
      <Reward>
        <AssessmentInterval xsi:nil="true"></AssessmentInterval>
        <ValueUnit>USD</ValueUnit>
        <ValueExpression>100</ValueExpression>
      </Reward>
  </BusinessValueList>
</GuaranteeTerm>
```

5.4.2 Incentives (Rewards/Penalties)

Due to the economic focus of the DRIVE prototype several remunerative financial incentives have been implemented. Positive incentives are inherent in a DRIVE economy as providers are rewarded by consumers for providing the agreed upon levels of service. Negative incentives (or penalties) have been implemented in the prototype to evaluate their effect on allocation. In particular constant and dynamic penalties are defined. In the constant model penalties are statically defined irrespective of the "size" or value of the failed task or the impact of the violation. Dynamic penalties attempt to reflect the value of the breach by determining the value of the job. For example penalties are defined as a percentage of the job value, metrics can also be used to determine a penalty based on job requirements. In the case of overbidding, penalties are calculated based on the difference between the original defaulting price and the substitute provider price to cover any losses experienced by the consumer. These penalty functions are explained and evaluated in Chapter 6.

5.4.3 Contract Manager

The Contract Manager is composed of two services, one acts as a general interface to create and manage contracts, while the other (context service) manages contract resources. When a contract request is made the Contract Manager discovers the auction result, creates a contract resource, and places the result on a confirmation queue. The Contract Manager contains a thread pool of confirmation threads, which respond to contract events on the confirmation queue to create and confirm contracts with one or more providers. If a contract is rejected some protocols allow recomputation of the auction (second chance substitute), in this case the Contract Manager obtains a substitute from the Auction Manager and attempts to re-confirm the agreement with the new winner. If there are multiple rejections, substitute requests are batched so as to limit the number of requests to the Auction Manager.

Contract resources store the contract for the duration of the provision. Specific security policies can be implemented allowing only certain entities access to the contract details. Guarantee terms are stored in resource properties to provide fine grained management and in the future monitoring of individual terms. The WS-Agreement specification also outlines a state model to represent the state of an agreement (or individual guarantee terms), this is not included in the current prototype as monitoring has not been implemented.

Establishing a Contract

Figure 5.7 shows the interactions between services in DRIVE when creating a contract. At the completion of an auction participating entities (clients and providers) are notified of the result, the same results are also published to Publishing Services. Any of the participating entities can initiate the contract creation process at any point after the auction concludes. The Contract Manager begins by obtaining the authoritative auction result from the Auction Manager, this result is used to create one or more contracts containing the job description, participating parties, rewards and penalties specified. A contract resource is created to represent the contract for the duration of the provision. The Contract Manager then iteratively attempts to confirm contracts with each of the winning bidders. Bidding Agents are expected to ensure they can meet contractual obligations and confirm the agreement, if they cannot they will reject the contract and a penalty is likely to be enforced. Assuming all Bidding Agents confirm the contract(s) all parties are notified of the completed contract(s) and the client is able to submit the job to the Execution Service at the specified time.

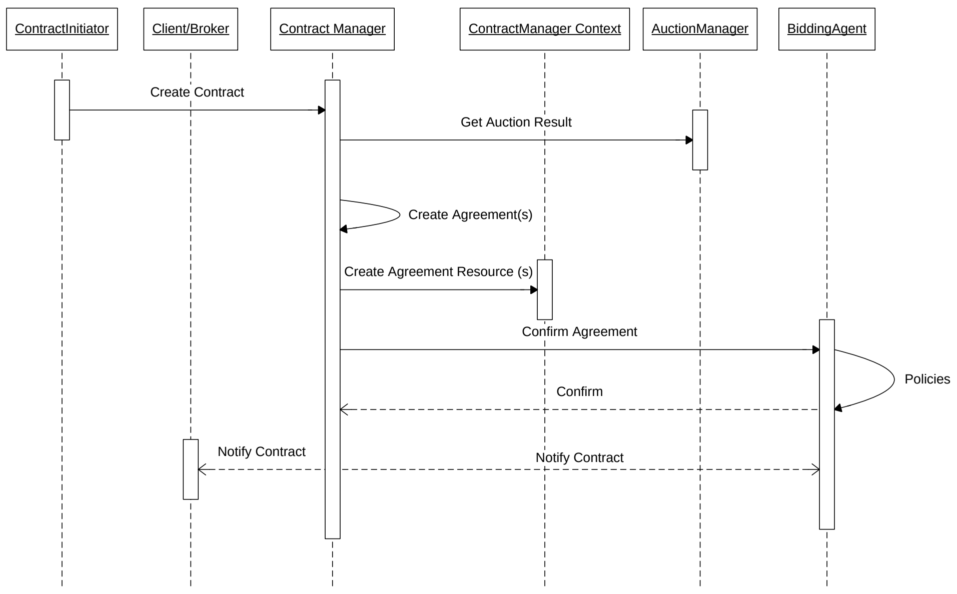


Figure 5.7: Contract creation.

Computing Second Chance Substitute Providers

Some protocols permit computation of second chance substitute providers in the event a provider withdraws from an auction after the auction has completed. This approach has the advantage of determining a new winner without recomputing the whole auction. Figure 5.8 shows the process of confirming a contract using substitute providers when the winning Bidding Agent cannot honour the tentative agreement. The winning Bidding Agent returns a signed rejection message to the Contract Manager indicating it cannot confirm the requested agreement, this message is self signed to verify that the message originates from the specified Bidding Agent. The Contract Manager passes the signed message to the hosting Auction Manager to determine if substitute providers can be computed. If so, the Auction Manager recomputes the winner by excluding the previous winner from the auction and returns the new auction result to the Contract Manager.

The contract hardening process is restarted for the new winner, replacing the original contract resource and contacting the new winner for confirmation.

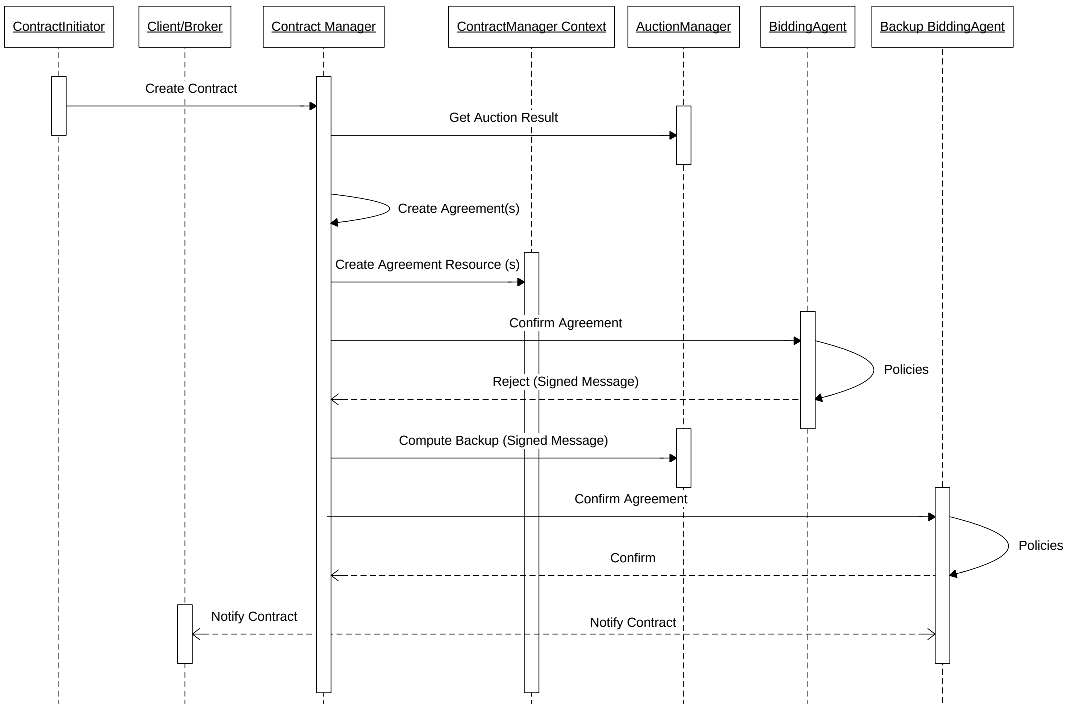


Figure 5.8: Contract creation using second chance substitute providers.

5.5 Registration and Discovery

There are two types of registration in DRIVE; first users and providers must register to join a VO (user registration), secondly providers also register services and capabilities to participate in the VO (service registration).

5.5.1 User Registration

User registration is vital for identification, authentication, and authorisation of entities within the system. The DRIVE prototype uses VOMS to manage VO membership and authenticate actions within the VO. The design of DRIVE is not bound to VOMS as other membership services may by more suitable in different domains. To demonstrate this flexibility the DRIVE based Social Cloud presented in Chapter 8 leverages Facebook as a means of user registration and authentication (VO management).

VOMS is a database-based mechanism used to manage authorisation information within multi-institutional collaborations. User roles and capabilities are stored in a database and a set of tools are provided for accessing and manipulating data. Credentials for users and resource providers are generated when required based on stored authorisation information. VOMS credentials extend authorisation information stored in standard X.509 based Grid proxies by including role and capability information as well as VOMS server credentials. Resource providers maintain access control lists to make authorisation decisions based on groups, VOs, roles, or capabilities. VOMS has the advantage that resource sites do not need to retrieve all VO lists multiple times a day, rather VO information is pushed through the certificate.

In the DRIVE prototype users and providers join the VO by registering with VOMS, this is done by passing an existing Grid certificate to the VOMS server. As is the case with normal Grid environments participants obtain certificates from Certificate Authorities. VOMS records user information and issues proxy certificates (extended Grid certificates with VO information) to be used for authentication. In the DRIVE model VO members may be part of several VOs and have any number of assigned roles and groups.

Having been granted VO membership by VOMS, users can generate a proxy to access Grid services using *voms-proxy-init*. Like the standard GT4 *grid-proxy-init*, a proxy certificate is created that is then used to transparently access services. An example of a VOMS proxy in DRIVE is shown in Appendix C (Listing C.5), the embedded VO information is included at the bottom of the proxy. DRIVE Services include specific security descriptors which direct access decisions to the specified Policy Decision Point (PDP) which in turn extends the abstract *VomsPDP* methods. Services call context methods to obtain user credentials, including *VomsCredentialInformation* which provides methods to get VO information and user roles relating to the caller. The DRIVE prototype only uses a subset of VOMS capabilities. Currently it checks VO membership of users to ensure they are in the DRIVE VO. However, the infrastructure is easily extensible to make fine grained access decisions based on user or VO roles and capabilities.

There are several limitations using VO management infrastructure in a global federated environment. As the main authorisation mechanism used in DRIVE it must be trusted and as such requires dedicated infrastructure to ensure the service is not compromised. Additionally consideration must be placed on scalability and fault tolerance ensuring the system is suitable for large scale environments. To limit these problems VOMS resides in the trusted core of the DRIVE implementation.

5.5.2 Service Registration

Providers register obligation and participation services upon joining the VO, the addresses of which along with resource profiles describing capabilities are registered with the Globus MDS Index Service. This DRIVE specific metadata allows auction advertisements to be targeted towards suitable entities that support the chosen protocol and have sufficient capacity. In addition a general purpose description of the service and the hosting environment is specified according to CaGrid Metadata schemas[2]. This semantic metadata describes the service and also specifies non-functional information such as the hosting provider description, physical address, and administrator contact details.

The proof of concept resource profile schema is shown in Appendix C (Listing C.6). This profile quantifies available capacity in terms of a three tuple containing Computation, Memory and IO. The profile also attempts to classify the provider according to one or more of these categories to indicate the type of jobs the provider is most interested in hosting. Auction protocols supported by the Bidding Agent are listed so auctions are not targeted at providers who are unable to participate in the chosen protocol. The initial value for each element is set in a bidding properties file deployed with the Bidding Agent, run time updates are based on user specified policies. Registration properties such as MDS addresses and refreshment periods are also configurable and are described in a separate registration properties file.

Each DRIVE Agent periodically submits resource profiles to be stored in the Index service, the values of which can be set dynamically or loaded from configuration files located on the host machine. Through Index service aggregation this information propagates around the VO ensuring global knowledge of the provider. The resource provider may choose to also register a standard Globus service data entry which summarises capabilities and current load of the provider in terms of processors, jobs, and waiting times. This information however is not currently considered in the DRIVE prototype.

The DRIVE approach to resource profiling is simplistic, however it can be used to great effect to reduce auction size by targeting auction advertisements at suitable providers. A more thorough representation of provider capacity could be used to increase the accuracy and expressiveness of resource profiles. For example mathematical functions can be used to describe projected or measured capacity over time. However in commercial environments providers may be unwilling to divulge projected capacity as it may be commercially sensitive. There may also be motivation to lie about capacity to ensure providers are notified of all auctions. Expressing capacity with functions has been proposed in SLA negotiation [224], in this work SLA terms are represented as multidimensional functions (or systems of functions) rather than constant values or variable ranges. The major limitation with this approach is difficulty generating the functions and increased matchmaking complexity.

5.5.3 Resource Discovery

Discovery mechanisms in any distributed infrastructure provide the backbone for cooperation. DRIVE relies on discovery of previously unknown entities to implement meta-scheduling mechanisms, locate providers, and determine market conditions. Users discover DRIVE services to manage auctions and create contracts while DRIVE services discover other services to implement auction protocols. In DRIVE filtered discovery mechanisms are used to identify suitable resource providers based on resource profiles in an effort to optimise allocation through targeted adver-

[2]`http://www.cagrid.org/display/metadata13/Metadata+Models`

tising. In addition both users and providers may want to discover auction results to determine market conditions.

Service and provider discovery relies on the Globus MDS Index service. Due to the standardised nature of the Index service there are a number of ways to access metadata. MDS exposes mechanisms by which XPath statements are evaluated against registered metadata and filtered results are returned. For most use cases in DRIVE the data is extracted using the generic Resource Property API, in which any resource property from any WSRF resource can be retrieved (assuming the entity is authorised to access the information). By specifying an XPath statement the resource providers returned are filtered against specific criteria. XPath queries are very expressive and are not limited to straight equality comparisons, there are capabilities to use inequalities, functions such as search or count, or parts of other elements. For example, when advertising a resource allocation in DRIVE, the Auction Manger can query the index service to return all providers classified as heavy computation providers or specify a ranking "of more than x" for each of the identified resource profile categories. An example query used to discover appropriate providers is shown in Appendix C (Listing C.7).

The DRIVE Publishing Service acts as a general purpose whiteboard where Auction Managers can publish auction advertisements and auction results for entities within the system to act upon. Auction advertisements allow any provider not explicitly notified of an auction the chance to participate. Published auction results allow non-winning entities and non-participating entities the chance to determine market conditions. While similar functionality could be implemented using MDS a separate Publishing Service has been implemented as it provides complete control of information and the ability to implement fine grained security and sharing policies. Additionally, providers can contribute Publishing Services to the VO as they do not need to be trusted entities and there is no requirement for information propagation.

5.6 Security

Grid and federated systems present complex security considerations as VOs span multiple administrative domains and contain heterogeneous resources each with independent security mechanisms, implementations, and policies. The DRIVE prototype considers a number of levels of security ranging from VO level security down to securing individual resources and communication channels.

The DRIVE prototype is based on the Grid Security Infrastructure (GSI) [202] as it is the de facto Grid security standard. GSI provides single sign on, credential delegation, secure communication using Secure Sockets Layer (SSL), and supports implementation of security mechanisms across organisational boundaries. Like Globus, the DRIVE prototype relies on X.509 certificates for authentication, every entity and service in the system is identified by a certificate. VO level security is required for general VO operations to make all parties accountable for the actions they take within the VO. DRIVE requires mechanisms that maintain site autonomy whilst also interfacing with a global authentication and authorisation system. As discussed in Section 5.5.1 VOMS provides this global VO management in the prototype. VOMS groups participants and provides authentication and authorisation mechanisms in a federated environment.

When considering the individual services that make up DRIVE there are additional security concerns. Many DRIVE services rely on plug-in components to provide flexibility and extensibility, for example the plug-in auction protocol mechanism supplied by the Auction Manager. The responsibility for ensuring services and their plug-ins do not compromise security is left to resource provider administrators. Every DRIVE service has been implemented with a flexible security model allowing administrators the option of configuring security levels such as service, method, and/or resource level security. The core DRIVE services each have differing default security configurations depending on their functionality, some services implement service level security allowing only particular entities to access the service, for example the Reservation Service is configured by default to only allow requests from local resource providers (Bidding Agents or Execution Services). Other services rely on method level security allowing entities access to some methods and not others, for example the Auction Manager restricts access to auction management methods whereas any entity (in the VO) can attempt to start an auction. Other DRIVE services rely on resource level security ensuring only certain entities can access particular resources, for example creator only security limits access to only the resource creator and is used so that only authorised entities can access reservations and auction state.

The DRIVE prototype makes use of secure channels and signed messages to ensure communication originates from the identified entity and is not altered. The secure auction protocols implemented have proprietary security mechanisms included in the protocol design such as garbling of circuits to obfuscate bid data. Care must be taken when creating or porting new protocols to ensure the DRIVE architecture does not compromise the validity of the protocol.

5.7 High Utilisation Strategies

The widespread adoption of commercial providers has re-motivated the use of economically aware allocation protocols to efficiently allocate resources in distributed environments. Economic allocation protocols, and in particular auction protocols have been widely studied in a distributed context with varying results. While auction protocols provide an ideal low communication mechanism for determining the market price for a good and producing optimal allocation they have some limitations in a high performance scenario due to their inherent latency. For this reason the worst case performance in an auction scenario is one composed of frequent short duration jobs. There has been little study of strategies which can be employed in economic environments to maximise occupancy and therefore utilisation. In the design and implementation of DRIVE several strategies have been developed that can be used to increase utilisation. In particular, strategies regarding bidding time, overbooking resources, computing second-chance substitute providers, and using flexible reservations have been implemented and evaluated.

5.7.1 Overbidding and Overbooking

The two phase contract creation mechanism used in DRIVE creates the opportunity for providers to implement unique negotiation and reservation policies. There is potential for considerable latency between initial negotiation and eventual contract establishment, this latency greatly impacts utilisation if resources are reserved while waiting for the result of the negotiation. In particular, an allocation generally has a single winner, and multiple, m, losers. While the winner gains the eventual contract, there is no such compensation for the m losers of the allocation process, and any resources r put aside during the allocation will decrease the net utilisation of the system by mr. From a providers perspective there is an opportunity to increase occupancy and therefore utilisation by participating in negotiations that could exceed capacity in the knowledge that it is unlikely they will win all allocations. Knowledge of existing negotiation can also be used in subsequent valuations, thus incorporating the possibility of incurring penalties for violating agreements. This process of entering negotiations beyond capacity, is termed *overbidding* and is most appropriate in an auction or tendering scenario.

Overbooking extends this concept to contract creation, allowing providers to create contracts that exceed available capacity in the knowledge some jobs may not use the resources "booked". The risk associated with overbooking is greater than overbidding as the penalties imposed for violating hardened contracts are generally much greater. However overbooking has been shown to provide substantial utilisation and profit advantages [200, 225] due to "no shows" (consumers not using the requested resources) and overestimated task duration.

While overbooking may seem risky it is a common technique used in yield management and can be seen in many commercial domains, most notably air travel [226, 227] and bandwidth reservation [228]. Most airlines routinely overbook aircraft in an attempt to maximise occupancy and therefore revenue by ensuring they have the maximum number of passengers on a flight. Without overbooking full flights often depart with 15% of seats empty [226]. Overbooking policies are carefully created and are generally based on historical data. Airlines acknowledge the possibility of an unusulaly large proportion of customers showing up, and include clauses to "bump" passengers to later flights and specify compensation to be paid [229]. Techniques used in bandwidth reservation have strong correlation to the type of scenario seen in distributed resource allocation. Typically network traffic is inconsistent and bursty by nature, as such consumers do not use all of their allocated bandwidth all the time. Telecommunications companies utilise this knowledge when selling bandwidth by overbooking in an attempt to increase revenue, this technique ensures maximum utilisation of the finite network resources deployed [228].

Overbooking and overbidding attempt to balance the revenue lost due to unused capacity and the penalties imposed by breaking contracts with consumers. Essentially creating a maximisation equation from which providers can determine the optimal amount of overbooking. However, the cost of non financial penalties such as unspecified damage to a providers reputation is difficult to calculate. Due to the widespread adoption of overbooking techniques there is substantial economic theory underpinning provider strategy [230, 231].

While overbooking has been previously suggested for Grid scheduling [232], few solutions and/or implementations have been presented. In [200] an overbooking scenario is simulated to compensate for no-shows and over estimated task duration. In [233] a probabilistic backfilling technique is used, where probabilistic models are applied to user estimated task durations, backfilling occurs if this value is less than a given threshold. Similarly, in [225] backfilling is combined with overbooking, where overbooking decisions are based on SLA risk assessment generated from job execution time distributions.

In DRIVE, the Reservation Service allows the provider to control the amount of resources committed to particular tasks, thus allowing specific overbooking policies to be defined and implemented. These policies are left to providers as there is no ideal approach, rather policies need to be carefully defined based on unique provider-specific historical data. One way of defining appropriate overbidding and overbooking limits is to use probability analysis to calculate the expected rate of winning auctions, no shows, and task underutilisation. These probabilities can then be used to create revenue maximising equations for providers. The remainder of this section considers approaches to defining an overbooking function based on expected no shows. The same approach can be taken to determine overbooking limits based on expected underutilisation and overbidding limits based on the probability of winning an auction.

Overbooking Limit

The simplest approach to define an overbooking limit (O), is to calculate the ratio of expected no shows to available capacity as described in Equation 5.7. For example with a no show probability of 0.5 a provider could potentially employ an overbooking limit of twice the available capacity of the system.

$$O = \frac{capacity}{1 - P_{noshow}} \tag{5.7}$$

While this approach is simple to implement it is overly simplistic and does not accurately model no shows. In airline overbooking it is common to model passenger no shows with a binomial distribution [234]. Binomial distributions can be used to determine the number of successes in a given sample. The same model can be applied to overbooking resources in a Grid environment. Given a sufficiently large number of jobs, a job can be treated as a passenger in the airline model. For example if the no show probability of a job is denoted p, then the probability of their being exactly i no shows from R confirmed reservations is:

$$Pr_i(R) = \binom{R}{i} p^n (1-p)^{R-i} \tag{5.8}$$

The cumulative probability is therefore given by:

$$Pr(X \leq x) = \sum_{i=0}^{x} \binom{R}{i} p^i (1-p)^{R-i} \tag{5.9}$$

The cumulative probability model can be applied to create an overbooking policy relating to the chance of there being a certain number of no shows. For example, a provider can specify that there is at least a probability 0.8 of there being less than x no shows, this information can be used to overbook a resource by x jobs with 80% certainty that there will be sufficient capacity.

$$Pr(X \leq x) \geq 0.8 \tag{5.10}$$

These equations implicitly assume homogeneous jobs and a limit based on the number of jobs hosted rather than a particular attribute capacity (CPU, memory), this is possible assuming a provider hosts a large number of jobs. In the case of a small scale provider in which few jobs are hosted the different requirements of each job must be considered. To represent this situation a measure of job requirements must be incorporated in the model. The difficulty is it is impossible to determine which of the jobs will be a no show (it could be a small job or it could be a big job) and any number of properties could be represented: CPUs, memory or disk. To model the difference between jobs a class like model could be used, again this is analogous to classes in airlines. Different classes of jobs may exhibit different no show properties and can therefore be used to create a more accurate model.

These models can be used to approximate the probability of no shows and therefore calculate projected utilisation, however this only considers one side of the overbooking strategy. In an economic environment, overbooking could be conducted relative to projected profit (including potential penalties) rather than projected utilisation as profit is presumably the most important maximisation target for a commercial provider. For example, it makes sense to overbook resources up until the point where profit is maximised. To do this the probability of incurring a penalty, and the value of the penalty must be considered. Similar models can be developed to determine the probability of incurring penalties which can then be combined with a profit maximising equation. Cumulative underutilisation values can also be included in an overbooking model. Likewise the probability functions outlined for overbooking, apply in the overbidding situation too. For brevity they will not be repeated.

Overbooking Revenue

In an overbidding and overbooking scenario revenue for the provider must include any penalties enforced for reneging on a tentative agreement or breaking a hardened contract. Revenue is defined as the total price paid for all jobs hosted minus the total penalties enforced. Equation 5.11 defines total provider revenue, where the revenue for each hosted task $i \in (1, n)$ is denoted R, the penalty for breaking a tentative agreement or contract is denoted P for m agreement rejections and o contract rejections.

$$R_{net} = \sum_{i=1}^{n} R_i - (\sum_{j=1}^{m} P_j + \sum_{k=1}^{o} P_k) \tag{5.11}$$

5.7.2 Second Chance Substitute Providers

In distributed environments resource state can change rapidly, limiting available capacity and therefore invalidating outstanding bids. If a winning provider can no longer meet their obligations, it wastes resources to repeat the auction process when there is sufficient capacity available from non winning providers. In some protocols, losing bidders can be given a *second chance* to win the auction, by re-computing the auction without the defaulting bidder. This optimisation will increase efficiency at the expense of some protocol security, for example more bid information may be released in a secure protocol.

Allowing second chance providers can be used to reduce the allocation failures generated from overbooking, therefore increasing both occupancy and utilisation of the system. One negative aspect of this approach is the potential for increased consumer cost, as they will now pay the price of a more expensive bidder. The net increase in cost (overhead) is given by the difference between the two bids (Equation 5.12). This additional cost can be offset through compensation imposed on the violating party. The additional cost to the consumer is given by the difference between the two bids offset by any penalties $P_i, i \in (1..n)$ enforced (Equation 5.13).

$$R_{overhead} = B_{substitute} - B_{original} \tag{5.12}$$

$$R_{consumer} = (B_{substitute} - B_{original}) - \sum_{i=1}^{n} P_i \tag{5.13}$$

In DRIVE second chance providers are optional and must be enabled by the task submitter and supported by the auction protocol. Consumers and providers are made aware of the protocol semantics in the protocol description and may choose not to participate. Auction policies determine if protocols allow computation of substitute providers. Policies may also dictate the number of substitutes which may be queried to reduce violating protocol properties. In Chapter 6 it is shown on average only 1 to 2 second chance providers are consulted when allocating a heavy synthetic workload with substantial auction latency. This type of information can be used to develop heuristics defining the maximum depth of substitutes permitted and also be used to reason about the security and privacy preserving properties of the protocol.

Second chance substitute providers are charged their bid price (or the new second price in a second price auction) rather than the defaulting price for two reasons, first this is their true valuation of the goods and therefore they are prepared to provide the service for this price. Secondly if the Auction Manager were to offer the goods for the defaulting price there is additional communication (and associated protocol) required to negotiate the new allocation. Paying their bid price requires no additional functionality added to the protocol and penalties can be used to compensate the consumer for the increase in price.

Rejecting a contract is equivalent to withdrawing a bid after the conclusion of the auction. Bid withdrawal is not often supported by auction protocols for a number of reasons, for example in open outcry auctions bid withdrawal can be used to artificially inflate the sale price. In sealed bid protocols withdrawal requires re-computation of the auction which may reveal additional information, such as the second price. While revealing the second price may seem trivial it could invalidate the winners valuation of the goods due to the *winners curse*. In secure auction protocols, withdrawal and re-computation can be complex and may release sensitive information if the protocol needs to decode multiple bids in order to determine which bidder to remove. Bid withdrawal in combinatorial auctions is even more difficult as finding an alternative solution without disturbing other winners may be impossible [235]. For these reasons secure combinatorial auction protocols and open outcry protocols are unlikely to permit re-computation of an auction. However, sealed bid non-combinatorial protocols can (most often) support such actions.

5.7.3 Flexible Advanced Reservations

Advanced reservations are commonly used to provide consumers with QoS guarantees and facilitate client based scheduling based on inter-dependencies (for example in workflows). However the advantages from a providers perspective are often neglected. Advanced reservations allow job scheduling in advance, which facilitates resource reservation (with a binding contract) without requiring real time scheduling. The flexible reservation model used in DRIVE creates an opportunity for providers to implement advanced scheduling techniques to optimise system utilisation [178]. Techniques such as backfilling, greedy algorithms, and heuristics can all be used to optimise provider scheduling due to the relaxed deadline requirements.

5.7.4 Just-In-Time Bidding

All auction protocols have inherent latency in the allocation process. During this period resource state may change thereby invalidating a providers valuation (or bid). In general, there are three ways to minimise the effect of latency:

1. **Reduce the duration of the auction.** The problem with this approach is that there is minimal time for providers to discover the auction and to compute their bids.

2. **Bid as late as possible.** This has the main advantage that providers can compute their bids with the most up to date resource state therefore requiring resources to be reserved for a shorter time. The primary problem with this approach is time sensitivity, the auction can be missed if the bid is too late or experiences unexpected network delays.

3. **Allow bid withdrawal or retraction.** While this provides an easy way for parties to invalidate bids, it is not always supported and may result in penalties being imposed. In addition this approach is time sensitive, as retraction must occur before the auction closes.

In some environments, for example open outcry protocols used in online auctions, Just-In-Time (JIT) bidding is common and has additional strategic advantages for combating shrill bidding and incremental bidding. For sealed bid auctions, JIT bidding traditionally has been seen to have no advantages. However, as shown in Chapter 6, significant utilisation advantages can be gained in sealed bid auctions due to the reduced auction latency and therefore more accurate capacity prediction.

Bid retraction and withdrawal can also be used to mitigate the effects of inaccurate capacity predictions due to latency. Bid retraction occurs during the submission period, whereas bid withdrawal occurs after the submission period has finished. Bid withdrawal in DRIVE is essentially the same as reneging on a tentative agreement and has the same advantages and disadvantages as discussed in Section 5.7.2.

In open outcry auctions it is typically not within the rules to retract a bid in part because bidders values may be based on observed bids and also because retraction (and withdrawal) could lead to cheating (artificially inflating prices). In sealed bid protocols it is sometimes possible to retract bids as it often does not effect the protocol. In secure auction protocols, bid retraction has the same limitations as bid withdrawal, while it may be possible to retract a bid it can be a computational expensive process and may release additional information. For example, in the case of threshold schemes several entities would collectively need to retract the bid, which may violate security properties of the protocol. Hidden information can also be revealed as entities may need to decode all bids in order to determine which bid to retract.

Bid withdrawal and retraction are often supported in simultaneous auctions in which individual goods have synergistic value (combinatorial) [236]. In these auctions without withdrawal bidders cannot bid aggressively as they may win subsets of required goods which reduces their synergistic value. An example of such an auction is the Federal Communication Commission spectrum auction, where each bidder simultaneously submits sealed bids for each license they are interested in. Bidders are given the option of retracting bids if they win multiple auctions.

JIT bidding provides the most generic approach to minimising latency requiring no alteration of auction management or protocols. Bid retraction and withdrawal are only supported in a subset of potential protocols and their use may be restricted due to complexity or security considerations. JIT bidding is easily implemented by providers without requiring knowledge of the given protocol rules. Bidding policies can also be applied to maximise returns based on bidding times.

5.7.5 Summary

Economic allocation and particularly auctions have been stereotyped as a low performance allocation solution due to the overheads and latency in the allocation process. This section has presented several strategies that can be employed in a distributed computational economy to increase occupancy and optimise utilisation. The utilisation techniques presented have not been collectively examined from the view of improving auction performance in any prior work. In addition to auctions, these strategies may be applicable in other forms of economic and non-economic resource allocation. Chapter 6 examines the effectiveness of these proposed approaches under differing auction workload scenarios.

Chapter 6

DRIVE Evaluation

Due to the scope of DRIVE there are a number of areas that warrant experimental validation. The following chapter presents experiments focused on both the individual components of DRIVE and the architecture as a whole. Section 6.2 describes experiments designed to evaluate various DRIVE components, services, and mechanisms. A brief analysis of the supported auction protocol implementations is covered in Section 6.3. Section 6.4 presents the effect of bidding policies on task distribution. Section 6.5 outlines the analysis of a production Grid workload and the creation of several synthetic workloads used to test the performance of DRIVE in different scenarios. Section 6.6 quantifies the increased occupancy and utilisation obtained from using the high utilisation strategies proposed in Chapter 5. The economic implications of these strategies combined with various bidding and penalty functions are then presented in Section 6.7. Section 6.8 measures the throughput of DRIVE before Section 6.9 presents the overhead of hosting core DRIVE services.

6.1 Experimental Testbed

All the experiments presented in this chapter are conducted on a virtualised testbed using a combination of two subsets of machines. Each subset is composed of 5 machines with the following specifications:

- **Subset A:** Core 2 Duo 3.0 GHz, 4 GB of RAM, Windows Vista
- **Subset B:** Pentium 4 3.2GHz, 1 GB of RAM, Fedora Core 5

Individual experiments utilise between 1 and 10 dedicated machines from this testbed. The protocol experiments described in Section 6.3 are conducted entirely on Subset B as these experiments were performed before Subset A was acquired. The DRIVE component and task distribution experiments presented in Section 6.2 and 6.4 respectively utilise machines from Subset A. The remaining experiments (high utilisation, economic implications, throughput and overhead) all use a combination of Subset A and Subset B. In these experiments a single Auction Manager, Contract Manager and MDS are run on separate hosts from Subset A in a container with 1 GB of memory allocated. The virtualised providers (Bidding Agents) are allocated 256 MB of memory to each hosting container and are evenly distributed over the remaining machines from both Subset A and B. When both subsets are used concurrently Subset B is used to host virtualised bidders only. Jobs are submitted from a client broker application outside the testbed using a machine with the same specifications as Subset A. An experimental configuration with 20 bidders is depicted in Figure 6.1

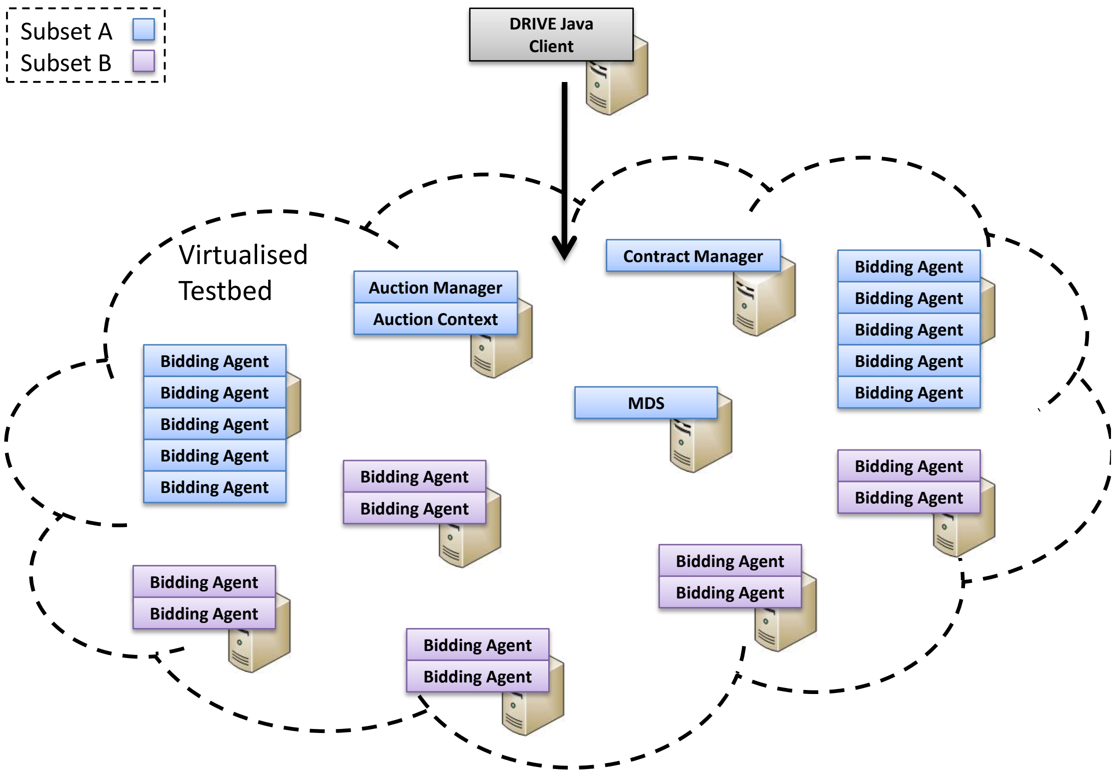


Figure 6.1: Experiment testbed configured for 20 virtualised bidders participating in an auction.

The client broker is implemented as a multi threaded Java application with capabilities to automate experiments by simulating experimental workloads. Experimental workloads can be generated through characteristic functions or based on pre-defined *DRIVE trace files*. The multi threaded architecture is used to create JSDL job descriptions, start auctions, and establish contracts following the specific workload without blocking on requests. The workload defines the interarrival time and individual job requirements which are used by the broker to submit heterogeneous jobs at the specified time. The client broker is able to poll and/or receive WS-notifications from the Auction Manager and Contract Manager. In most experiments job requirements are encoded in a JSDL document along with a string describing additional experiment specific information (bidding strategy, bidding policies, reservations) this allows centralised configuration of bidding behaviour for individual experiments (or jobs). The auction request defines optional Contract Manager(s) and Publishing Service(s) to be used along with auction end conditions (duration or maximum number of bidders).

Performance measurements are either measured client side or service side depending on the requirements of the experiment. Client side measurements quantifying auction and contract performance are logged by the broker for every job in a workload. In addition workload summary files are created describing each job, auction result, contract result (including original winner and substitute winners), and performance measurements for a single workload. Each service also has performance measurement and logging functionality to record the time taken to perform various actions. In addition Bidding Agents log real time monitoring information to files describing pricing, bids, contracts, and system utilisation. This information is used to analyse results from the perspective of each individual provider.

Each of the DRIVE services is configurable with respect to thread pool sizes, service addresses (MDS registration and queries, publishing services), logging options, monitor timeouts, resource lifetimes, and protocol or valuation plug-ins. The Bidding Agent can be configured to use statically defined bidding policies and valuation functions (for example in the Grid bidding policy experiments) or can be configured dynamically through the job description (for example in the high utilisation experiments). The advertised metadata can also be configured statically depending on the experiment. Registered metadata is used to simulate different auction scenarios, for example it is used to determine if bidders are directly advertised jobs or if they must discover auctions through Publishing Services.

The resource usage of each DRIVE service is measured using a customised monitor in the Windows Performance and Reliability Monitor. The experiments are set up such that each individual service runs on a dedicated host to ensure accurate measurements.

6.2 Component Evaluation

In the implementation of the DRIVE architecture the performance of discovery, contract creation, and reservation mechanisms have been analysed independently to confirm design decisions. This section summarises experiments on these components independent of the complete DRIVE architecture.

6.2.1 Discovery Performance

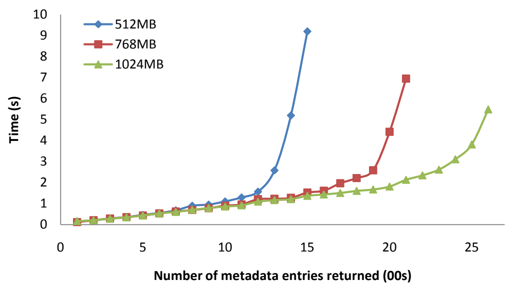
DRIVE utilises MDS to store and query registered metadata describing the capabilities of providers. There are three general queries used in DRIVE; simple metadata queries, text comparison based queries, and inequality based queries. Simple metadata queries place no select conditions and return all registered DRIVE metadata allowing filtering to be conducted on the client side. Comparison queries are used to find text based data in the registered metadata and inequalities are used to filter numeric data according to user defined limits.

The cost of registering metadata is simply the cost of creating a WSRF resource which is constant (approximately 3ms in this testbed). Propagation of information around the VO is performed out of band by the Index Service and as such is not considered in this analysis. A comprehensive MDS performance study is presented by Schopf et al. [237].

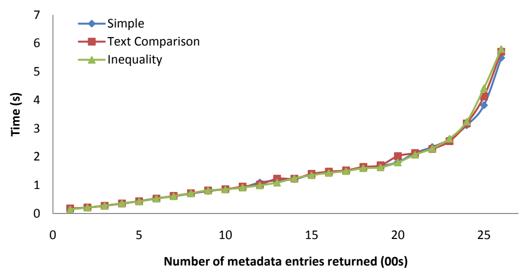
Figure 6.2(a) shows the time taken to retrieve an increasing number of DRIVE metadata entries with 512MB, 756MB and 1024MB of memory allocated to the MDS container. In this experiment every registered entry is returned and the number of registered entries is steadily increased. The graph shows a linear relationship between retrieval time and number of entries returned up until a point of saturation based on the amount of memory allocated to the container. The linear relationship is approximately 1200 registered entries can be returned per second. The results show over 2000 individual providers can be registered and discovered within 2 seconds using a relatively small scale single MDS service (1GB RAM). This time is insignificant when compared to latency seen in distributed environments, typical auction duration, and the cost of executing (secure) auction protocols. The saturation point motivates the need for a distributed discovery infrastructure in order to scale above thousands of registered providers.

Figure 6.2(b) shows a comparison of the three different query types used in DRIVE discovery. This experiment looks at the time to query an increasing number of registered entires for a single MDS Index Service running with 1GB of memory allocated to the container. The results show there is no significant difference between the three querying approaches. On overall average, text comparison is approximately 2% slower than the simple query retrieving all entries, inequality queries are less than 1% slower than the simple query.

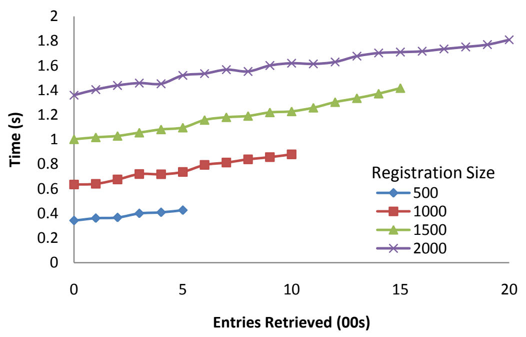
The final experiment looks at the time taken to query an increasing percentage of the total registered entries. In this experiment the Index Service is allocated 1GB of memory and an inequality query is used to control the number of results returned. Four different registration groups (500, 1000, 1500, 2000) are used to determine the effect of the number of registered entries on retrieval time. Each group defines the total number of entries registered. Figure 6.2(c) shows the time taken to query an increasing number of entries from each group, incremented by 100 entries at a time. The relationship is close to linear for each series with only a small difference in time taken between retrieving no results and retrieving all of the registered entries. The gradient of each series is 0.16, 0.24, 0.27, and 0.23 ms per entry retrieved for registration groups 500, 1000, 1500, and 2000 respectively. For each series the difference between no results and all entries returned ranges between 20-30% of the total query time for all registered entries. This can be explained by the observation that the most significant cost is processing registered entries.



(a) Discovery time for increasing registered entries and different amounts of container memory.



(b) Discovery time for different queries for an increasing number of registered entries.



(c) Discovery time for retrieving an increasing number of entries for differing total registration sizes.

Figure 6.2: Metadata retrieval time for increasing number of registered entries, different query types, and different number of results returned.

6.2.2 Contract Performance

This experiment examines the total time taken to reach final agreement when bidders reject initial agreements and substitute providers have to be computed. A single fake bidder is used to simulate a scenario in which bidders randomly reject initial agreements at a specified rate. The Contract Manager attempts to confirm the agreement with the bidder repeatedly until the bidder accepts the agreement. Figure 6.3 shows the time taken to reach final agreement at different rejection rates using both synchronous and asynchronous invocation with WS-notifications. The time is measured from when the job is submitted until the final contract is created and the consumer is notified, this time is the average time calculated over 100 auctions. The graph shows the base time with a 0% rejection rate is approximately 2s, as the rejection percentage increases there is an exponential increase in time taken to reach final agreement. These results can be interpreted as a rule of thumb – if the rejection rate is below 60% contracts on average are substituted within 200% of the base creation time. It should also be noted there is little increase in overhead between synchronous and asynchronous invocation.

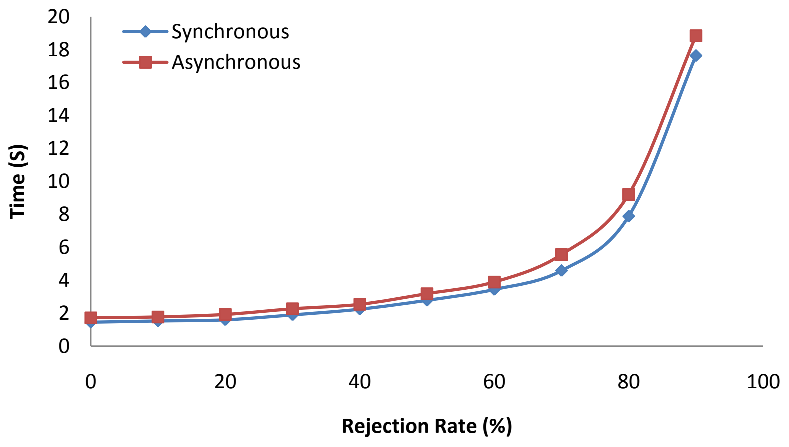


Figure 6.3: Time taken to confirm agreements with increasing rejection rate.

6.2.3 Reservation Performance

The major way in which DRIVE components interact with the Reservation Service is querying for future reservations to determine availability. Figure 6.4 shows the time taken to search forward through the reservation schedule for all reservations over an increasing period of time. The experiment is set up as a worst case example with the schedule pre-loaded with 960 1 minute reservations evenly spread over the day. The graph shows the query time increases linearly with the period of time searched and is obviously proportional to the number of reservations retrieved. The time taken to return close to 1000 reservations is less than 50 milliseconds. In comparison with the cost of valuing resources, submitting a bid and consulting the individual reservations with respect to local policies this time is insignificant. These results show the time for retrieving reservations is very fast for a reasonable number of registered reservations.

6.3 Protocol Evaluation

The DRIVE prototype includes three distributed auction protocol implementations: a sealed-bid first-price auction protocol, a sealed-bid second-price auction protocol and a secure, verifiable Garbled Circuit protocol. The performance of the first-price and second-price protocols are essentially the same for single good auctions as the difference between the protocols is only how the winning price is determined, the means of soliciting bids and determining the winner are the same. For this reason the two sealed-bid protocols are considered collectively in this analysis. The following experiments look at how the number of bidders and goods effects auction time and resource creation overhead for each protocol implemented in the DRIVE architecture. These experiments are not designed to provide comprehensive analysis of the auction protocols, rather

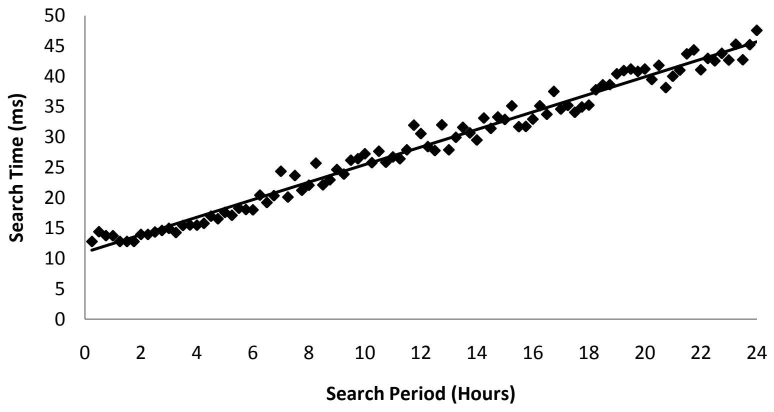


Figure 6.4: Time to search and retrieve an increasing number of reservations from the Reservation Service.

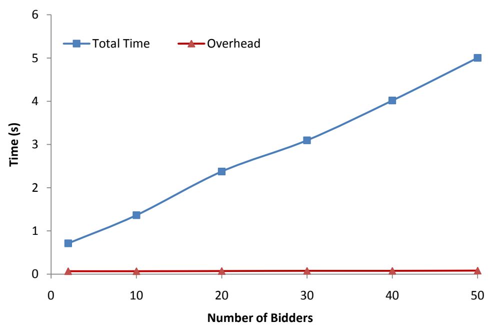
they present a high level comparison of the implemented protocols and show that they perform similarly to the same protocols deployed independently from DRIVE. Complete analysis of these protocols in a RMI scenario is presented in [238].

The following experiments are performed on Testbed B, auction times are recorded service side from when the Auction Manager receives a job request until the winner is determined. Auctions are concluded when all bids are received rather than using explicit end times. Overhead is defined as the time taken to setup auction infrastructure such as Auction Components, WSRF resources storing auction state and any protocol specific computation that can be performed before the auction commences such as the circuit creation and garbling in the Garbled Circuit protocol. Communication between the broker and auction services is not included in the auction or overhead time. The Garbled Circuit protocol uses 10 bit price representation.

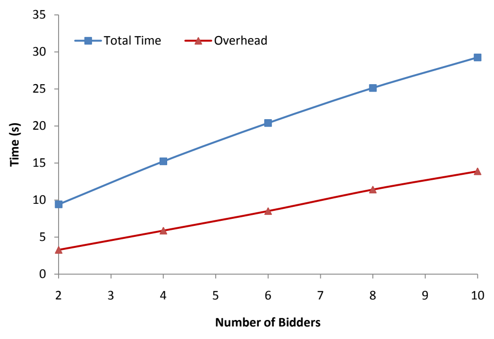
6.3.1 Number of Bidders

Auction protocol scalability is influenced by the number of bidders effectively supported. The time taken to solicit bids and determine a winner in both the sealed bid (Figure 6.5(a)) and Garbled Circuit (Figure 6.5(b)) protocols is shown to be linear. The overhead of a sealed bid auction is constant as there is only a single WSRF resource created and no preliminary computation is required. The Garbled Circuit protocol on the other hand has significant overhead which increases linearly with the number of bidders, this is due to the extra cost of instantiating an Auction Component and circuit creation/encryption used to provide trust and privacy. Comparison between Figure 6.5(a) and Figure 6.5(b) shows there is substantial computational cost in terms of auction time and overhead when using complex trustworthy protocols, the sealed bid auction takes 5 seconds to determine a winner with 50 bidders whereas the Garbled Circuit protocol takes almost

30 seconds with only 10 bidders. As mentioned previously, complex trustworthy protocols are likely to be used in high value auctions where fewer bidders would be expected and participants would be willing to sacrifice auction time for security and bid privacy.



(a) Sealed Bid.



(b) Garbled Circuit.

Figure 6.5: Number of Bidders vs Auction Time and Overhead for a Sealed Bid Auction and a Garbled Circuit Auction with a single good.

6.3.2 Number of Goods

One of the limitations of combinatorial auctions is the time taken to process auctions with a number of goods. Figure 6.6 shows the time taken to compute the winner (and the associated overhead) of the Garbled Circuit protocol when increasing the number of combinatorial goods. The graph is shown on a logarithmic scale. As expected, the time taken and overhead is exponential with the increase in the number of goods, this is due to the exponential increase in circuit size and the increased computation required for each resource combination both for bidding and for winner determination.

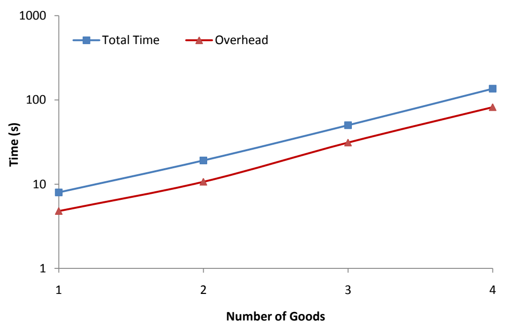


Figure 6.6: Number of Goods vs Auction Time and Overhead for Garbled Circuits.

6.4 Task Distribution

This set of experiments aims to provide a general verification of the DRIVE architecture and also demonstrate the effect of bidding policy on task distribution. To visualise task distribution 25 providers are configured in a Grid layout (each host is identified by its xy coordinate), 100 tasks are submitted to this Grid and the resulting number of tasks allocated to each host is presented. A single Auction Manager is used to conduct allocation and all providers must submit a bid for every task. Each task uses 1% of CPU time for the duration of the experiment meaning it is possible for all 100 tasks to be allocated to a single host. The delay between submitting each task is random and ranges between 0 and 15 seconds. Auction duration is 10 seconds, which means that most of the time a task will be submitted before the previous task has been allocated. Randomness in the following experiments is due to the order in which bids are recieved and the auction latency, that is the period of time between when providers bid and the auction concludes. Tied bids are resolved in a First Come First Served (FCFS) basis.

In the following experiments tasks are submitted using a simple comma separated description language designed for these experiments. This language defines the type of task, the CPU utilisation required, and the duration of the task. For example a job which uses 100% CPU for 60 seconds is denoted:

```
DRIVEGRID, 100, 60
```

This simple language serves to identify the crucial aspects of the task whilst reducing the construction and processing time of the description and resulting contract. The matrices presented in each test show, for each provider at grid position xy, the average number of tasks allocated to that provider ($\overline{n}_{xy}$) and the standard error ($\sigma_{\overline{n}}$) that defines a 95% confidence interval[1]. All tests have been run over 50 times to ensure accurate destribution and reduce the standard error. A sealed bid second price auction protocol is used to determine the allocation and providers have a bid space between 1 and 100. Resource provider bidding policy and therefore bid valuation is denoted β.

6.4.1 Random Bidding Policy

The first experiment uses a random bidding policy to form a baseline against which the later experiments can be evaluated. In this experiment each provider randomly determines a price for hosting a task irrespective of its current load ($\beta = random$). The resulting distribution matrix is presented in Table 6.1 and the resulting allocation is shown graphically in Figure 6.7.

As expected the number of tasks allocated to each provider is close to 4 (100 tasks allocated over 25 providers) and each point overlaps with all of the other coordinates when taking the standard error into consideration. The largest difference is 0.48 (3.75 to 4.23) which is roughly twice the average standard error (0.261). All points include the expected value 4.0 within their respective error margins. These results show there is little statistical significance between the

[1]A 95% confidence interval by the Central Limit Theorem is $\sigma_{\overline{x}} = 1.96\frac{s}{\sqrt{n}}$

$$\beta = random$$

$$\overline{n}_{xy} \pm \sigma_{\overline{n}} = \begin{pmatrix} 4.04 \pm 0.25 & 4.23 \pm 0.27 & 4.15 \pm 0.25 & 4.09 \pm 0.23 & 3.83 \pm 0.26 \\ 3.94 \pm 0.29 & 4.15 \pm 0.24 & 4.23 \pm 0.30 & 3.81 \pm 0.30 & 3.79 \pm 0.29 \\ 3.89 \pm 0.25 & 3.81 \pm 0.28 & 3.92 \pm 0.22 & 4.19 \pm 0.26 & 4.08 \pm 0.25 \\ 3.87 \pm 0.29 & 4.23 \pm 0.27 & 3.75 \pm 0.25 & 4.23 \pm 0.26 & 3.81 \pm 0.29 \\ 4.11 \pm 0.26 & 3.77 \pm 0.20 & 3.94 \pm 0.26 & 4.08 \pm 0.24 & 3.88 \pm 0.29 \end{pmatrix}$$

Table 6.1: Average number of tasks allocated to each provider when using a random bidding policy.

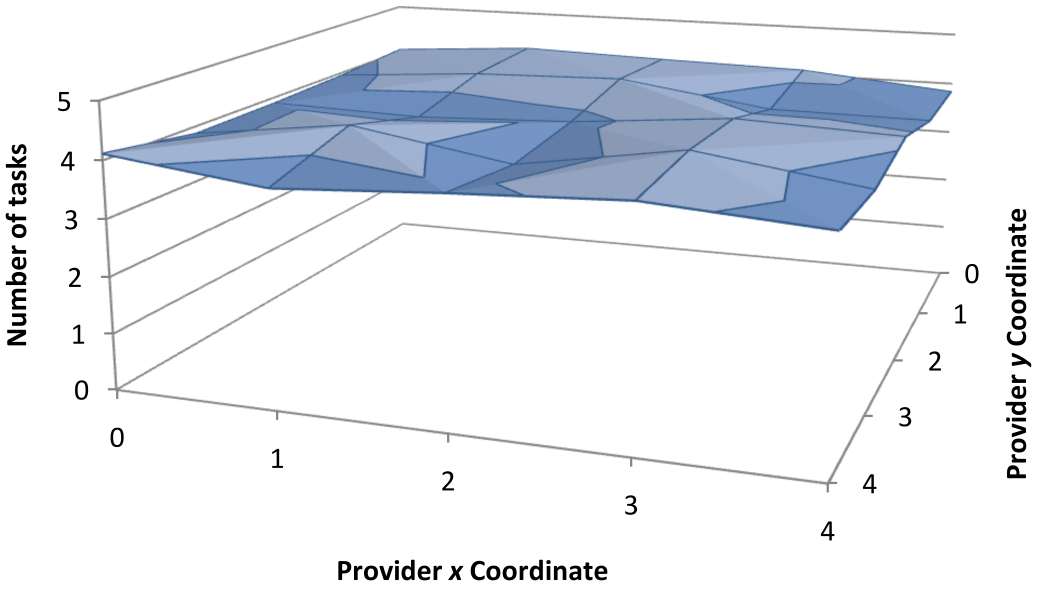


Figure 6.7: Average number of tasks allocated to each provider when using a random bidding policy.

means. This is expected as with a significantly large sample a random bidding policy should tend to 4. Figure 6.7 shows no major peaks or troughs reinforcing the even distribution of tasks over the group of providers.

6.4.2 Uniform Bidding Policy

This experiment aims to produce even task distribution over the pool of providers by using a uniform bidding model. Each provider determines the cost of hosting a task based on its current load, the bidding policy β is simply the current utilisation of the provider ($\beta = u$), where the current utilisation is equal to the number of tasks hosted on the provider as each task uses

1% of provider capacity. The bidding policy is based on utilisation so that bidding behaviour changes over time, a bidding policy independent of utilisation would result in the same bidders winning every auction. The results shown in Table 6.2 and Figure 6.8 show similar distribution to the random bidding policy as expected. The average standard error of 0.152 is smaller than random allocation while the largest difference in allocation between providers is 0.25 (3.84 to 4.09) showing that the uniform bidding model provides a more even distribution than the random benchmark model for the given number of tests. If the delay between taks was increased beyond auction latency each provider would have exactly the same number of tasks as any other because each provider would then be bidding its true utilisation at the time of auction completion.

$$\beta = u$$

$$\overline{n}_{xy} \pm \sigma_{\overline{n}} = \begin{pmatrix} 3.98 \pm 0.15 & 4.00 \pm 0.14 & 4.00 \pm 0.15 & 4.09 \pm 0.17 & 3.95 \pm 0.15 \\ 4.07 \pm 0.14 & 4.04 \pm 0.11 & 3.95 \pm 0.15 & 3.87 \pm 0.16 & 3.89 \pm 0.16 \\ 4.09 \pm 0.14 & 4.02 \pm 0.14 & 4.07 \pm 0.17 & 3.87 \pm 0.15 & 4.07 \pm 0.12 \\ 3.87 \pm 0.17 & 4.04 \pm 0.13 & 3.91 \pm 0.15 & 4.02 \pm 0.17 & 4.05 \pm 0.15 \\ 4.09 \pm 0.18 & 3.84 \pm 0.14 & 4.09 \pm 0.15 & 3.95 \pm 0.19 & 4.05 \pm 0.17 \end{pmatrix}$$

Table 6.2: Average number of tasks allocated to each provider when using a uniform bidding policy.

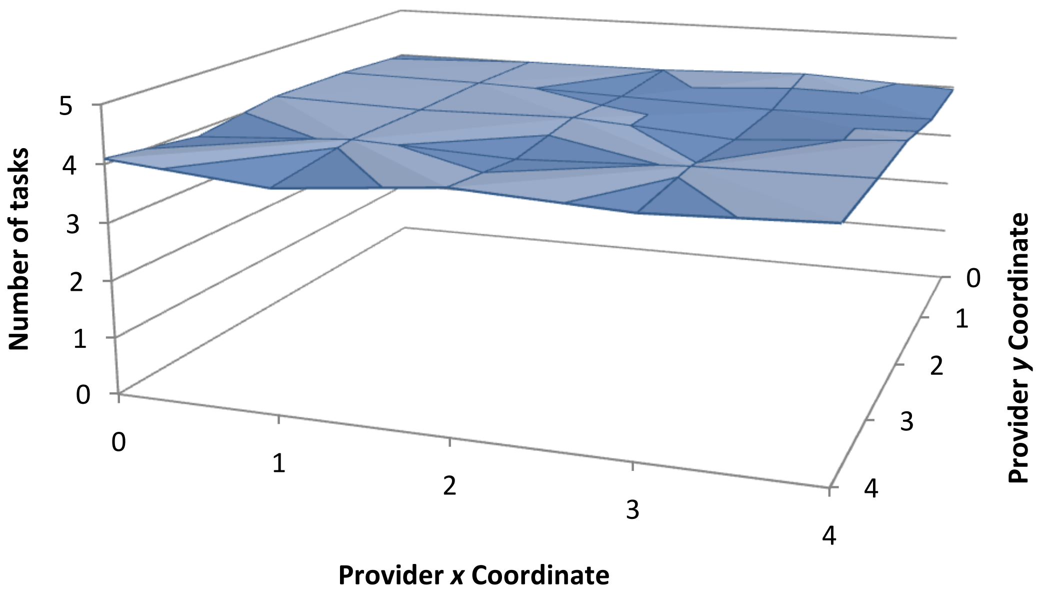


Figure 6.8: Average number of tasks allocated to each provider when using a uniform bidding policy.

6.4.3 Weighted Bidding Policy

In this experiment the bidding policy is weighted towards one corner of the grid, specifically bids are calculated as twice the sum of the providers grid position plus the current utilisation of the provider ($\beta = 2(x+y)+u$). This experiment is designed to show the effect of bidding policy on task distribution by applying an unequal weighting to each providers bid. Table 6.3 and Figure 6.9 show that most tasks are allocated to the providers with low grid positions as they have the least cost and therefore submit bids with the smallest value. The table also shows little difference in the means of the diagonals (for example [0,2][1,1][2,0]) because providers at these positions compute the same bid value. The average standard error of 0.148 is very similar to the constant model indicating little variance between experiments. The slightly larger confidence intervals for the providers at positions [0,4], [1,3], [2,2], [3,1], [4,0] is due to the number of tasks submitted in each test. In order to allocate tasks to these providers there must have been almost 50 tasks already allocated to the other positions leaving fewer tasks to be allocated over the remaining 19 providers with similar bids, this results in a large deviation between each run of the experiment due to the smaller sample size.

$$\beta = 2(x+y)+u$$

$$\overline{n}_{xy} \pm \sigma_{\overline{n}} = \begin{pmatrix} 11.54 \pm 0.08 & 9.82 \pm 0.11 & 7.49 \pm 0.11 & 5.25 \pm 0.15 & 3.51 \pm 0.20 \\ 9.47 \pm 0.11 & 7.42 \pm 0.10 & 5.40 \pm 0.17 & 3.63 \pm 0.21 & 1.86 \pm 0.26 \\ 7.77 \pm 0.14 & 5.91 \pm 0.17 & 3.39 \pm 0.20 & 1.65 \pm 0.40 & 0.00 \pm 0.00 \\ 5.25 \pm 0.11 & 3.65 \pm 0.18 & 1.61 \pm 0.32 & 0.00 \pm 0.00 & 0.00 \pm 0.00 \\ 3.68 \pm 0.19 & 1.60 \pm 0.49 & 0.00 \pm 0.00 & 0.00 \pm 0.00 & 0.00 \pm 0.00 \end{pmatrix}$$

Table 6.3: Average number of tasks allocated to each provider when using a weighted bidding policy.

6.4.4 Weighted Bidding Profile II

To further illustrate the effect of bidding profile on task distribution the final experiment demonstrates an increased weighting towards the centre of the grid. Here the bidding policy is based on the Euclidean distance from the centre ($\beta = 5\sqrt{(x-2)^2+(y-2)^2}+u$), therefore those providers closer to grid position (2,2) produce the smallest bids. The results shown in Table 6.4 and Figure 6.10 clearly show a preference for the centre of the grid with the provider (2,2) hosting on average 12.8 tasks. There are no tasks allocated to the providers in the corners as their bid valuations with no load is almost 14, which will never be a winning bid when only 100 tasks are submitted. In order for these providers to win an auction the other 19 providers must compute a value equal to or larger than 14 which requires at least 122 tasks previously allocated. The average standard error for the entire Grid is 0.153, which reflects the small variance between experiments.

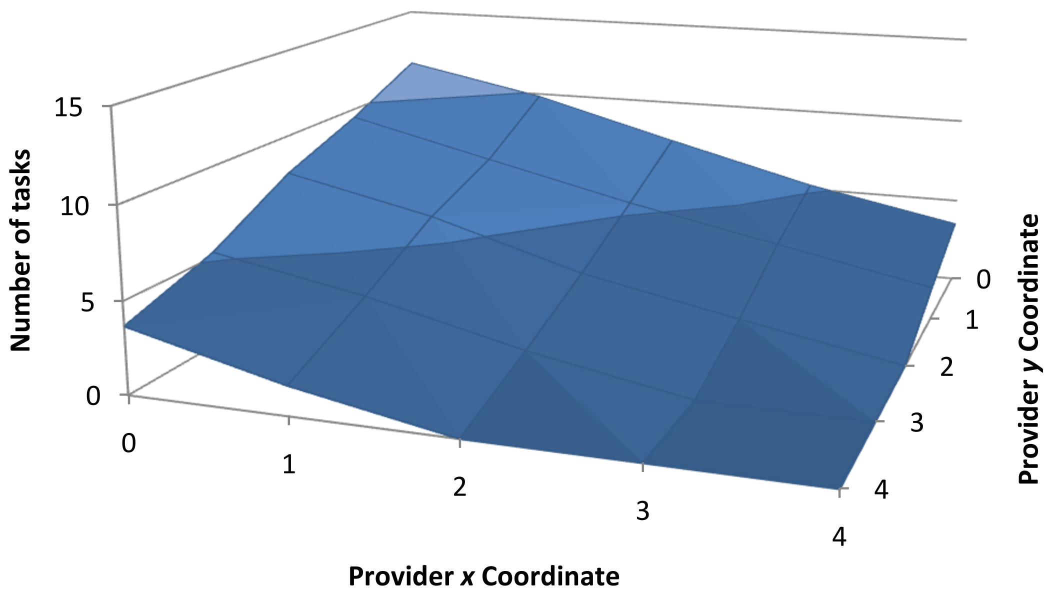


Figure 6.9: Average number of tasks allocated to each provider when using a weighted bidding policy.

$$\beta = 5\sqrt{(x-2)^2 + (y-2)^2} + u$$

$$\overline{n}_{xy} \pm \sigma_{\overline{n}} = \begin{pmatrix} 0.00 \pm 0.00 & 2.45 \pm 0.22 & 3.11 \pm 0.21 & 2.42 \pm 0.26 & 0.00 \pm 0.00 \\ 2.24 \pm 0.24 & 6.02 \pm 0.12 & 8.25 \pm 0.12 & 5.95 \pm 0.12 & 2.22 \pm 0.24 \\ 3.04 \pm 0.17 & 8.09 \pm 0.13 & 12.80 \pm 0.09 & 8.29 \pm 0.15 & 3.05 \pm 0.20 \\ 2.27 \pm 0.23 & 6.07 \pm 0.12 & 8.00 \pm 0.14 & 5.76 \pm 0.12 & 2.44 \pm 0.22 \\ 0.00 \pm 0.00 & 2.27 \pm 0.26 & 2.95 \pm 0.20 & 2.13 \pm 0.24 & 0.00 \pm 0.00 \end{pmatrix}$$

Table 6.4: Average number of tasks allocated to each provider when using a Euclidean distance weighted model.

6.4.5 Bidding Policy Summary

These results demonstrate that the DRIVE meta-scheduler, and in particular the allocation mechanisms, are acting as desired. It is clear from the results presented that the advertising mechanisms, auction mechanisms, and bidding policies all function correctly as the distribution follows the expected pattern. These experiments have shown that the bid function is reflected in the task distribution. The random and constant bidding profiles produced near equal numbers of tasks allocated to each provider. The weighted models show task preference to the providers with lower cost weightings and therefore lower bids. If the delay between tasks was increased beyond the auction duration then the results would be less variable and would only differ due to the order in

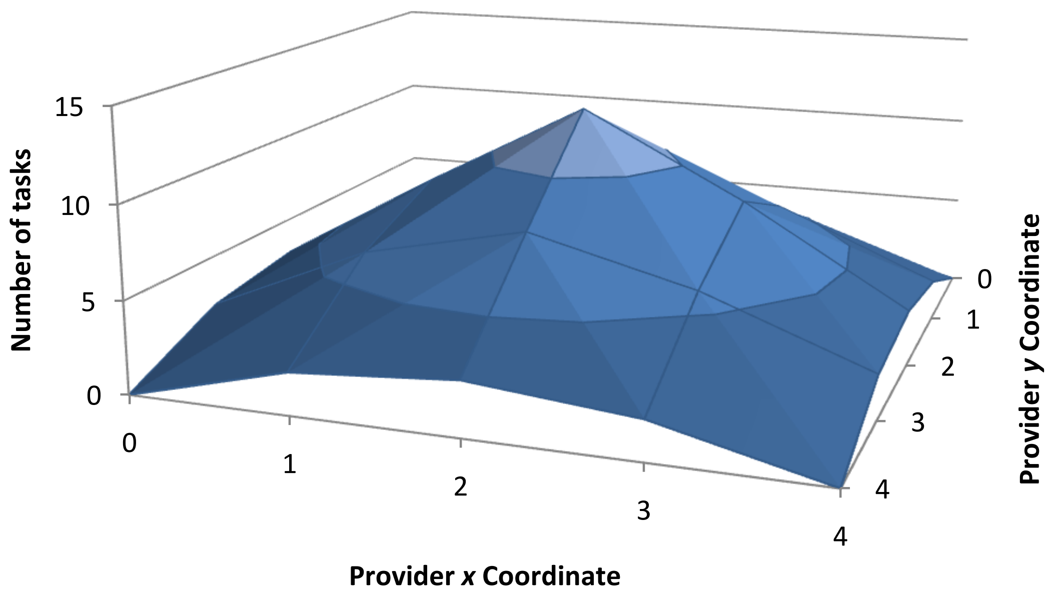


Figure 6.10: Average number of tasks allocated to each provider when using a Euclidean distance weighted model.

which bids arrive.

6.5 Grid Workload

Realistic Grid and/or Cloud workloads are required to analyse the effectiveness of resource allocation in DRIVE and also to study the various high performance strategies identified in Chapter 5. To simulate different resource provider loads three fine grained synthetic workloads and a batch workload have been derived from trace data obtained from a European production Grid. This section outlines various attributes of the original trace and the techniques used to create synthetic workloads aimed at evaluating the DRIVE allocation mechanisms.

6.5.1 Workload Characteristics

The synthetic workloads used in the following experiments are derived from workload traces generated by AuverGrid[2], a production Grid located in the Auvergne region of France. AuverGrid is part of the EGEE project and uses LCG (Large hadron collider Computing Grid) middleware on its 475 computational nodes organised into 5 clusters (with 112, 84, 186, 38, 55 nodes respectively). The major users of AuverGrid are from medical, bioinformatics and high energy

[2]`http://www.auvergrid.fr/`

physics. This trace was chosen as it was the most complete trace available in the Grid Workloads Archive [239]. While AuverGrid is a relatively small scale Grid the model obtained from the workload can be scaled up to be used in the analysis of DRIVE.

The full workload trace contains 404176 sequential jobs submitted over a 1 year period from January 2006 through to January 2007. To analyse this workload all incomplete jobs have been removed from the trace, resulting in a *processed trace* containing 304,992 jobs. Incomplete jobs are defined as jobs with missing information and jobs that did not run (run time of 0). There is no information in the trace regarding resubmission of jobs. The characteristics of the original and processed trace workloads are summarised in Table 6.5. In this table system utilisation is defined as the ratio between available CPU time and CPU time consumed, where the CPU time consumed is calculated as the sum of the consumed CPU time of each job. Job arrival rate defines the number of jobs arriving over a fixed period of time, in this case hourly. Interarrival time is the delay between individual jobs. Run time is the length of time a job is running. CPU utilisation is defined as the ratio of run time to CPU time for a single job.

	Original		Processed	
	Maximum	Average	Maximum	Average
Overall System Utilisation (%)	100	57.48	100	58.60
Job Arrival Rate (jobs/hour)	823	48.31	540	34.8
Job Interarrival Time (s)	196042	78.02	196042[3]	103.41
Job Run Time (s)	345707	21661.27	345707	27230.56
Job CPU Utilisation (%)	100	56.14	100	61.79

Table 6.5: Summary of original and processed workload characteristics.

The workload characteristics show that more than half the system capacity is consumed on average for the duration of each trace with peaks up to 100%. The small difference in average utilisation indicates the usage of the jobs removed in the processed trace have minimal computational requirements as expected. Jobs are generally long running with an average of 7.5 hours (27230s) per job in the processed trace, they are also mostly CPU bound with average CPU time consumed equating to more than half the length of time the job runs. The difference between average utilisation in each trace is due to the number of incomplete jobs removed which have 0% CPU usage. The average arrival rate is substantially lower than the maximum arrival rate which indicates a highly bursty nature. This is true of both the original trace (823 maximum, 48.31 average) and the processed trace (540 maximum, 34.8 average). Figure 6.11 shows a frequency graph of arrival rate for the original and processed trace. This graph shows that generally the arrival rate is below 100 jobs per hour. The processed trace has a higher peak as the removed incomplete jobs reduces the number of total jobs and therefore the number of jobs per hour. Figure 6.12 presents comparison of arrival rate for each of the two traces over time. In both traces the arrival rate is generally low with only a few periods of rapid arrival. The difference in maximum arrival rate

[3]The large maximum interarrival rate is probably due to an outage as 196042 equates to 54 hours between jobs. Examining the trace any jobs that arrive at the beginning of this time have a wait time of approximately the same length, indicating that the system may have failed or been down for maintenance and recovered those jobs when restarted.

during peak times between the two traces indicates that a lot of these jobs are incomplete, this is confirmed by examining the trace data as many of the jobs on the busiest days do not receive compute time due lack of capacity. The figure also highlights periods of inactivity in both traces (day 0 for 53 hours, day 112 for 13 hours, and day 200 for 54 hours), these periods may be due to system outage or maintenance. The outage on day 0 could potentially be due to setting up the Grid for the first time.

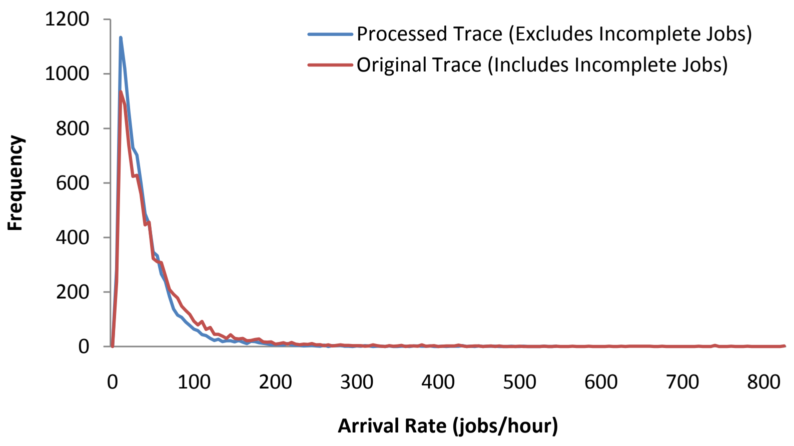
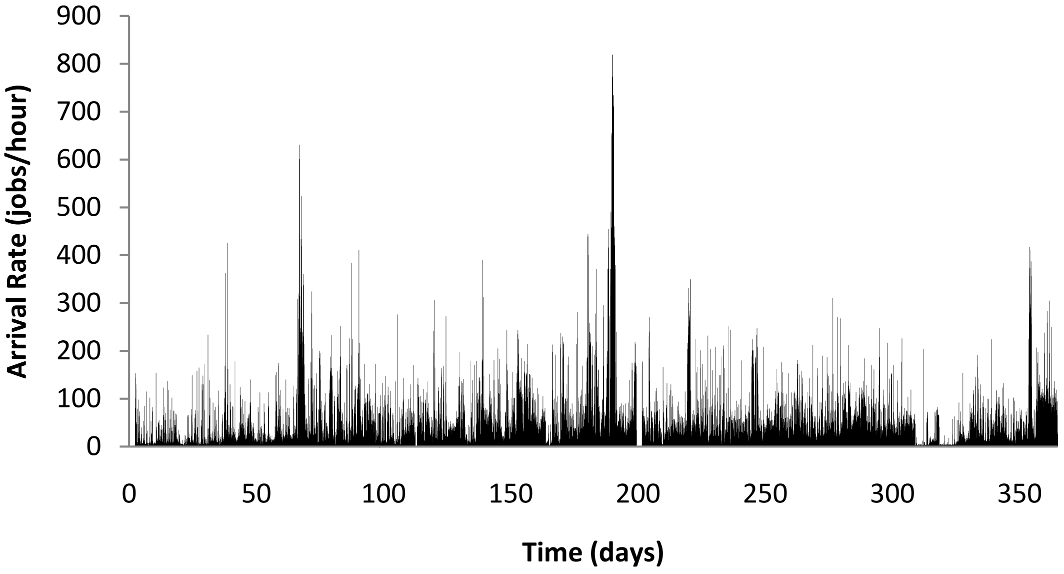


Figure 6.11: Arrival rate frequency for both the original and processed traces.

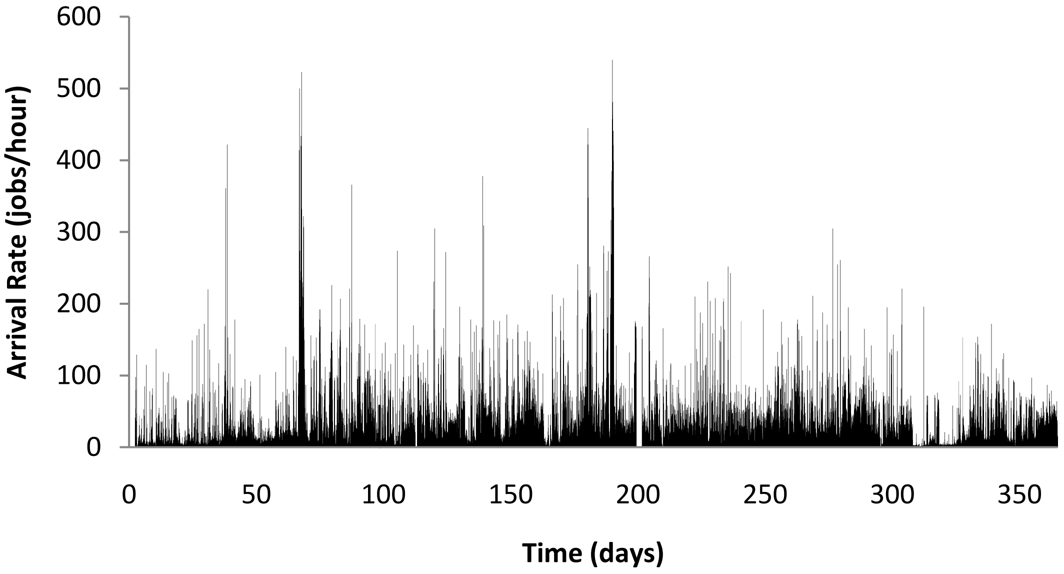
6.5.2 Characteristic Scatter Plots

Figures 6.13, 6.14, 6.15, 6.16,and 6.17 show scatter plots of CPU Utilisation, CPU Time, Memory Usage, and Run Time compared against one another. Each graph is based on the same 32,000 point random sample from the processed trace which excludes incomplete jobs.

Figures 6.13 and 6.14 show the relationship between CPU and memory usage by jobs. These graphs show there is an even spread and little tendency for jobs to be either computationally or memory intensive. Figure 6.13 appears to show that jobs with high memory usage consume little processor time, however Figure 6.14 shows that this does not mean that these jobs consume a small amount of processor, rather that the run time (and therefore CPU time) is short, this is also confirmed in Figure 6.15. It is interesting to note in all three of these graphs memory usage is low and there also appears to be definite memory bounds. These horizontal lines (m1, m2, m3) correspond to approximately 320, 600, and 768 MB. It is unclear if these bounds are due to the job type, job specifications or the physical hosts on which the job runs. To further clarify these limits Figure 6.18 shows the frequency of memory usage for the entire processed trace, again there are clear peaks at the these same values with an additional peak at approximately 50 MB that is less clear in the scatter plots.



(a) Job arrival rate (jobs/hour) for the original trace (includes incomplete jobs).



(b) Job arrival rate (jobs/hour) for the processed trace (excludes incomplete jobs).

Figure 6.12: Arrival Rate (jobs/hour) for the original and processed traces.

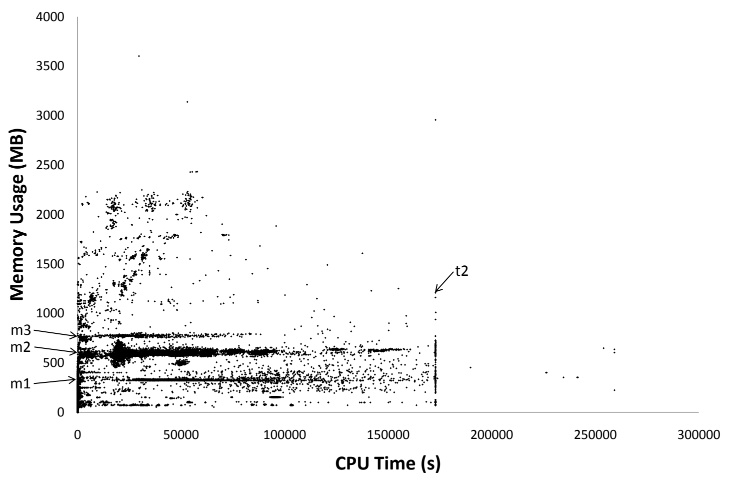


Figure 6.13: CPU Time vs Memory Usage.

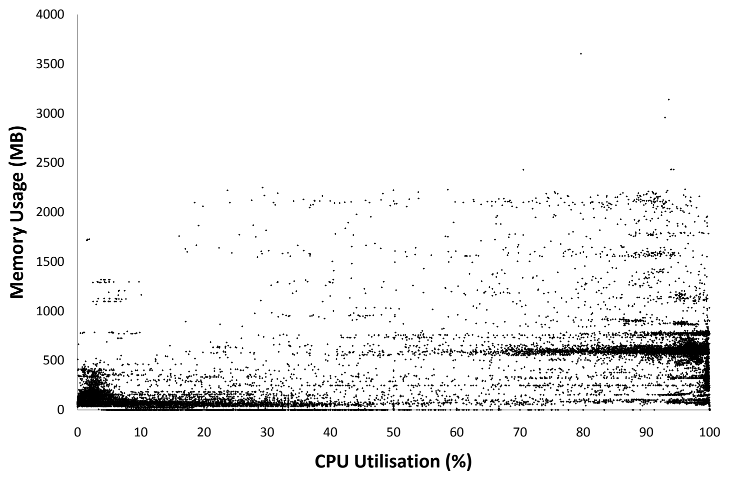


Figure 6.14: CPU Utilisation vs Memory Usage.

In terms of CPU usage Figure 6.16 shows that the CPU time used by a job is generally equal to the job run time, this is confirmed in Figure 6.17 with most jobs approaching 100% CPU usage, interestingly the variation in CPU utilisation is only evident in short running jobs which may indicate errors initialising job, accountancy errors (rounding), or a lot of very small tasks being submitted to the Grid.

Another interesting aspect of these graphs is the apparent limit on job run time. In each of the graphs comparing job run time (Figures 6.15, 6.16,and 6.17), there is a clear line after which

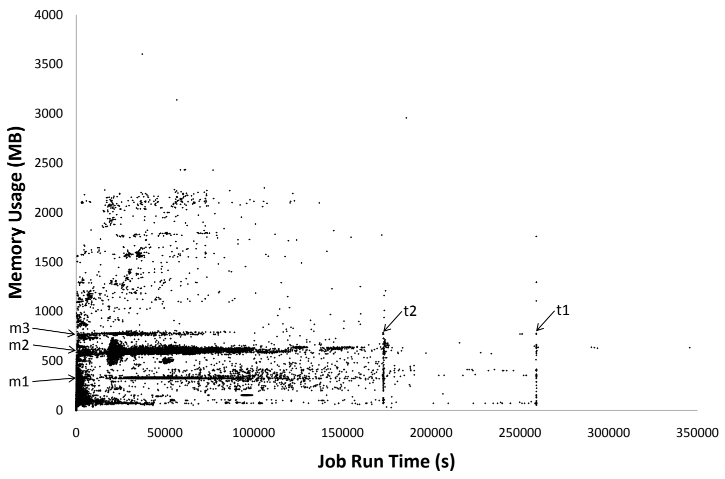


Figure 6.15: Run Time vs Memory Usage.

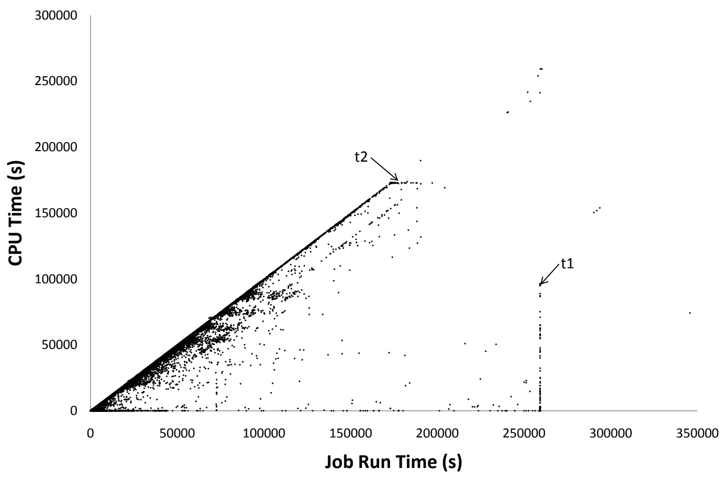


Figure 6.16: Run Time vs CPU Time.

very few jobs are still running. This limit (t1) is approximately 259,000 seconds (or 72 hours), as it cannot be seen in Figure 6.13 this limit is most likely placed on overall run time. Figure 6.15 shows a second limit (t2) at approximately 173,000 seconds (or 48 hours), this limit may be associated more closely with CPU time as it is clearly evident in Figures 6.13 and 6.16. Figure 6.17 also shows CPU utilisation drop off at (t1) proportional to the run time (this is because CPU utilisation is the ratio of CPU time to run time). These limits may be imposed by the system, scheduler, or application to ensure jobs do not run forever consuming resources. It is also possible these limits

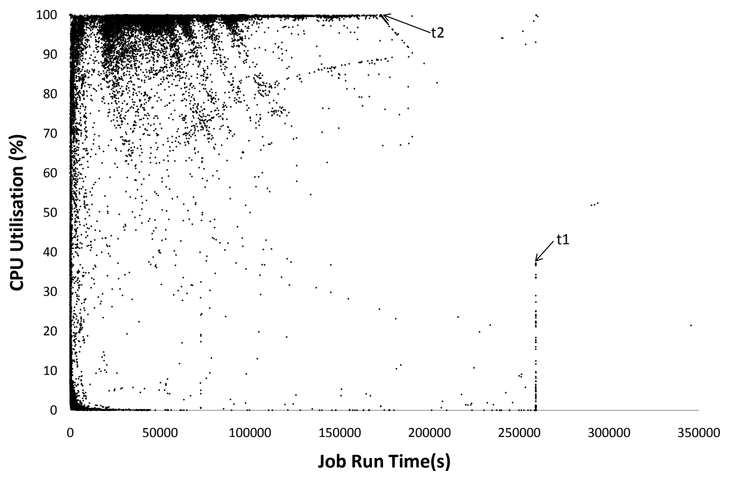


Figure 6.17: Run Time vs CPU Utilisation.

may be characteristic of common applications, however due to the discrete values (2 days and 3 days) it is unlikely. These two limits can also clearly be seen in the Job Duration Probability Density Functions presented in Section 6.5.3.

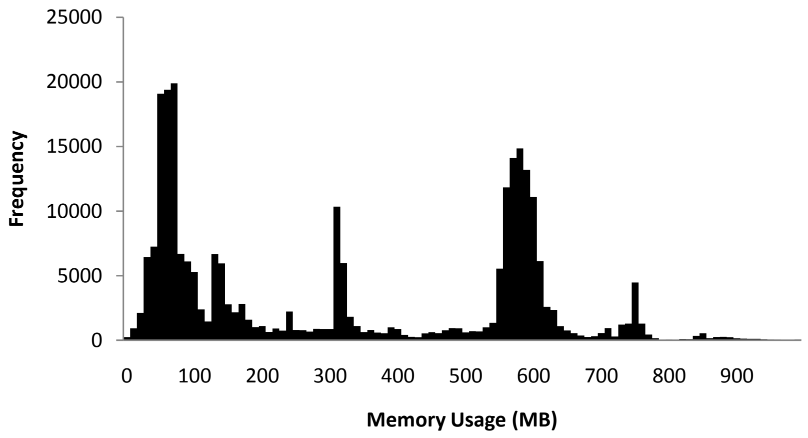


Figure 6.18: Memory usage frequency for the processed trace. The range of this graph includes jobs that use between 1MB-1000MB.

6.5.3 Probability Distribution Functions

Probability Density Functions (PDF) can also be used to characterise and model various aspects of the workload. PDFs describe the relative likelihood a random variable X occurs within a given space x. This section presents PDFs of the three workload characteristics used in the synthetic DRIVE workloads (Interarrival Time, Job Duration, and CPU utilisation).

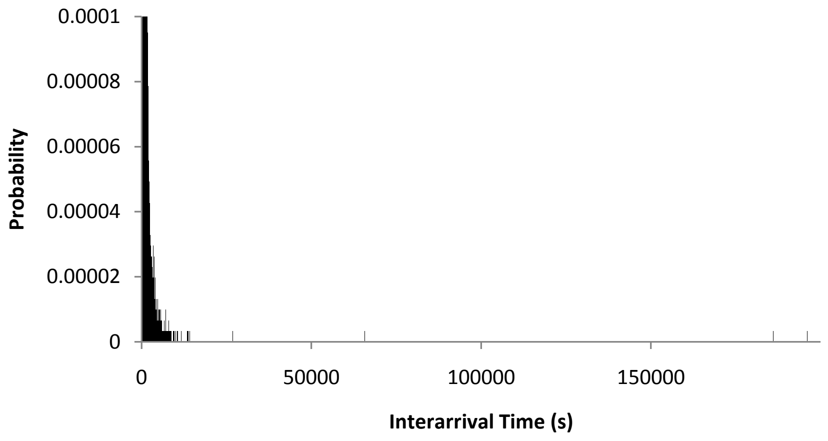
Interarrival time

Figure 6.19 shows PDFs describing job interarrival time for different probability and interarrival time ranges. The AuverGrid workload trace has a granularity of one second, therefore any jobs that arrive within one second are considered as arriving at the same time (interarrival time of 0). The probability of jobs arriving with no delay in this workload is 0.17 and the cumulative probability that jobs will arrive with an interarival time less than 10 seconds is 0.44. Figure 6.19(a) shows the interarrival PDF for the probability range [0,0.0001], the range is reduced as the large peak at 0 seconds dominates the entire graph on the full probability range [0,0.17]. This graph highlights the fact that the majority of jobs arrive within a short interval. Due to this observation Figure 6.19(b) further decreases the range of interest looking only at the interarrival times of jobs in the range [0,1000] seconds and probability [0,0.05]. This range includes 98.5% of all jobs with only 4692 (of 304,992) jobs having a larger interarrival time. This graph also shows an interesting pattern of interarrival times. The distribution appears to decay exponentially, however there are obvious peaks (t1,t2,t3,t4) every 60 seconds (60, 120, 180, 240). These peaks may occur due to particular scheduling metrics employed by schedulers or applications, for example schedulers periodically submitting tasks every minute. They could also be an artefact of Grid data gathering such as periodic performance monitoring jobs.

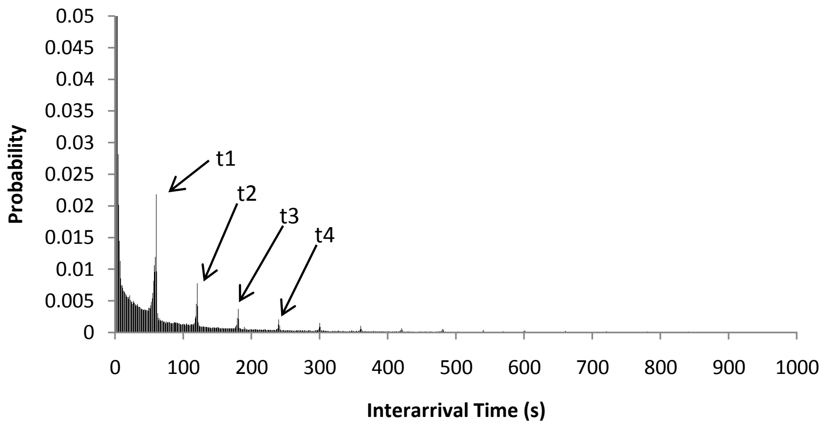
Due to these peaks it is impossible to model interarrival time with a single distribution. However both the general trend of interarrival, and the peaks could be modelled by separate distributions. An exponential distribution is a suitable model as it is a continuous distribution that is commonly used to describe the delay between events in a homogeneous poisson process (jobs). Grid jobs are normally modelled with a geometric distribution (such as a poisson distribution) as events are discrete and non-negative, rather than continuous over an infinite range. Jobs are also considered poisson distributed [240] as arrivals occur independently of one another and the number of arrivals in a given interval generally follow a poisson distribution. As interarrival rate is defined as the delay between Grid jobs (events in a poisson process) it is appropriate to model both interarrival time and the peaks observed every minute with individual exponential distributions.

Duration

Figure 6.20 shows PDFs for job duration. The graphs presented use constrained probability ranges as short duration jobs dominate the PDF with a full probability scale [0, 0.07]. In this trace the



(a) Interarrival Time PDF for probability range [0,0.0001].



(b) Interarrival Time PDF for interarrival time [0,1000] and probability [0,0.05].

Figure 6.19: Probability Density Functions for Job Interarrival Time.

probability that job duration is less than 10 seconds is 0.065, less than 100 seconds is 0.28 and less than 1000 seconds is 0.41. Recall the average job duration is 27230 seconds (Table 6.5), however there is a 0.65 probability that a job will run for less time, indicating a large spread above the mean. 95% of all jobs in the trace run for less than 100,000 seconds. Figure 6.20(a) shows the job duration PDF for the probability range [0,0.001]. This highlights the fact that most jobs are short running, as smaller durations have the highest probability. Two clear peaks can be seen at approximately 173000 (d1) and 256000 (d2) seconds which corresponds to 48 and 72 hours respectively, this is indicative of a run time or CPU time limit placed on a running job, reaffirming what was discussed in Section 6.5.2 (Characteristic Scatter Plots). Log transformations are often applied to data to reduce the effect of outliers and flatten data sets. Figure 6.20(b) shows the job duration PDF having applied a log transformation to the data. This graph shows two clear groupings, therefore job duration could be modelled with mixed bimodal log-normal distributions with means approximately 1.8 and 4.8. The overlapping section of the two log-normal distributions would therefore account for the third cluster at approximately 3.4.

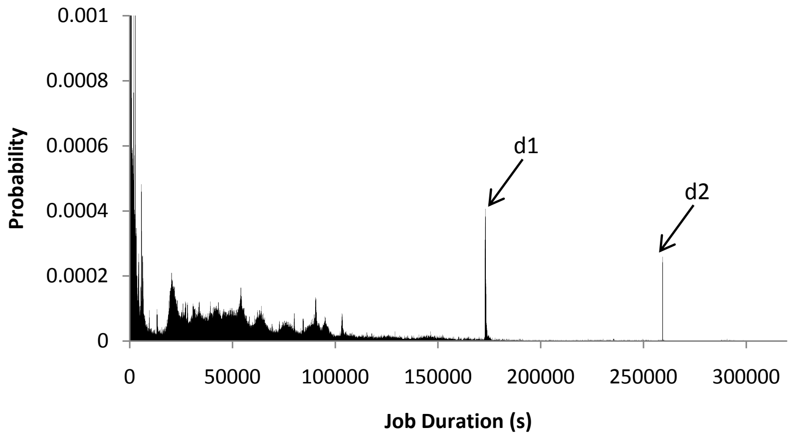
CPU Utilisation

PDFs for CPU utilisation are shown in Figure 6.21. Figure 6.21(a) shows that most jobs in this workload use close to 100% capacity, although there is a grouping towards 0 and also a small peak at 50%, these features are discussed later in this section. The probability of a job using 99% or more CPU is 0.27, indicating more than a quarter of all jobs consume the entire processing resources of a host. Figure 6.21(b) shows the same CPU utilisation PDF over the probability range [0,0.025], this highlights the grouping close to 0 and the observed peak at 50%. The peak at 50% is an accounting artifact of small jobs as run time and CPU time are rounded to the nearest second, 72% of these jobs (classified as 50% CPU Utilisation) run for only 2 seconds and use 1 second of CPU. This same rationale applies to the small peaks at (25%,33%,67% and 75%) as they represent common fractions. CPU utilisation is a ratio (CPU Time/Run Time), which places a limit on the range of values [0,100]. Therefore CPU utilisation cannot be modelled with a continuous distribution. A Bathtub curve is a more appropriate model to apply to this data. Bathtub curves are used in reliability modelling to describe the life time of a population of products. They are also commonly used to model human mortality, where the probability of both infant mortality and old age mortality are high whereas the period in between exhibits relatively low mortality.

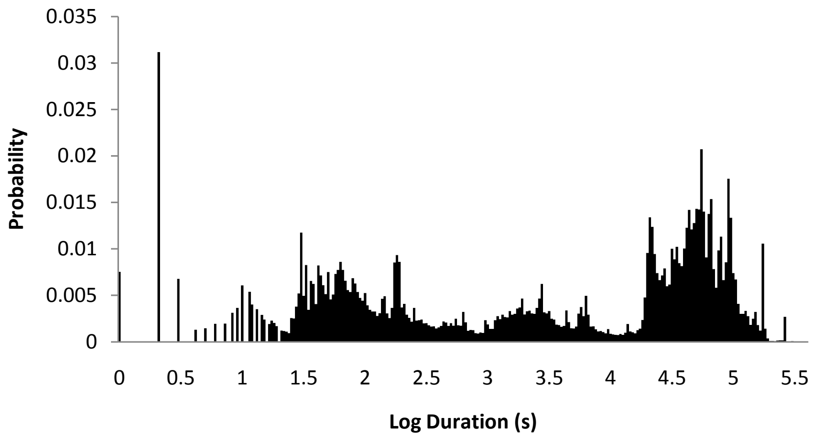
Cumulative Distribution Functions

The analysis of the AuverGrid workload presented on the GWA[4] suggests interarrival and job duration can be fitted with common distributions (Normal, Weibull, Gamma). These common distributions proposed are continuous and range from $(-\infty, \infty)$, however, the PDFs presented in Section 6.5.3 (Probability Distribution Functions) clearly show that interarrival rate and job duration cannot be modelled with a single continuous distribution. A continuous positive distribution

[4]`http://www.gwa.ewi.tudelft.nl/pmwiki/reports/gwa-t-4/trace\_analysis\_report.html`

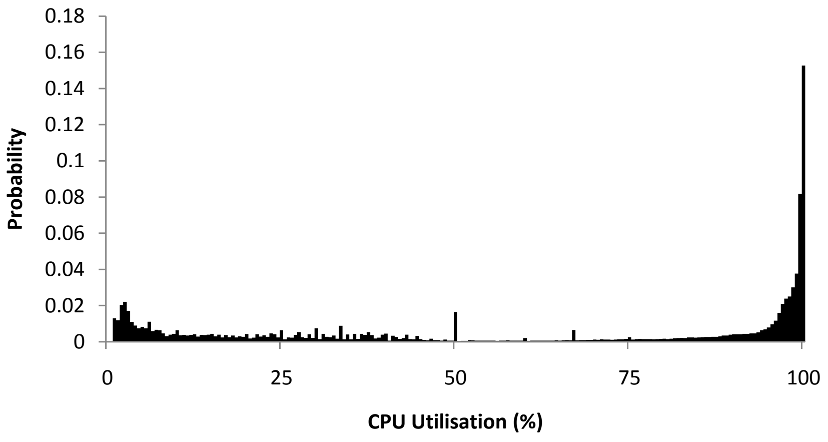


(a) Job Duration PDF for probability range [0,0.001] .

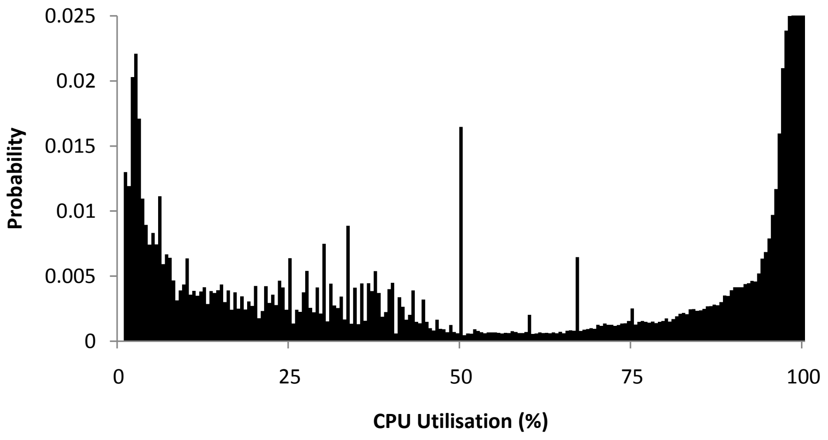


(b) Log Job Duration PDF.

Figure 6.20: Probability Density Functions for Job Duration.



(a) CPU Utilisation PDF.



(b) CPU Utilisation for probability range [0,0.025] .

Figure 6.21: Probability Density Functions for CPU Utilisation.

Parameter	Reference Distribution	KS Test d	KS p-value
InterArrival	Normal	0.085	< 0.0001
	Gamma	0.130	< 0.0001
	Weibull	0.145	< 0.0001
Duration	Normal	0.159	< 0.0001
	Gamma	0.164	< 0.0001
	Weibull	0.227	< 0.0001

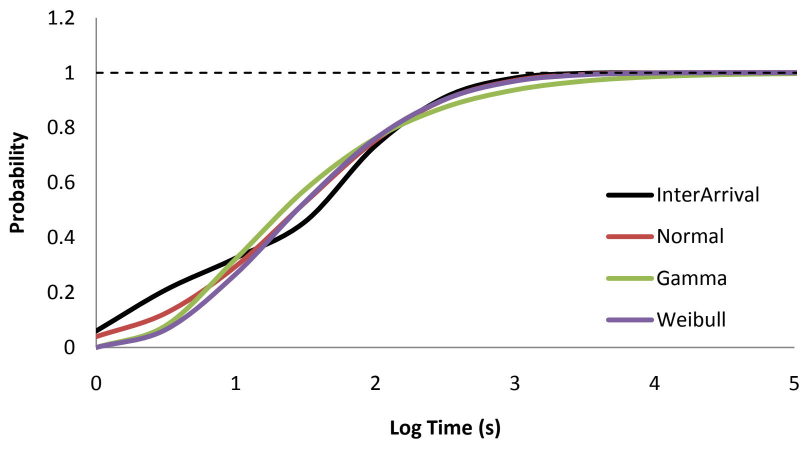
Table 6.6: Summary of Kolmogorov-Smirnov test results.

is more appropriate for interarrival time and a geometric (discrete) positive distribution is more appropriate for job duration. To further emphasise this point the Cumulative Distribution Functions (CDFs) for these characteristics have been fitted with common distributions in Figure 6.22 following the approach presented on the GWA. The reference distributions have been fitted using the Moments estimation method. Following the approach taken on the GWA a log transformation has also been applied to the data to minimise the effect of outliers.

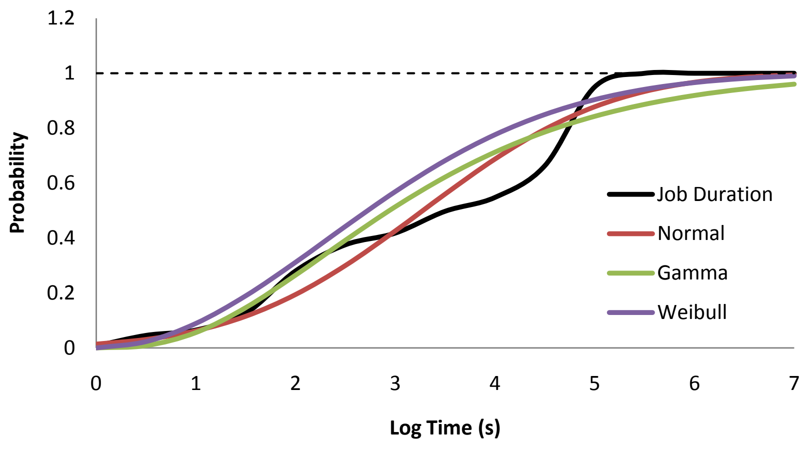
CDFs describe the probability distribution of a particular variable obtained by calculating the cumulative frequency. A CDF shows the probability that the variable X takes on a value less than or equal to x, that is, the proportion of the population whose value is less than x. The CDF of X is given by Equation 6.1.

$$x \mapsto \Phi_X(x) = P(X \leq x) \tag{6.1}$$

A graphical approach provides a simple way to determine the best fit, however is difficult to infer that the data follows a particular distribution. *Goodness-of-fit* tests are often used as a quantitative measure of how well a reference distribution models the sample data. The Kolmogorov-Smirnov test is one such test designed to compare a sample distribution to a reference distribution using a minimum distance estimation method. The results of the Kolmogorov-Smirnov test (Table 6.6) show the distance (d) that the sample differs from the fitted distribution and the probability (p-value) that the rejection of the hypothesis (that the data fits the given distribution) would be wrong. Generally a p-value less than 0.5 is used to determine if the hypothesis is rejected, a p-value greater than 0.5 indicates the hypothesis is not-rejected (however it does not infer acceptance). The p-values for this trace are very low due to the large size of the sample. While the samples appear closely modelled by the fitted reference distributions in Figure 6.22, the results of the KS test conclude that the hypotheses (that the sample follows any one of the proposed distributions) should be rejected due to large distances and very small p-values. This finding supports the conclusions made when studying the PDFs that neither job duration or interarrival time can be modelled with a single continuous distribution.



(a) Interarrival Rate.



(b) Job Duration.

Figure 6.22: Cumulative Distribution Functions of interarrival rate and job duration with fitted distributions.

6.5.4 Workload Summary

The workload analysis presented describes typical job characteristics on a small to medium scale Grid. In this trace arrival rate was shown to be bursty with peaks substantially more than average arrival time for both the original and processed traces. AuverGrid jobs are typically long running and computationally intensive. Analysis of PDFs showed that interarrival rate, job duration, and CPU utilisation could be be approximated by exponential, bimodal log-normal, and bathtub distributions respectively. Various memory and duration bounds were also identified in this analysis.

There is little information in the literature regarding analysis of real Grid or Cloud workload traces, most analysis is based on models, simulations, traces from non-production grids, or traces of parallel workloads. Real Grid workloads typically exhibit properties such as *Pseudo-periodicity*, *long range dependencies* and *bag-of-tasks* like behaviour [241, 242]. Pseudo-periodicity and long range dependency models can be applied to job arrival rate using stochastic point processes. Periodicity is common in arrival rates as indicated by peaks in arrival graphs corresponding to *seasons* (weekends, nights) and can be modelled with various techniques. Long range dependent characteristics are also seen in arrival rates and can be modelled using a Multifractal Wavelet Model (MWM) [243, 240]. Similarly Markov Modulated Poisson Processes (MMPP) can be applied to model short to medium term arrival rates [240], however parameter estimation is difficult. As workload analysis is not a focus of this thesis we do not explore these attributes in detail. Comprehensive Grid workload analysis is presented in [239, 242, 244], early analysis of Cloud workloads is also beginning to be published [245].

Using the entire AuverGrid workload as a basis for DRIVE experimentation is infeasible due to the duration (1 year) and cumulative utilisation (475 processors). To reduce the size, smaller traces can be generated by random sampling of the full trace to produce synthetic traces. Alternatively shorter activity periods could be chosen from the trace, but due to the low arrival long duration jobs only low performance is required from the allocation mechanisms. Synthetic workloads can be created by increasing the arrival rate and decreasing the job duration to create a high performance simulation.

6.5.5 Synthetic Workloads

The AuverGrid workload is characteristic of a long duration batch model, in which jobs arrive infrequently and are on average long running. There are two ways to use this data: first a sample can be taken over a fixed period of time to simulate a workload characterised by long slow batch processing, or secondly a synthetic high performance workload can be generated to reflect modern fine grained workloads[5] by increasing the throughput while maintaining the general workload model. Fine grained usage models can be used to represent modern (interactive) usage

[5]At the time this analysis was conducted fine grained workloads had not been published, however since this time, similar short duration workloads have been published and analysed, for example `http://dev.globus.org/wiki/File:Falkon-logs-history.jpg` – last accessed March 2011. These modern workloads support the synthetic workloads constructed for this thesis.

of distributed systems as seen in workflows and smaller scale adhoc personal use. These two interpretations of the data can be used to generate workloads at either end of the perceived use case spectrum.

The results presented in section 6.6 focus on fine grained high frequency synthetic traces as they present a worst case auction scenario (high frequency short duration jobs). Section 6.6.10 generalises these same experiments for batch submission using a sample period obtained from the original trace. This section describes the generation of high performance fine grained synthetic workload models and a long duration batch workload model.

Fine Grained High Performance Synthetic Workloads

To create high performance synthetic traces, the time based attributes of the original trace have been reduced by a factor of 1000 (that is a second of real time is 1 millisecond of simulated time). By reducing each parameter equally, relativity between parameters is maintained and therefore the distribution is not effected. This also has the effect of producing high frequency short duration jobs, which is the worst case situation for auction performance. The performance analysis presented in the following section looks specifically at the allocation performance without considering the duration of the jobs. However, this reduction in time-based characteristics effects the ratio of interarrival time to auction latency which exaggerates the effect of auction latency. In the reduced format the auction period is a much greater percentage of the entire task than in the original workload, this will result in providers bidding on more auctions within the period of a single auction than would be the case in the original batch model. In a production environment latency and processing time would be far greater than what is seen in these experiments, however this will not completely account for the increased ratio in the simulation. These synthetic workloads present a more fine grained, rather than long term batch model as shorter jobs are submitted more frequently.

Table 6.7 presents a summary of the three different synthetic workloads developed for this analysis. The synthetic workloads: low utilisation, medium utilisation, and high utilisation contain 2111, 4677, and 11014 jobs respectively, with each workload spread over almost 9 hours. The average job run time for each model is approximately 59 seconds with average job CPU utilisation of 93-94%. The workloads aim to represent increasing overal utilisation and are based on the utilisation limit of the testbed (in these experiments 20 providers are configured in the testbed). The low utilisation model has a peak of 86.55% overall utilisation which can be completely hosted in the testbed. The medium model slightly exceeds the capacity of the testbed with various points above the available capacity of 100%. The high utilisation model greatly exceeds the available capacity of the testbed with most values well above 100%. In each workload the average job CPU utilisation is higher than the original trace as failed and short running jobs (less than the minimum monitoring granularity of 5 seconds) have been removed. Short running jobs tend to have much lower CPU utilisation as was shown in Section 6.5.2. The arrival rate of tasks also increases with each workload as the number of jobs increases over the same period of time. The maximum

	Num Jobs	Average Arrival Rate (jobs/hour)	Maximum Arrival Rate (jobs/hour)	Average Run Time (ms)	Average Job CPU (%)	Maximum Total Utilisation (%)
Low	2111	234	357	58181	93.02	86.55
Medium	4677	519	814	59128	93.67	178.20
High	11014	1223	1892	59579	93.62	424.40

Table 6.7: Experiment Workload Characteristics.

arrival rate of the medium workload matches the maximum arrival rate of the original trace. The arrival rate for the high utilisation workload greatly exceeds the arrival rate of the original trace, with a maximum arrival rate more than double the peak arrival rate seen in the original trace. The number of tasks arriving each hour for each workload is shown in Figure 6.23.

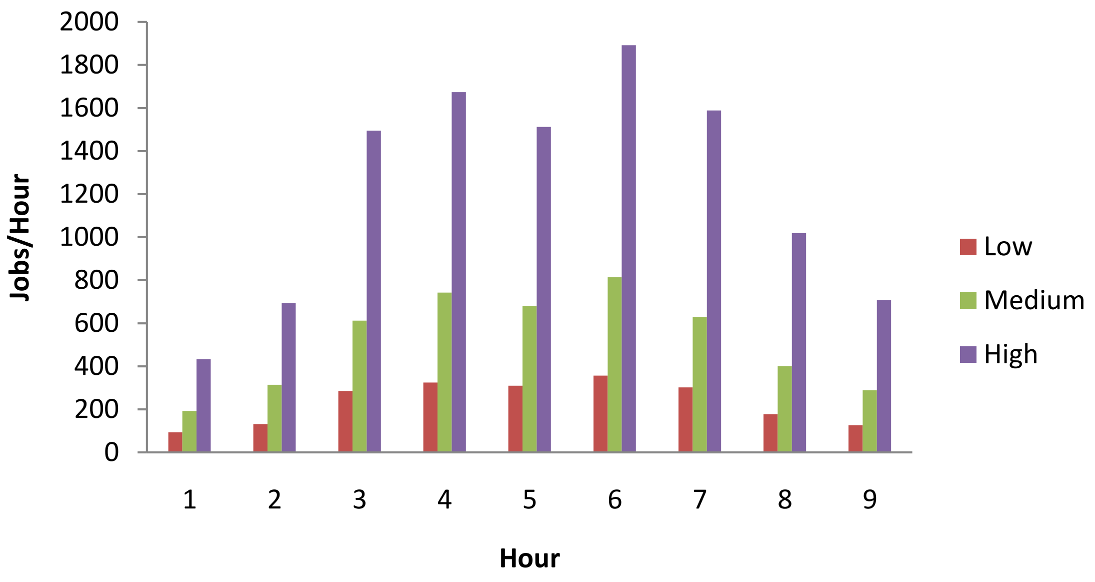


Figure 6.23: Average number of jobs arriving per hour for the high performance synthetic workloads.

Figure 6.24 shows an Exponential Moving Average (EMA) of the total combined utilisation of each of the synthetic workloads as jobs are submitted. In this graph 100% corresponds to the maximum capacity available on the testbed. Figure 6.25 shows the total system utilisation represented by each of the three traces over the experimental duration, these graphs show ideal utilisation which does not include any latency in allocation or maximum provider capacity limitations.

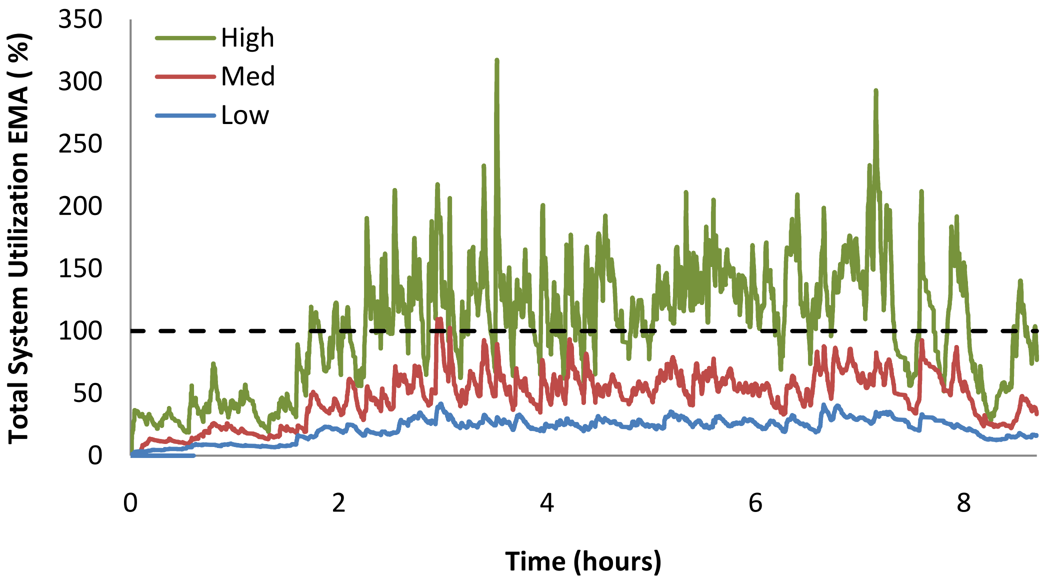
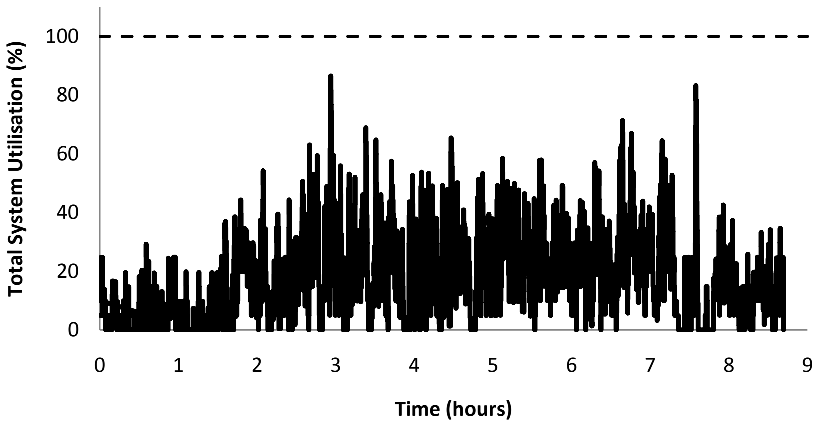


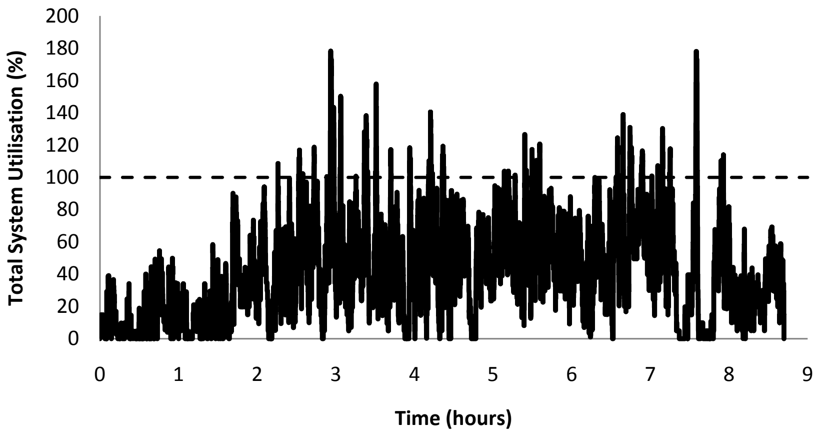
Figure 6.24: Total system utilisation of the three synthetic workloads shown using an Exponential Moving Average (EMA) with 0.99 smoothing factor. The dashed line indicates the total system capacity of the testbed.

Traditional Batch Workload

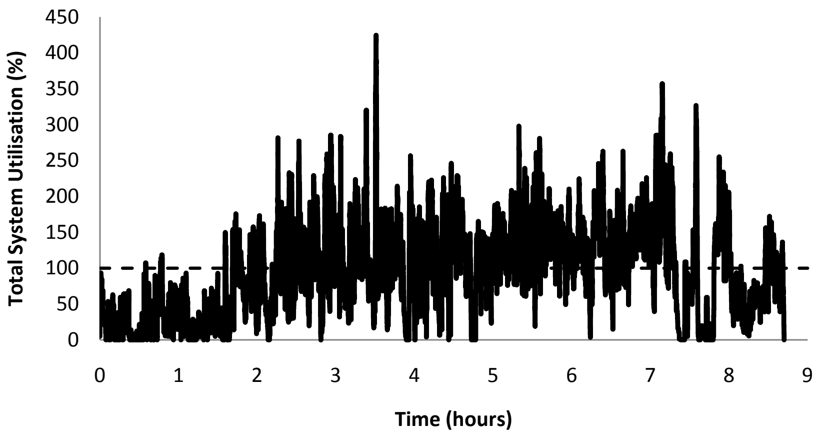
The batch workload is generated from a two day sample of the complete AuverGrid trace. The two days chosen include the busiest day (number of jobs) in the trace. A summary of the workload characteristics are presented in Table 6.8. The total number of jobs over the two day sample is slightly less than the high utilisation synthetic workload (over a period of 9 hours). The arrival rate is similar to the low synthetic workload with 211 jobs per hour submitted on average. Arrival rate over the duration of the trace is shown in Figure 6.26(a). Job duration is on average over 1 hour, and jobs use approximately 80% of a single processor. Due to the nature of the trace this workload represents much greater system requirements as AuverGrid has 475 machines, the two day workload peaks at 32709% utilisation which is equivalent to 327 machines completely utilised. Reducing the sample size such that this workload can be hosted on the 20 machine testbed is impossible as the resulting number of jobs would be minimal (approximatly 600 jobs over 48 hours). Instead experiments using the batch workload utilise an increased testbed capacity by simulating "larger" providers. In these experiments a single provider has the equvialent of 15 processors. Maximum system capacity is therefore 30000%. The batch workload and the capacity of the testbed is shown in Figure 6.26(b).



(a) Low Workload.



(b) Medium Workload.

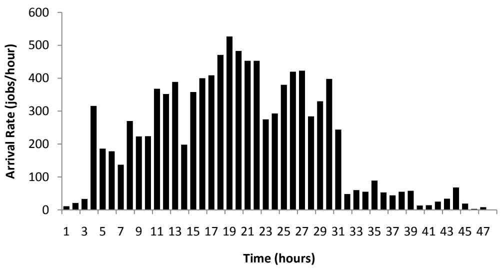


(c) High Workload.

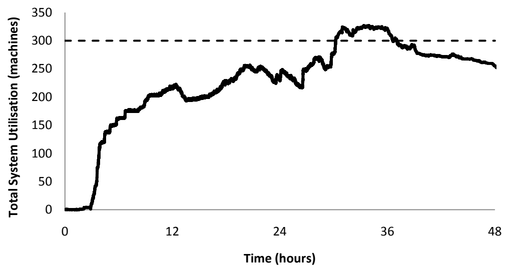
Figure 6.25: Total system utilisation for the high performance synthetic workloads showing the maximum capacity of the testbed.

	Num Jobs	Average Arrival Rate (jobs/hour)	Maximum Arrival Rate (jobs/hour)	Average Run Time (s)	Average Job CPU (%)	Maximum Total Utilisation (%)
Batch	10153	211.52	527	5152.73	82.40	32709

Table 6.8: Two day batch workload characteristics.



(a) Arrival rate for the two day batch workload.



(b) Cumulative utilisation over time for the two day workload sample.

Figure 6.26: Two day batch workload sample.

Summary

These synthetic workloads provide a means of simulating Grid usage ranging from a traditional batch model through to a more modern fine grained model. Each of the traces is derived from production Grid data to ensure validity when analysing DRIVE. The fine grained traces typify modern adhoc Grid usage, each workload maintains the important characteristics of the original trace by providing similar or greater maximum throughput whilst maintaining the duration of jobs relative to arrival time and one another. The fine grained workloads also model an interactive system with high frequency short duration jobs which presents a worst case scenario for auction based allocation. The batch workload on the other hand is characterised by infrequent job submission and long duration computationally intensive jobs which represents more favourable conditions for auction based allocation.

6.6 High Utilisation Strategies

Section 5.7 proposed several high performance strategies which may be used to increase occupancy and utilisation in auction based allocation. The strategies presented in this section are evaluated with respect to the number of auctions completed, contracts created and overall system utilisation. The strategies each define the criteria by which providers choose to bid, they are denoted: Overbidding (O), Second chance substitutes (S), and advanced Reservations (R). In addition a Guaranteed (G) strategy is also implemented against which to compare the high util-

	Bid Decision	Auction Actions	Contract Actions
G	Potential utilisation	bid/don't bid	accept
O	Current utilisation	bid/don't bid	accept/reject
S + O	Current utilisation	bid/don't bid	accept/substitute/reject
R + O	Projected utilisation	bid (res)/don't bid	accept (res)/reject
R + S + O	Projected utilisation	bid (res)/don't bid	accept (res)/substitute/reject

Table 6.9: Strategy summary for each experiment (res indicates reservation windows are taken into account).

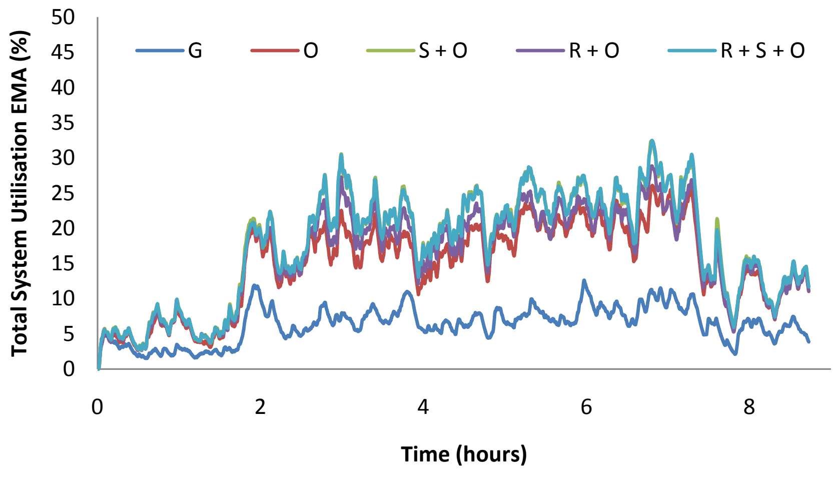
isation strategies. The different experiments presented examine the effect of different strategy combinations on allocation.

In the following experiments a sealed bid second price protocol is used to allocate tasks, each provider implements a random bidding policy irrespective of job requirements or current capacity, the bid range is 1 to 100. Contracts are accepted only if their is sufficient capacity regardless of what was bid. Table 6.9 summarises the experiments presented in this section, each experiment differs based on the bid decision process and what actions can be taken at the auction and contract stages. In the Guaranteed strategy providers bid based on potential utilisation (if all outstanding bids win their respective auctions). Providers will only bid if potential utilisation is less than their capacity. Contracts are never rejected as providers never bid beyond capacity. For the Overbidding strategy providers bid based on current utilisation (irrespective of outstanding bids), as providers can bid beyond capacity they may choose to accept or reject contracts depending on current capacity, if a contract is rejected the task is never reallocated. In the Substitute strategy providers bid based on current utilisation, however winning providers can be substituted at the contract stage if they do not have sufficient capacity. In the Reservation strategy, providers bid based on projected utilisation allowing for the ability to schedule tasks according to the defined reservation window, likewise contracts can be accepted based on the reservation window defined by the task.

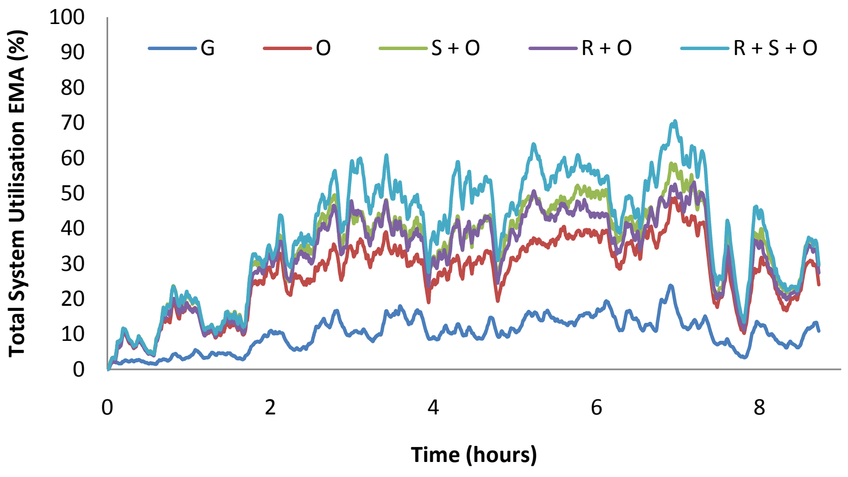
Figures 6.27(a), 6.27(b) and 6.27(c) show an Exponential Moving Average (EMA) of total system utilisation for each of the experiments on the low, medium and high workloads respectively. These figures are used to compare the strategies against one another throughout this section, individual system utilisation for each strategy and workload is also presented in the respective section.

6.6.1 Guaranteed Strategy

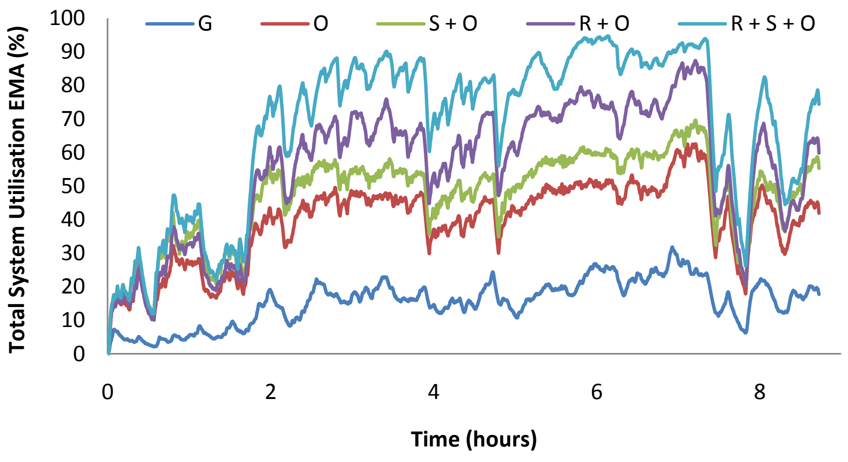
In the baseline configuration providers implement a guaranteed strategy where every bid by a provider is a guarantee that there will be sufficient capacity. In this strategy providers include contracts and any potential future contracts (bids) when calculating their potential utilisation. The provider will only bid if the potential utilisation is less than maximum capacity. This is clearly a naive bidding strategy and is only included to evaluate the proposed high utilisation strategies.



(a) Low Workload.



(b) Medium Workload.



(c) High Workload.

Figure 6.27: Total system utilisation for each workload and each strategy shown using an Exponential Moving Average with 0.99 smoothing factor.

Figure 6.28 shows the resulting total utilisation over the group of providers for each of the three workloads. It is evident in each of the graphs that the overall utilisation is extremely low, well below the available capacity of the testbed. Average utilisation is 6.35%, 10.5% and 15.72%, and the maximum utilisation is 29.65%, 43.65%, and 53.80% for each of the low, medium and high workload respectively. In all three workloads the rejection rate is substantial with only 34.41%, 26.75%, and 16.96% of tasks allocated for each workload respectively. As expected these rejections occur during auctioning and no contracts are rejected, as no provider should ever bid outside its means.

These results highlight the issue with bidding only on auctions a provider can guarantee to satisfy. A large number of tasks are rejected even though there is sufficient available overall capacity. The reason for this is the number of concurrent auctions taking place and the latency between submitting a bid and the auction concluding. In this testbed there is a $\frac{1}{20}^{th}$ chance of winning an auction (when all providers bid) so for the duration of the auction all providers have reserved their resources with only a probability of 0.05 of actually winning. The advantage of a guaranteed approach is no auctions are won without available capacity, ensuring no contracts are ever rejected. Additionally the burden on the allocation infrastructure is reduced as no attempts are made to establish contracts which cannot be honoured.

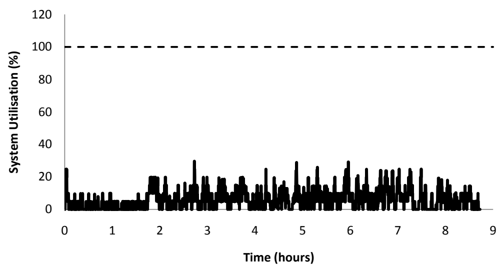
In this strategy the key factor to consider is the ratio of auction latency to interarrival time. If the auction latency was reduced or the frequency of tasks being submitted was reduced, the number of tasks allocated and total utilisation would improve as bidders have a more absolute view of utilisation when bidding on subsequent auctions.

The low allocation rate of the Guaranteed strategy motivates the need for providers to bid beyond capacity. Overbidding can be used to increase system utilisation in the knowledge that only one of the bidders will effectively win an auction.

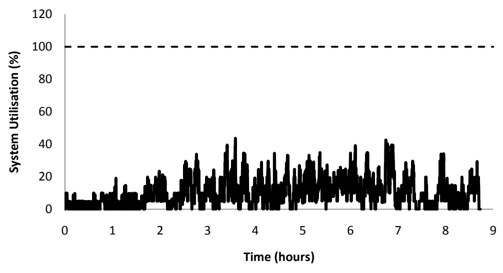
6.6.2 Overbidding Strategy

As discussed in Section 5.7.1 the potential benefits of bidding beyond provider capacity can result in increased utilisation and profit, assuming the penalties for breaking contracts are sufficiently small. Using an overbidding strategy providers compute their utilisation considering only established contracts and therefore choose to bid irrespective of any outstanding bids. Figure 6.29 shows the increased overall utilisation for each workload when overbidding. The substantial improvement over the guaranteed approach can be seen in Figure 6.27 for each workload considered. In the high workload maximum system utilisation approaches the maximum testbed capacity at points. Average utilisation (15.03%, 27.13%, 39.99%) and allocation rate (84.54%, 67.13%, 42.39%) for all three workloads is more than double that of the guaranteed strategy highlighting the value of overbidding. Although this represents a significant improvement in performance there is still substantial underutilisation of the system when viewed across all providers.

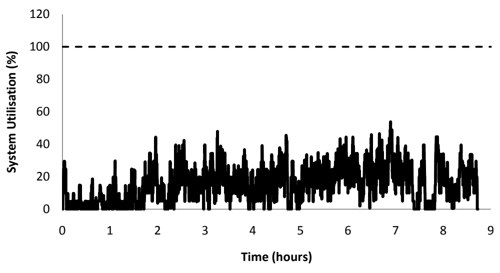
In each workload very few auctions fail (for example only 42/11014 in the high workload) as providers only reach maximum capacity for a short period of time. The major limitation of over-



(a) Low Workload.



(b) Medium Workload.



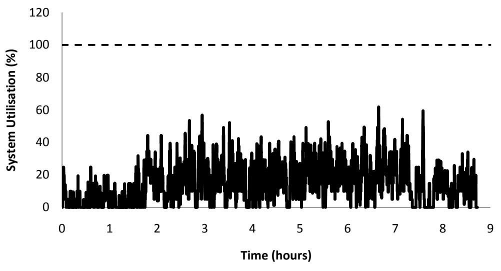
(c) High Workload.

Figure 6.28: Total system utilisation using a guaranteed strategy.

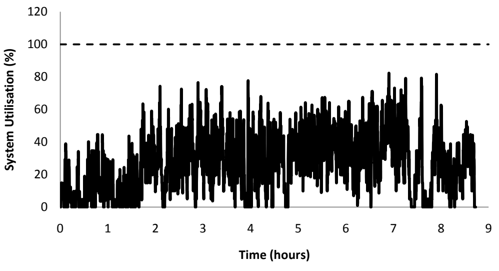
bidding is the number of auctions completed that are then unable to be converted into contracts, this approaches 60% of all tasks in the high workload, 33% in the medium workload, and 15% in the low workload. In the high workload if penalities were approximately equal to rewards then the system would operate at a loss as the number of contracts rejected exceeds the number of contracts confirmed. However, as these penalities are assumed to be low, there is potential for increased profit due to the increased occupancy generated by overbidding. This experiment represents a naive overbidding strategy in which there is no limit on the degree of overbidding. Providers could implement limited overbidding policies based on previously observed win rate to optimise the liklihood of being able to satisfy their bids. The number of contracts unable to be established when overbidding effects system performance as the auction and contract negotiaton processes are wasted, this can be considered pure overhead. One way to optimise this process is to reuse the existing auction rather than re-executing the whole auction process, as is evaluated in the next section.

6.6.3 Second Chance Substitutes and Overbidding

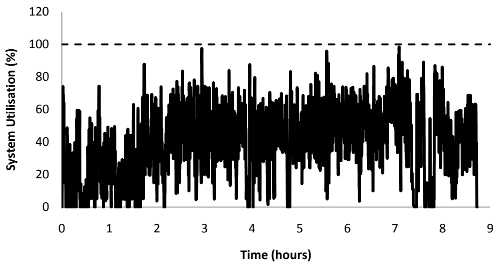
In the event of a rejected contract, a losing bidder can be offered a second chance to seamlessly satisfy the auction. Allowing second chance substitutes reduces the contract failures caused by overbooking resources and therefore increases utilisation of the system. The limitation when using second chance providers is the potential for consumer price increases as the new provider



(a) Low Workload.



(b) Medium Workload.



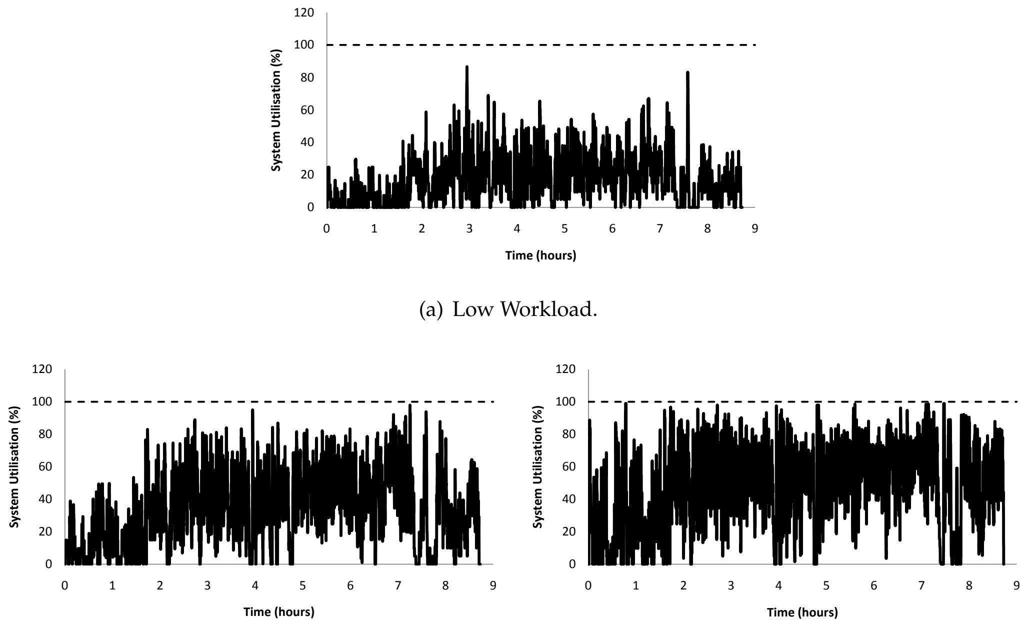
(c) High Workload.

Figure 6.29: Total system utilisation using an overbidding strategy.

	S + O Average Depth	R + S + O Average Depth
Low	1.46	1.23
Medium	2.29	2.04
High	2.55	2.76

Table 6.10: Average number of substitute providers considered.

is no longer the lowest bid, however this increase can be offset by the penalties imposed on the original provider for breaking the tentative agreement. Figure 6.30 shows the resulting utilisation when using second chance providers. Average utilisation is shown to be improved from the overbooking strategy by up to 26% (low - 17%, med - 26%, high - 20%) and task allocation is increased to 99.94%, 85.27%, and 51.17% for the low, medium, and high workloads respectively. These results show a large improvement from the previous strategies. Interestingly the average depth of substitute providers consulted is less than 3 for each workload (Table 6.10) indicating that there is sufficient reserve capacity in the system. This information could be used to define policies regarding the maximum number of substitute providers able to be consulted.



(a) Low Workload.

(b) Medium Workload.

(c) High Workload.

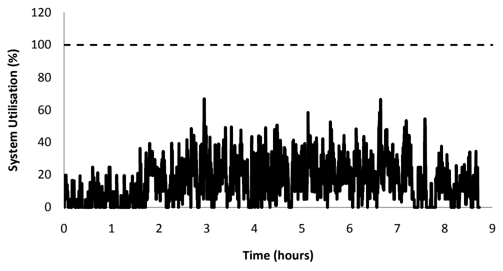
Figure 6.30: Total system utilisation using second chance substitutes.

6.6.4 Reservations and Overbooking

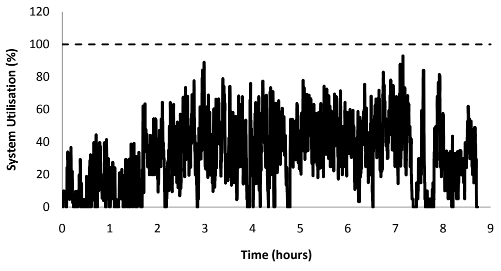
Reservations are widely used in large scale distributed and parallel systems as a means of coordinating resource usage. They have have also been shown to increase performance and provide flexibility [178]. In the AuverGrid workload trace there is no explicit execution windows defined, so in order to evaluate this strategy and analyse the effect on allocation and utilisation we define an execution window for each task as 50% of the task duration. This provides a reasonable approximation of a reservation window allowing providers room to schedule tasks (sensitivity analysis is shown in Section 6.6.9). In this experiment providers implement a simple First Come First Served (FCFS) scheduling policy. There are better techniques for scheduling reservations, such as backfilling [178], however, FCFS is sufficient to provide comparability for these experiments.

Each provider again uses an overbidding strategy in this experiment due to the considerable utilisation improvements seen over guaranteed bidding. Each task is submitted with a flexible reservation window, defining start time, end time and duration. The end time has been extended by 50% of the duration.

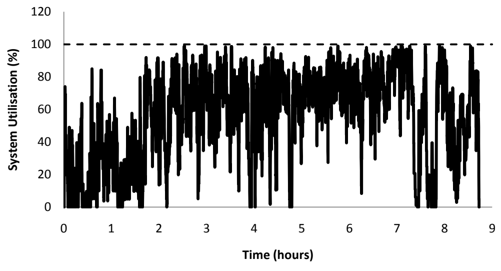
Figure 6.31 shows the system utilisation for each workload using flexible reservations. As the density of the workload increases the improvement gained by using reservations is greater than that of computing substitutes (Figure 6.27). For the low and medium workloads the allocation rate (90.32% and 78.34%) is less than that of using substitutes, however in the dense high workload the



(a) Low Workload.



(b) Medium Workload.



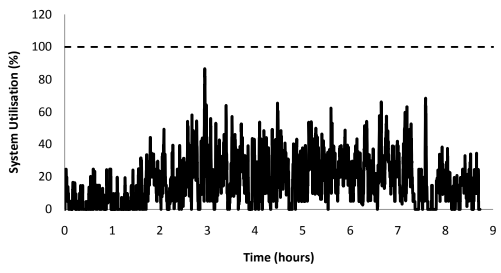
(c) High Workload.

Figure 6.31: Total system utilisation using flexible advanced reservations.

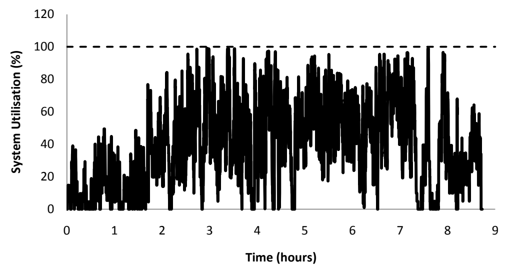
improvement exceeds the gains made using substitutes peaking at 55.75% of all tasks allocated. The same pattern is seen in average utilisation as it is generally proportional to the number of tasks allocated. The improvement is clearly evident in the EMA graphs presented in Figure 6.27, for the low and medium workload both the substititue and reservation strategies are similar, however in the high workload the system utilisation using reservations is substantially improved.

6.6.5 Reservations, Second Chance Substitutes, and Overbidding

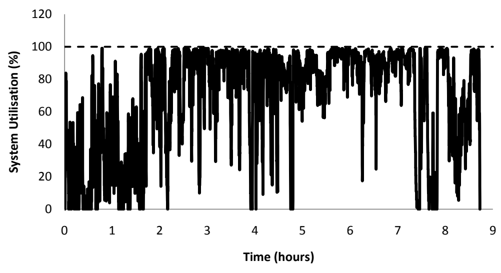
The final configuration combines the use of reservations with the ability to compute substitute providers and overbook resources. As expected this combination gives the best results for each of the workloads, with 100% of the low workload tasks allocated, 98.38% of the medium workload allocated and 65.54% of the high workload allocated. In the low and medium workload no auctions fail as providers are not fully utilised for long periods of time, on average only 75.67 contracts are rejected in the medium workload due to short periods of time when providers are fully utilised. Due to the nature of the high workload which consistently exceeds the capacity of the test bed, 65% of tasks allocated represents very high degree of workload allocation, close to the maximum obtainable allocation. This is also reflected in the average system utilisation of 68%, which given the constraints of the workload represents high overall utilisation. Figure 6.32(c) highlights the near capacity utilisation of the system, with a tendency for system utilisation to be fully used for long periods of time. In each workload the combination of all three strategies



(a) Low Workload.



(b) Medium Workload.



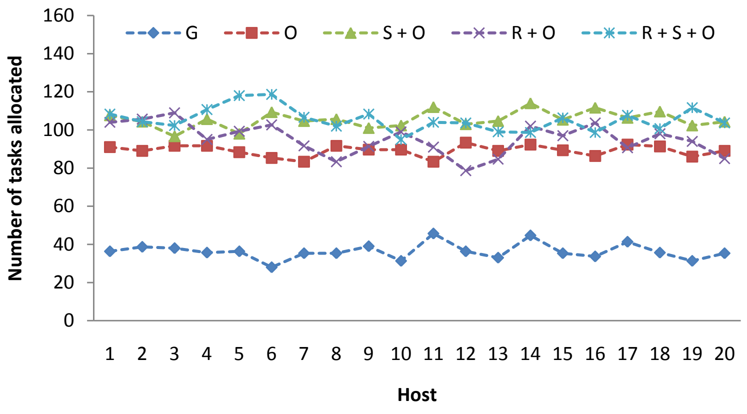
(c) High Workload.

Figure 6.32: Total system utilisation using reservations, substitutes, and overbidding strategies.

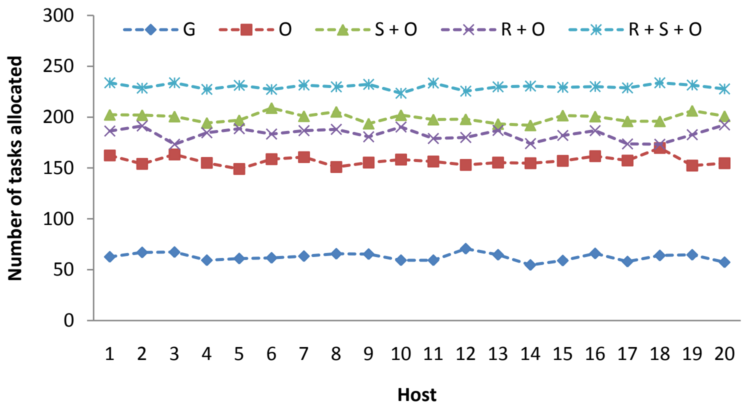
results in the highest number of allocated tasks and the highest system utilisation which validates the use of the proposed high utilisation strategies.

6.6.6 Provider Allocation

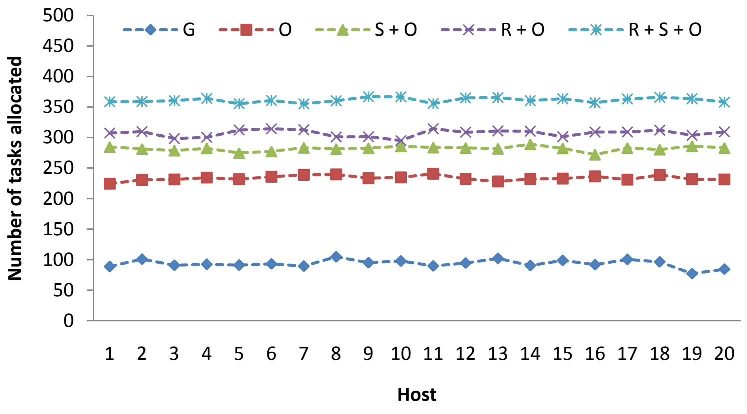
The previous experiments focused on global allocation and total system utilisation without considering the benefits obtained by individual providers. As providers are assumed to be self interested and competitive there is no incentive to implement or utilise high performance strategies without individual gain. The number of tasks allocated to each host for a single run of each experiment is shown in Figure 6.33, the hosts have been arbitrarily numbered 1-20 to visualise the data. These graphs clearly demonstrate the advantage to individual providers of implementing each of the high utilisation strategies. In both the medium and high workloads the advantages between each strategy is seen for every provider. Due to the small number of jobs submitted in the low workload the difference between strategies is less clear. As expected the tasks are evenly distributed across the providers in each workload due to the random bidding policies used. The high workload shows the most even distribution due to the increased sample size.



(a) Low Workload.



(b) Medium Workload.



(c) High Workload.

Figure 6.33: Allocation distribution for each strategy showing the number of tasks hosted by each provider. The points are joined to highlight the allocation difference between strategies.

6.6.7 High Utilisation Strategy Summary

Table 6.11 summarises the performance of the proposed high utilisation strategies when applied to each of the synthetic workloads. In particular the table outlines the number of auctions completed, contracts created, and overall system utilisation. Each of the strategies has been shown to increase occupancy and utilisation in the DRIVE testbed deployment.

The baseline guaranteed strategy provides a reference from which to evaluate the other approaches with maximum allocation of 34.41%, 26.45%, and 16.96% for each of the workloads. The poor allocation rate is also reflected in the low average system utilisation. Overbidding provides the single greatest increase in allocation and utilisation, however the increased contract rejection rate may be viewed as pure overhead due to the wasted negotiation process. Contract rejection when overbidding is 15.46%, 32.87% and 57.22% of all tasks for each workload. The use of second chance substitute providers greatly increased the overall allocation rate by up to 27% (medium workload) therefore reducing allocation overhead.

The additional flexibility obtained when using reservations was shown to be advantageous, especially in the most dense high workload. In fact, in the high workload the increased allocation rate obtained through reservations (31%) outperformed that of the substitute strategy (20%). Finally the use of all four strategies simultaneously was shown to provide the highest allocation rate (100%, 98.38%, 65.54%) and system utilisation (17.72%, 39.34%, 68.91%) over the testbed. This equates to substantial allocation improvement over a guaranteed approach of 190.64%, 267.81% and 286.44% for each of the workloads considered. The results presented in this section have demonstrated the value of employing such strategies for both consumers and providers. The economic implications of these strategies are discussed in Section 6.7

6.6.8 Just-In-Time Bidding

In each of the previous experiments (except guaranteed bidding) the number of contracts rejected by providers is significant. This occurs as provider state changes between bidding and auction completion such that required resources are unavailable. This section examines the effect of auction latency on allocation before presenting the performance improvements gained by JIT bidding for each of the high utilisation strategies.

Auction Latency

A series of symmetric workloads are used to examine the effect of auction latency on allocation rate. In each experiment the workload is allocated over a 10 host virtualised testbed. Table 6.12 details the workload characteristics. Each workload has increasing individual job resource requirements and therefore increasing maximum system utilisation. The total system utilisation for each workload is shown in Figure 6.34(a). In each workload, tasks are submitted every 5 seconds with a duration of 2 minutes each. Individual task utilisation ranges from 20% of a single providers capacity allowing 5 tasks to be hosted simultaneously on a single provider, through to

		Auctions		Contracts			Avg Sys	Max Sys
	Tasks	Failed	Complete	Rejected	Substitutes	Allocated	Util %	Util %
Low								
G	2111	1384.67	726.33	0	N/A	726.33 (34.41%)	6.35	29.65
O	2111	0	2111	326.33	N/A	1784.67 (84.54%)	15.03	61.90
S + O	2111	0	2111	1.33	375.67	2109.67 (99.94%)	17.71	86.55
R + O	2111	0	2111	204.33	N/A	1906.67 (90.32%)	16.18	66.80
R + S + O	2111	0	2111	0	225.33	2111 (100%)	17.72	86.55
Medium								
G	4677	3426	1251	0	N/A	1251 (26.75%)	10.50	43.65
O	4677	0	4677	1537.33	N/A	3139.67 (67.13%)	27.13	82.25
S + O	4677	14	4663	675	1489	3988 (85.27%)	34.28	97.90
R + O	4677	0	4677	1013	N/A	3664 (78.34%)	32.26	92.90
R + S + O	4677	0	4677	75.67	1146.67	4601.33 (98.38%)	39.43	99.05
High								
G	11014	9146	1868	0	N/A	1868 (16.96%)	15.72	53.80
O	11014	42	10972	6303	N/A	4669 (42.39%)	39.99	98.35
S + O	11014	325.33	10688.67	5052.67	2463.33	5636 (51.17%)	48.09	99.00
R + O	11014	30.33	10983.67	4843	N/A	6140.67 (55.75%)	56.67	99.00
R + S + O	11014	419.67	10594.33	3375.67	3091	7218.67 (65.54%)	68.91	99.05

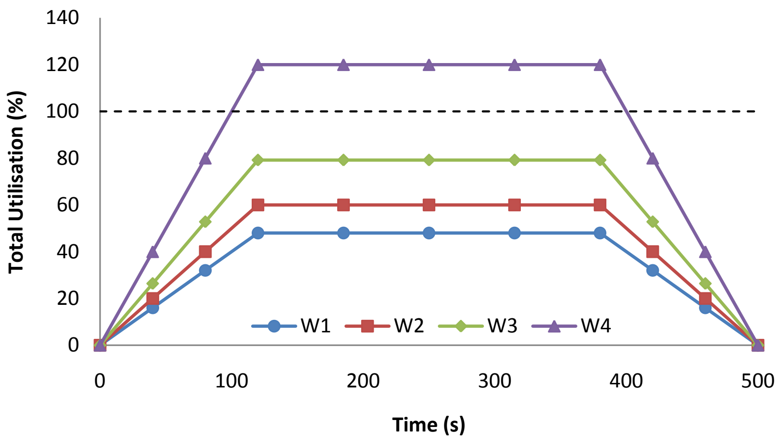
Table 6.11: Summary of auction rate, contract rate, allocation rate and system utilisation for each high utilisation strategy.

Workload	Job Size (relative to one host)	Maximum System utilisation
W1	20%	48%
W2	25%	60%
W3	33%	79.2%
W4	50%	120%

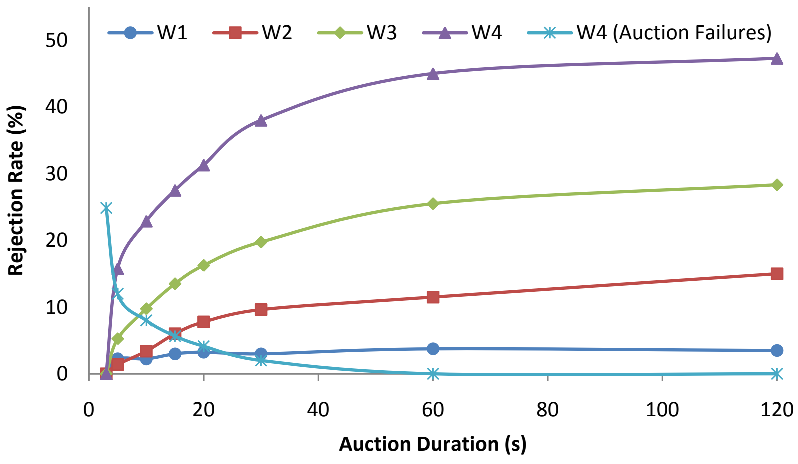
Table 6.12: Synthetic workload characteristics for auction latency.

50% which allows only 2 tasks to be hosted on a single provider simultaneously. When using 20% tasks all tasks can be hosted using only 50% of total cumulative provider capacity, increasing task utilisation increases the total system capcity used by the workload beyond the total capacity of the testbed.

Figure 6.34(b) shows the number of contracts rejected when each workload is submitted using an increasing auction duration. The graph also includes the auction failure rate for workload W4 as this workload exceeds provider capacity and therefore some auctions must fail. As expected when the auction duration (latency) is less than the period in which tasks are submitted no contracts are rejected for any of the workloads. Nearly a quarter of the auctions fail in workload 4 because the providers do not have sufficient capacity. For each of the workloads the rejection percentage increases with the increase in auction duration highlighting the effect of auction latency. The number of auction failures exhibited in workload W4 decrease due to underutilisation of the system.



(a) System utilisation for each synthetic latency workload.



(b) Contract (and Auction) rejection rate for each of the synthetic latency workloads.

Figure 6.34: System utilisation of the synthetic latency workloads and the rejection rate when applied to each strategy.

Time Before Auction Close	S + O Average Depth	R + S + O Average Depth
Medium		
30 seconds	2.29	2.04
10 seconds	1.81	1.69
3 seconds	1.68	1.49
0.5 seconds	1.13	1.23
High		
30 seconds	2.55	2.76
10 seconds	1.98	2.16
3 seconds	1.77	1.74
0.5 seconds	1.19	1.29

Table 6.13: Average number of second chance substitute providers considered as JIT bidding gets closer to auction close.

JIT Bidding

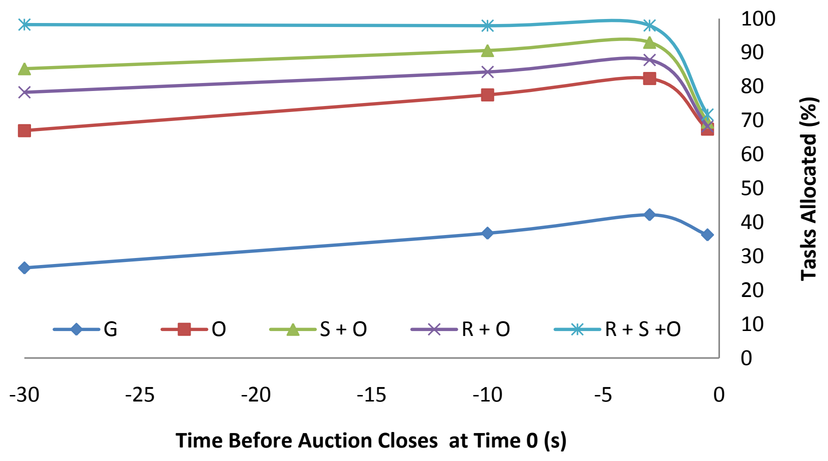
In general there are two techniques to mitigate the effect of latency; reduce the auction period or bid closer to the end of the auction. JIT bidding refers to withholding ones bid until as late as possible to ensure close to absolute capacity knowledge. This section investigates the effect of JIT bidding on the various high utilisation strategies outlined in Section 6.6.

Figure 6.35(a) and Figure 6.35(b) show the increased allocation when bidding closer to the close of the auction for the medium and high utilisation workloads respectively. For both workloads the number of tasks allocated increases by approximately 10% for each strategy up until a point of saturation - at which time bids are not received before the auction closes. The two strategies employing second chance substitutes in both workloads do not exhibit as much of an improvement, that is, the auction will not fail as long as alternative substitutes are available. Table 6.13 shows the average number of substitutes considered for each strategy as bidding time gets closer to auction close. Although the utilisation improvements are smaller for the second chance strategies, the number of substitutes considered decreases significantly as bidding occurs closer to the auction close. This is beneficial as it reduces the overhead required to compute second chance providers.

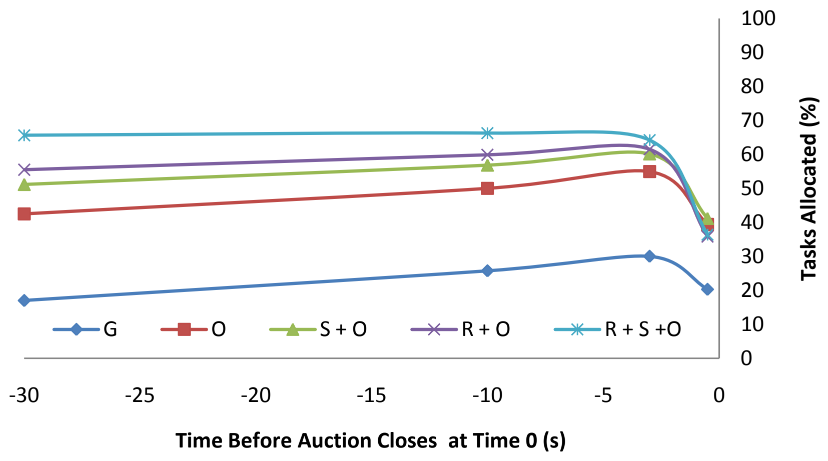
6.6.9 Reservation Window Sensitivity Analysis

The results shown in this section (Section 6.6) use an artificial flexible reservation window defined as 50% of the job duration. Figure 6.36 presents a sensitivity analysis that shows the effect of altering this articifial window as a percentage of job duration for each of the workloads and reservation strategies. The low workload with reservations and second chance substitutes (R + S + O) is not included as there are on average only 1.3 rejections without any reservation window.

As expected the rejection rate decreases with an increase in reservation window. For all combinations the greatest decrease occurs between not having reservations and a 10% reservation



(a) JIT Bidding for each strategy on the medium workload.



(b) JIT Bidding for each strategy on the high workload.

Figure 6.35: JIT Bidding for each strategy on the medium and high workloads.

window, this is indicitive of the short duration jobs and the finely packed schedules the providers are operating with. Even a minimal increase in job reservation window gives the flexibility to greatly increase allocation. The strategies with both reservations and substitutes decrease more gradually than those with only reservations, this is probably due to the fact that the reservation and substitute series are closer to maximum utilisation and therefore have fewer oportunities to improve occupancy. The high workload strategies have the lowest allocation rate due to the density of the workload. As the degree of time flexibility increases, it is expected that all algorithms tend to 100%, the rate of increase is dependent on the workload.

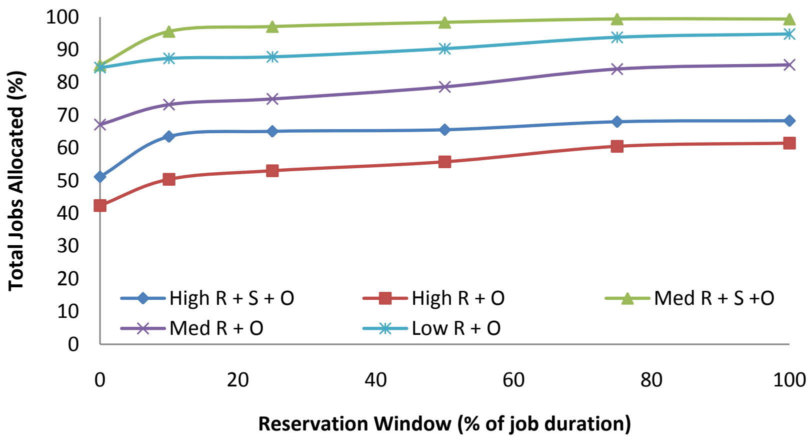


Figure 6.36: Total number of jobs allocated for an increasing reservation window (as a percentage of individual job duration).

6.6.10 Batch Workload

The experiments presented in Sections 6.6.1 – 6.6.9 are based on high performance synthetic workload traces derived from the original AuverGrid trace. These traces represent a fine grained model designed to simulate worst case auction conditions and have been scaled such that they can be hosted in a small scale virtualised testbed. To complete the evaluation of the proposed strategies, this section examines the benefits of each strategy under a traditional batch model characterised by infrequent long duration jobs. This model represents favourable auction conditions as the ratio of auction latency to interarrival time is large. Due to the requirements of the trace the providers in the testbed are configured with increased individual capacity, 20 providers each have 15 times the capacity of the previous testbed. This increased capacity has a significant effect on rejection rate as providers can win multiple concurrent auctions without rejecting agreements, only when providers are heavily utilised will the same rejection occur.

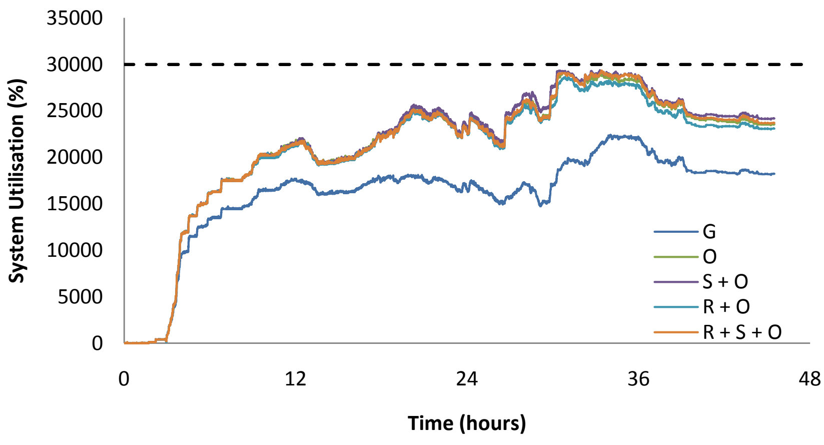
Each of the high utilisation strategies has been repeated on the two day batch workload using three different auction durations (2 minutes, 5 minutes, and 10 minutes). A summary of the results of these experiments is presented in Table 6.14. The total system utilisation for each strategy and workload is shown in Figure 6.37. These results show that even under conditions favourable to auctions the allocation rate is greatly increased when employing overbidding. The guaranteed baseline strategy results in allocation rates of 68%, 59% and 47%. Using an overbidding strategy these allocation rates are all increased beyond 95%.

The difference between the other strategies (O, S + O, R + O, R + S + O) is less dramatic for a number of reasons, including: increased provider capacity, utilisation characteristics of the workload, and decreased auction latency. The increased provider capacity significantly alters the allocation model from that used in the fine grained experiments. In this model each provider effectively has 15 times the capacity, this results in masking the limitations of overbidding as providers can win multiple simultaneous auctions without rejecting any agreements. The maximum system requirements of the workload only marginally exceed the capacity of the testbed resulting in a very high achievable allocation rate. The allocation rates for each strategy are therefore similar to the low utilisation fine grained workload previously examined. For each auction duration considered the difference between strategies becomes clear as the workload approaches testbed capacity (Figure 6.37), when providers are underutilised (first 24 hours) little difference can be seen as few contracts are rejected, however at 30 hours, differences begin to appear for each duration. Finally the auction duration combined with the low arrival rate of the workload results in providers bidding on few auctions without absolute knowledge. On average, approximately 35 auctions are conducted over the maximum 10 minute duration, 17 auctions over the 5 minute duration, and only 7 auctions over the 2 minute duration.

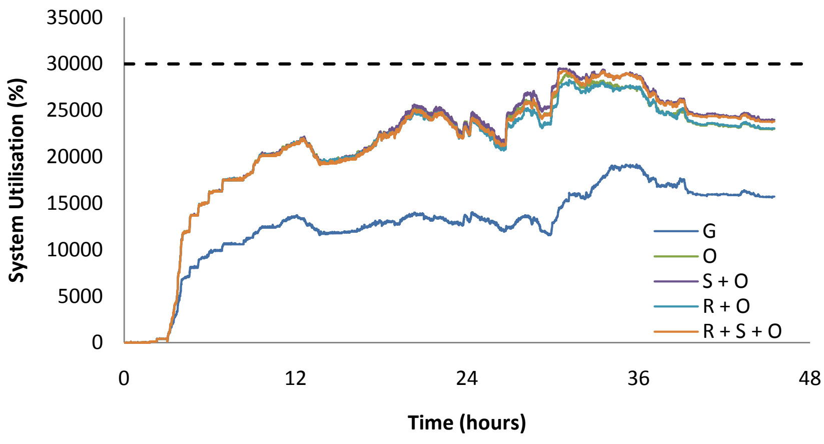
As was the case in the fine grained experiments using a combination of reservations, second chance substitutes and overbidding results in the highest allocation rate and system utilisation for each of the auction durations. The allocation rate for all strategies decreases as auction latency increases (as was shown Section 6.6.8), this is due to providers bidding with limited knowledge. The experiments allowing second chance substitutes (S + O and R + S + O) are less sensitive to the increase in auction duration as allocation rates only differ by up to 0.41% (S + O) and 0.07% (R + S + O), in contrast the overbidding (O) and reservation (R + O) strategies differ by approximately 2% over the durations examined. For each auction duration the overbidding (O) and reservation (R + O) strategies also exhibit approximately 2%–4% lower allocation rates than their substitute equivalents (S + O and R + S + O). Of the strategies considered the use of reservations has the least impact on allocation rate, this is primarily due to the sparse workload used. Section 6.6.4 showed reservations to be most valuable on dense workloads such as the high utilisation workload, the low utilisation workload (which is similar to the batch workload) exhibited little improvement using reservation strategies.

6.6.11 Summary

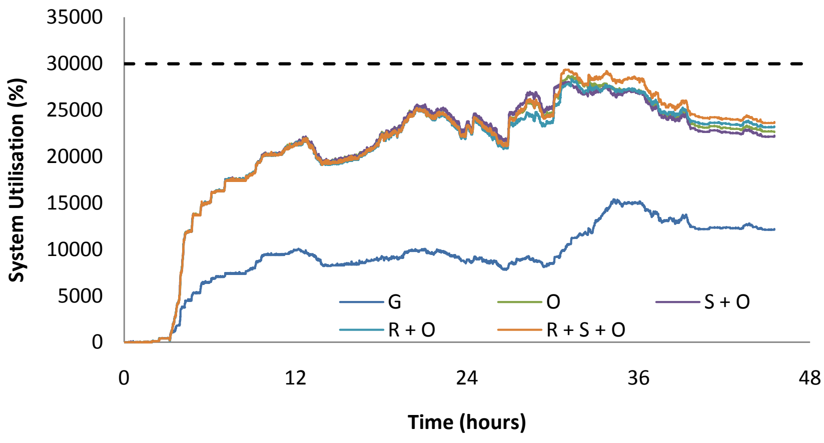
Economic allocation has been traditionally stereotyped as a low utilisation solution due to the overheads and latency in the allocation process. This section has validated the proposed economic strategies that can be employed in a distributed computational economy to increase oc-



(a) 2 minute auction duration.



(b) 5 minute auction duration.



(c) 10 minute auction duration (not done).

Figure 6.37: Cumulative system utilisation for each auction duration using the two day batch workload.

		Auctions		Contracts			Avg Sys	Max Sys
	Tasks	Failed	Complete	Rejected	Substitutes	Allocated	Util %	Util %
2min								
G	10153	3246	6907	0	N/A	6907 (68.03%)	53.21	74.64
O	10153	2	10151	265	N/A	9886 (97.37%)	70.33	97.35
S + O	10153	47	10106	39	303	10067 (99.15%)	71.30	97.85
R + O	10153	0	10153	227	N/A	9926 (97.76%)	69.28	95.42
R + S + O	10153	19	10134	17	251	10117 (99.65%)	71.48	97.76
5min								
G	10153	4111	6042	0	N/A	6042 (59.51%)	42.25	63.93
O	10153	7	10146	347	N/A	9799 (96.51%)	69.61	96.39
S + O	10153	40	10113	77	430	10036 (98.85%)	71.09	98.33
R + O	10153	0	10153	298	N/A	9855 (97.06%)	68.89	94.23
R + S + O	10153	13	10140	14	347	10126 (99.73%)	71.25	97.43
10min								
G	10153	5311	4842	0	N/A	4842 (47.69%)	30.80	51.30
O	10153	0	10153	505	N/A	9684 (95.03%)	68.91	95.69
S + O	10153	17	10136	111	775	10025 (98.74%)	69.18	93.47
R + O	10153	0	10153	424	N/A	9729 (95.82%)	68.48	94.49
R + S + O	10153	1	10152	31	488	10121 (99.68%)	70.19	97.88

Table 6.14: Summary of auction rate, contract rate, allocation rate and system utilisation for each high utilisation strategy using the batch workload model.

cupancy and therefore system utilisation. Each of the strategies proposed (overbidding, second chance providers, reservations, and JIT bidding) were shown to improve the allocation rate and overall system utilisation of the testbed using both synthetic fine grained workload models and a traditional long duration batch model.

Each of the strategies can be implemented by different parties in the system, for example overbidding and JIT bidding are implemented by providers while second chance providers are determined by the protocols (and participating entities). From a providers perspective the improvement gained from overbooking is substantial and when combined with appropriate bidding policies could be used to greatly increase revenue. JIT bidding can also be implemented by providers to minimise the effects of latency and permit pricing decisions based on more up to date market data therefore increasing profit and minimising risk. The use of reservations and second chance providers are dependent on the task requirements and the auction protocol used, they are therefore not controlled by providers or consumers. Participants could choose to "opt out" of allocations that use second chance providers, however as providers can obtain higher allocation rates and consumers are more likely to have jobs allocated it is both parties best interest to participate. From a consumers perspective these strategies can be used to increase the rate of resource allocation and potentially reduce the cost of resource usage. The economic implications of these strategies are examined in the next section.

6.7 Economic Performance

The analysis presented in the previous section focused on the allocation performance of DRIVE without considering the economic principles on which DRIVE is built. The goal of providers in a competitive economic system is to maximise their revenue. In DRIVE providers can implement any user defined pricing function or bidding policy to value requests and compute a bid. Several pricing and penalty functions have been implemented in the DRIVE prototype to evaluate their impact on allocation and revenue. This section examines different pricing and penalty functions applied to the high utilisation strategies defined in Section 6.6.

The following experiments are conducted on the complete virtualised testbed (Subset A and B). A sealed bid second price auction protocol is used to allocate jobs. In each experiment providers bid using non uniform pricing and penalty models. Bids are based on the requirements of a job (in units). The pricing function is used to determine a unit price between 0 and 20. Due to this decreased bid space (previous experiments used a 100 value bid space) tied bids are more likely. For this reason tied bids are resolved randomly so that provider latency does not impact auction results. The auction period used is 30 seconds and the reservation window is again set at 50% of job duration.

6.7.1 Pricing Functions

The pricing functions implemented for these experiments are: Random, Constant, Available Capacity, Win/Loss Ratio, and Time based. These functions were chosen to explore the use of a range of local information without the use of published market information. All bid prices are determined based on a combination of current conditions and previous bidder experience. In the following equations P_{unit} is the price per unit and B denotes the bid range ($B \in (0, B_{max})$) given by the maximum allowable bid value (for these experiments $B_{max} = 20$). Job units are defined as the product of CPU utilisation and duration ($J_{units} = J_{utilisation} \times J_{duration}$).

- Random: the unit price is determined irrespective of any other factors. Random acts as a baseline configuration from which the other pricing functions can be compared. The price per unit is given by the equation:

$$P_{unit} = Random(0, B_{max})$$

- Constant (10 & 20): the unit price is constant for every request and is given by the equation:

$$P_{unit} = c, c \in (0, B_{max})$$

- Available Capacity: the unit price is calculated based on projected provider capacity. The unit price is given by the following equation, where $U_{provider}$ is the current utilisation of the provider, U_{job} is the utilisation of the requested job, and $C_{provider}$ is the total capacity of the provider.

$$P_{unit} = \lceil \frac{U_{provider} + U_{job}}{C_{provider}} \times B_{max} \rceil$$

Using a bid space of 20, the unit price decreases every 5% of unused provider capacity. For example if the projected utilisation is 94% of provider capacity the unit price is 19, if the projected utilisation is 12% of provider capacity the unit price is 3.

- Win/Loss Ratio (20:1 & 10:1): the unit price is based on the previous win/loss ratio seen by the provider. In the first configuration a win is the equivalent of 20 losses (based on the assumption a provider will win 1 of 20 auctions as there are 20 providers). The second configuration uses a ratio of 1:10, that is 1 win is equivalent to 10 losses. The unit price is determined using the following formula, where R is the specified ratio, W is the number of wins, and L is the number of losses. The price is scaled to the the bid range and also offset to the middle of the bid range.

$$P_{unit} = (RW - L) \times \frac{B_{max}}{R} + \frac{B_{max}}{2}$$

- Time Based (30 & 5): the unit price is based on the time since the provider last won an auction. In the first configuration the unit price decrements every 30 seconds, in the second configuration it decrements every 5 seconds. The unit price is given by the following equation, where $T_{lastwin}$ is the time since the last auction win and T_{period} is the specified time period.

$$P_{unit} = B_{max} - \left(\frac{T_{lastwin}}{T_{period}} \right)$$

6.7.2 Penalty Functions

Overbidding has been shown to increase occupancy and utilisation, however the major limitation of bidding beyond capacity is an increased defaulter rate. The use of substitute providers was shown to minimise the impact of resource state change between bidding and winner determination. However, this leads to an increased overall average net price to consumers as a result of differing provider valuations. DRIVE uses the notion of negative incentives (or penalties) to offset price increases when agreements are not honoured within the system. As described in Section 4.5.2 there are a range of penalty types suitable for encouraging compliance in DRIVE. This section focuses only on financial penalties as quantifying non functional penalties (such as reputation damage) is out of scope. In DRIVE, policies regarding penalties may be issued at the VO level, by consumers, or collectively by providers. In all cases the penalty function is defined before negotiation.

The range of permissible penalties differs depending on the strategy used. When overbidding, financial penalties may be constant, based on the task “size”, or the value determined in the original negotiation. The use of substitute providers adds an additional parameter to consider, that is the difference between the original winning price and the reserve substitute price. This knowledge can be used to define penalties that truly reflect the impact on consumers.

The DRIVE prototype includes two different types of penalties invoked for every broken contract: constant and dynamic penalties. *Constant* penalties are fixed penalties that are statically defined irrespective of any other factors. *Dynamic* penalties are based on a non-static variable to reflect the value of a job. Two classes of dynamic penalties have been implemented to model the impact of a breach, these classes are termed α and β penalties. α penalties are defined based on the relative size of the job or the established price. β penalties attempt to model the increased cost incurred by the consumer using a ratio of the original and substitute prices to determine a penalty that reflects the effect of the breach. Specifically the different penalty functions implemented are:

- Constant: a constant penalty defined statically irrespective of the job requirements or bid price. Constant penalties are given by the equation:

$$P_{defaulter} = c, c \in \mathbb{R}_{\geq 0}$$

- Job Units: an α penalty based on the requirements of the job in units. This penalty is given by the following equation, where J_{units} is the number of units in a job, and c is a constant penalty per unit.

$$P_{defaulter} = J_{units} \times c$$

- Win Price: an α penalty based on the winning bid price, given by the following equation where $Price_{win}$ is the price to be paid by the winning bidder and ϕ is the penalty impact factor. The penalty impact factor describes the fraction of the win price imposed in a penalty (for the following experiments the full win price is used ($\phi = 1$)):

$$P_{defaulter} = Price_{win} \times \phi, \phi \in [0, 1]$$

- Substitute Price: an α penalty based on the substitute price, given by the following equation where $Price_{substitute}$ is the price paid by the substitute winning bidder and ϕ is the penalty impact factor:

$$P_{defaulter} = Price_{substitute} \times \phi, \phi \in [0, 1]$$

- Bid Difference: a β penalty defined as the difference between the original win price and the substitute price.

$$P_{defaulter} = Price_{substitute} - Price_{win}$$

- Bid Difference/Depth (1): a β penalty that determines the impact of an individual provider defaulting on a contract. The impact is calculated as the difference between original win price and substitute price divided by the number of substitute providers considered. In the first configuration only a single penalty is applied to the original winning provider, this is given by the following equation, where $Depth_{substitute}$ is the number of substitutes considered.

$$P_{defaulter} = \frac{Price_{substitute} - Price_{win}}{Depth_{substitute}}$$

- Bid Difference/Depth (2): the second configuration imposes a fraction of the difference penalty on each defaulting provider. This has the effect of spreading the penalty amongst all winners (and substitute providers) that cannot honour the agreement. The penalty is given by the equation:

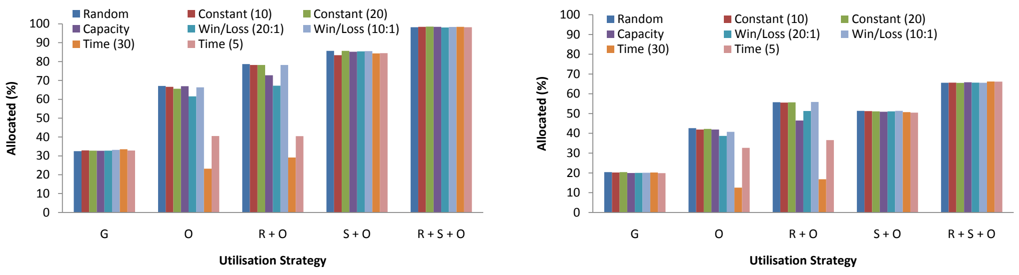
$$\forall i \in D : P_i = \frac{Price_{substitute} - Price_{win}}{Depth_{substitute}}$$

In general there is a tradeoff between fairness and complexity of penalty functions. For example while a constant penalty is easy to enforce and requires no computation it is not fair in terms of which defaulters pay the penalty and does not reflect the size or value of the job (both small and large jobs are penalised the same amount). The second bid difference depth penalty is arguably the fairest penalty as it calculates the value of the breach and spreads the penalty over all defaulting providers. However, the implementation of this penalty is complex as the difference in price is calculated and and then a penalty is applied to each defaulting provider. Due to this complexity and the number of potential defaulters this function could take considerable time. To resolve this limitation the penalty function could be optimised to reflect VO policies, for example limits may be placed on the maximum time to impose all penalties so that providers are aware of penalties immediately. In addition a maximum defaulter depth could be used to penalise only the initial defaulters. A simple weighted decay function could also be implemented such that initial defaulters are penalised more harshly than substitute defaulters.

6.7.3 Allocation

Figure 6.38 shows the allocation rate for each pricing function and utilisation strategy. Penalty functions are not included in these results as currently penalties are not considered by bidders prior to bidding and therefore do not effect allocation. As expected the allocation rate is similar for each pricing function due to the fact pricing is designed to only influence the price paid. Each function is designed to produce lower bids when resources are underutilised and higher bids when resources are overutilised. The guaranteed (G) strategy results in near identical utilisation across all pricing functions, this is because every bid is a guarantee and therefore the bid value is inconsequential. As providers do not bid beyond capacity the resulting contract can always be honoured. The substitute methods (S + O and R + S + O) also produce near equal allocation independent of the pricing function used, this is because substitute methods are not bid sensitive – assuming there is sufficient available capacity, rejected contracts can be substituted by a different provider at an increased cost.

The more risk averse pricing functions time and win/loss, produce lower allocation for the non substitute strategies (O and R + O). This is primarily due to the way in which providers compute bids and the fact that substitutes are not used. Providers compute bids based on previous events, therefore a providers bid cannot increase until they win a new auction. As a result, from the time a bidder becomes the lowest bidder until they win another auction their bid will remain the lowest. During this time if multiple auctions are held there will be considerable failure. This clearly applies to the time based pricing function, it also applies to the win/loss strategy assuming providers continue to bid (and lose) at the same rate as other providers in the system. Available capacity pricing leads to slightly lower allocation rates when using reservations on both workloads, this is due to an implementation oversight as the calculation of utilisation does not include any queued reservations. Therefore the bid price is reflective of currently running tasks only, in the situation where a larger task is queued the bid price will be artificially low. The constant pricing functions produce the same allocation rate as the other functions due to the random order in which auctions are advertised and the random tie resolution.



(a) Medium Workload. (b) High Workload.

Figure 6.38: Number of tasks allocated for each of the pricing functions and strategies considered.

6.7.4 Revenue

This section examines the revenue generated for each combination of high utilisation strategy, pricing function, and penalty function on the medium and high workload. Results are not shown for the low workload as it has few rejections and therefore few penalties are enforced. The penalty functions are denoted: Constant 100 (C100), Constant 200 (C200), Job Units (JU), Win Price (WP), Half Win Price (1/2WP), Substitute Price (SP), Half Substitute Price (1/2SP), Bid Difference (BD), Bid Difference/Depth 1 (BDD1) and Bid Difference/Depth 2 (BDD2). In addition No penalty (NP) is included to examine the total revenue generated when no penalties are enforced. Only constant and α penalties are shown for the Guaranteed (G), Overbidding (O), and Reservation (R + O) strategies because β penalties cannot be applied as no substitute providers are used. Additional α and β penalties related to substitute providers are shown for the Substitute (S + O) and Reservation/Substitute (R + S + O) strategies. The following graphs show total revenue generated across all providers when different penalties are applied. Positive revenue indicates a profit to the provider, whereas negative revenue indicates a loss.

When supporting second chance substitute providers there are two different types of penalties that can be enforced: some contracts are rejected and are unable to be satisfied by a substitute provider, while others can be allocated to a substitute provider at an increased cost to the consumer. The experiments that do not utilise substitutes (G, O and R + O) study an array of penalty functions focused on the first category (no substitutes). The experiments that utilise substitutes (S + O and R + S + O) are designed to examine the second case (using substitute providers).

- In these experiments no penalty is applied if contracts are rejected without the ability to substitute the winner, this provides the ability to accurately compare the different penalty functions.

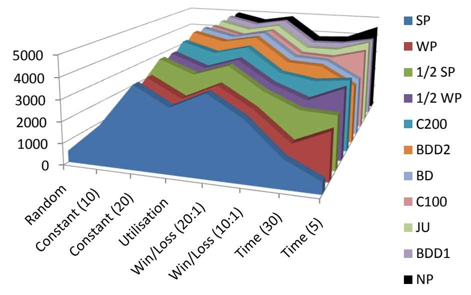
When examining the revenue generated it should be noted that in reality additional penalties would also be applied however these would be constant across all penalty functions. In the medium workload between 1% and 23% of contracts fail, in the high workload between 35% and 60% of contracts fail, this failure rate equates to considerable additional penalties.

Figure 6.39 shows the revenue generated for each combination of pricing and penalty function when using the reservation and substitute strategy (R + S + O) on the medium and high workload. For visualisation the graphs are ordered by increasing total average revenue generated from each penalty function and pricing functions are joined even though the points are unrelated. The horizontal axis is ordered by increasing complexity of pricing function. The only difference in penalty order between the two graphs is the bid difference (BD, BDD2) penalties have a greater effect on revenue than constant 100 (C100) in the high workload. This difference is due to the increased number of substitutes used in the more dense workload. The total revenue generated in the high workload is approximately double the revenue generated in the medium workload, this is due to the increased number of jobs allocated and the allocation aware pricing functions.

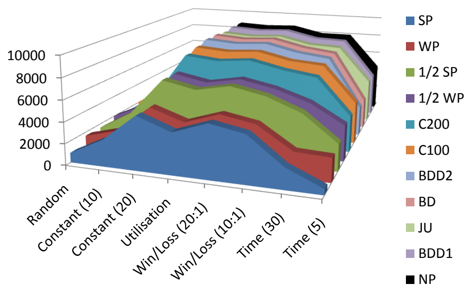
It is evident that the different pricing functions generate vastly different total revenue, for example the random pricing function generates very low system revenue whereas the utilisation based pricing function generates close to maximum revenue. The penalty functions also vary in their sensitivity to different pricing functions, for example in both workloads the Time (5) pricing function has higher revenue using win price penalties rather than substitute price penalties, whereas the other pricing functions do not vary much between win price and substitute price penalties. To examine the effects of the different combinations the following sections present slices of these graphs (by pricing function) for each high utilisation strategy. The guaranteed strategy is included in this analysis for completeness. As bids are guaranteed, no contracts should ever be rejected and therefore no penalties will be enforced. This results in the same revenue for all of the penalty functions examined using the guaranteed strategy.

Random Pricing Function

Provider revenue for the random pricing function is shown in Figure 6.40. Of the pricing functions considered random generates the least total revenue due to the range of expected bid values. Each provider generates a bid value (1 – 20), assuming all 20 providers bid randomly then the winning provider should win with a low bid value. The use of reservations has an interesting effect on



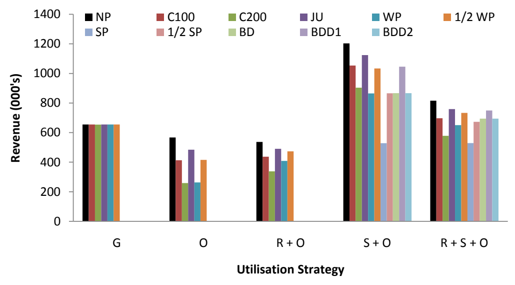
(a) Medium Workload.



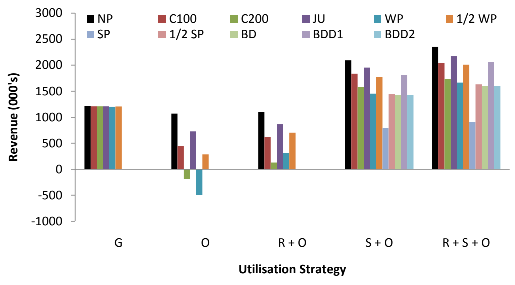
(b) High Workload.

Figure 6.39: Total system revenue using a random pricing function and the R + S + O strategies.

total system revenue without penalties (NP) in the medium workload, here the strategies using reservations generate less revenue than those that do not use reservations (for example compare S + O to R + S + O and O to R + O). This decreased revenue is because providers have more flexibility in scheduling jobs, therefore when providers win auctions they balance workload rather than rejecting the contract and using a substitute provider at an increased cost. This is not the case in the high workload as the significantly higher allocation rate using reservations leads to greater revenue generation. The medium workload operates at a profit for each penalty strategy considered, whereas the high workload exhibits loss in the overbidding strategy and little profit in the reservation strategy. However it is important to note in the substitute strategies, providers operate at a profit as penalties are only enforced when substitutes can be found. If penalties were enforced for contract failures the substitute strategies may also operate at a loss as 5000 and 3329 contracts are rejected in the substitute (S + O) and reservation/substitute (R + S + O) strategies respectively.



(a) Medium Workload.



(b) High Workload.

Figure 6.40: Total system revenue using a random pricing function.

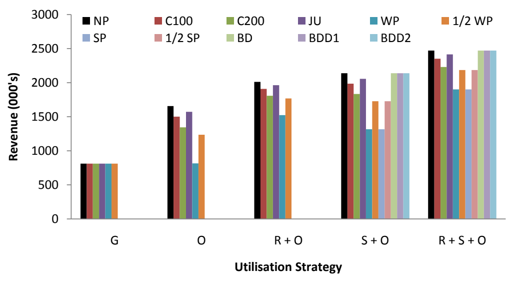
For the non substitute strategies (O and R + O) the penalties have similar effects for both workloads. The job unit (JU) based penalties incur the least decrease in revenue, this implies the bid price is generally greater than 1 per unit as one would expect. The difference between revenue and the constant penalties (C200 and C100) is exactly double as expected. In the medium workload using the overbidding strategy the constant (C200) penalty is very similar to the win price (and constant C100 is similar to half win price), this indicates that the average price of failed jobs is approximately 200. In the high workload using an overbidding strategy the win price and constant (C200) penalties make the overall revenue negative, this is due to the low allocation rate. The average win price in the high workload is also larger than the medium workload as revenue with win price penalties is much lower than C200.

The constant and α penalties applied to substitute strategies (S + O and R + S + O) exhibit similar characteristics as seen in the non-substitute strategies. The substitute price (SP) penalty presents the largest decrease in provider revenue. This shows there is a large difference between the substitute and original win price. Recall the substitute price is not necessarily the second highest bid it could be any providers bid as the Auction Manager iteratively determines the next highest bidder. The bid difference (BD) penalty in each case is similar to both the initial win price (WP) and also half substitute price (1/2SP) for both substitute strategies, this indicates the increase in price due to substitutes is roughly equivalent to the initial job price. Perhaps the fairest penalty is the measure of how much an individual defaulting provider impacts the price paid (BDD2), essentially this penalty spreads the difference over each defaulting party. The first configuration (BDD1) only imposes this penalty on the original defaulter which is why the revenue generated is greater than the BD penalty. The relatively small difference between BDD1 and BD reinforces the finding that average backup depth for the medium and high workload is between 2 and 3 (Section 6.6.3). The second configuration (BDD2) imposes a fraction of the penalty on each defaulting party, as expected this results in the same total revenue over all providers as imposing the entire difference on a single provider (BD).

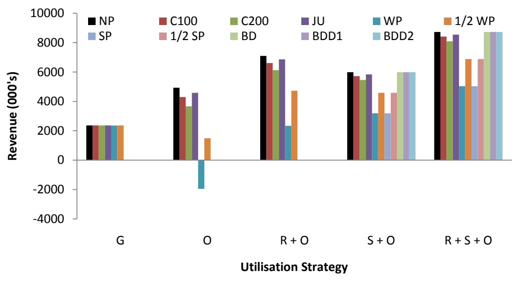
Constant Pricing Function

The constant pricing functions provide reference revenue values. Constant (20) provides an upper bound on the potential revenue generated, whereas constant (10) provides a mid point. Figure 6.41 shows the revenue generated when each penalty is applied to the constant pricing functions. As expected constant (20) generates the highest revenue of the pricing functions examined and constant (10) generates exactly half as much revenue for each strategy. These strategies highlight the advantage of price fixing as all providers agree to bid the same price, therefore there is little competition amongst providers and system profit can be maximised. However, in non competitive environments there is nothing to reinforce this collusion.

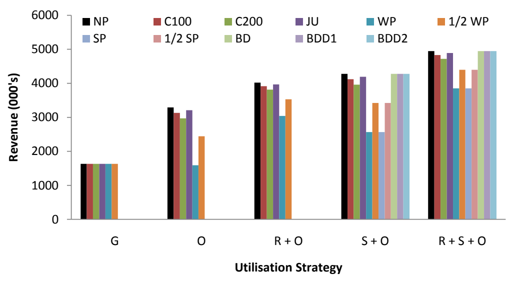
Win price (WP) penalties on the non substitute strategies represents the largest penalty in both workloads as bid prices are relatively high. In fact on the high workload using an overbidding strategy WP penalties results in a net loss for providers as more contracts are rejected than honoured. Using reservations the allocation rate increases above 50% which in turn leads to a profit. In the substitute strategies win price (WP) and substitute price (SP) penalties produce identical revenue as providers implement a price fixing scheme and therefore all bids are identical. This is



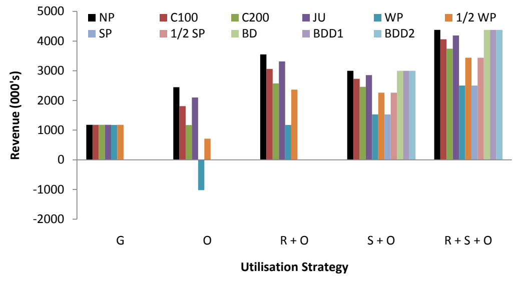
(a) Medium Workload - Constant 10.



(b) High Workload - Constant 10.



(c) Medium Workload - Constant 20.



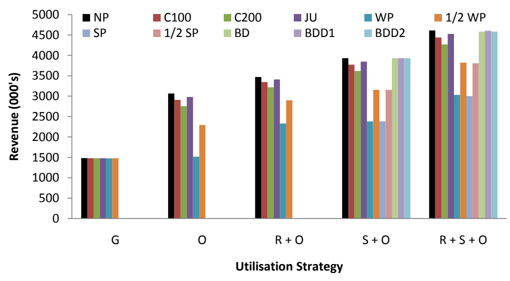
(d) High Workload - Constant 20.

Figure 6.41: Total system revenue using a constant pricing function.

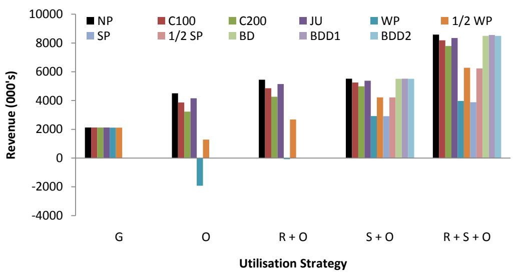
also reflected in the difference penalties (BD, BDD1, BDD2) as all three functions produce optimal revenue due to the fact there is no difference between bids and therefore no penalty is imposed.

Available Capacity Pricing Function

Figure 6.42 shows the revenue generated for the available capacity pricing function. The revenue is near maximum (similar to Constant 20) for each strategy indicating most providers bid high due to little available capacity. This lack of capacity is not due to high system utilisation, rather it shows that most jobs use close to 100% capacity, this is evidenced by the average job size of 93.67% (Section 6.5.5). Again the win price (WP) penalties operate at a loss for overbidding and reservations on the high workload. The high average job size means that few jobs can be hosted simultaneously by a provider which results in providers producing the same bid for most jobs. This similarity in bid price results in no difference between win price and substitute price which results in near maximum revenue for the difference penalties (BD, BDD1, BDD2). These similarities are more clearly shown in Figure 6.43, here the total value of all penalties is shown as a percentage of total system revenue. The win price and substitute penalties are substantially more than any of the other penalaties.

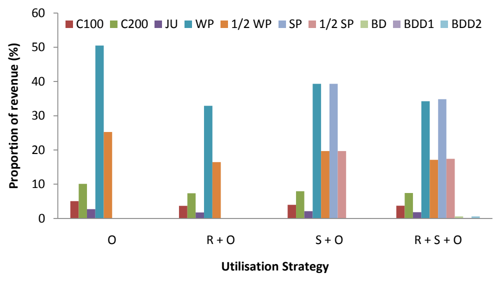


(a) Medium Workload.

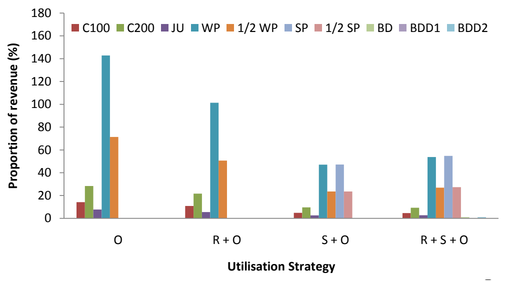


(b) High Workload.

Figure 6.42: Total system revenue using an available capacity pricing function.

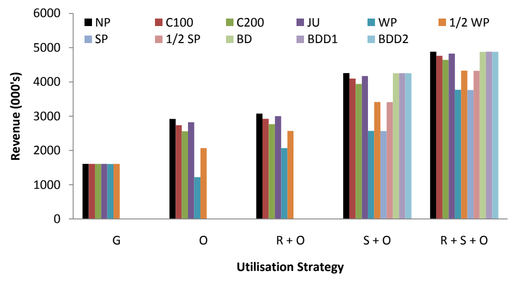


(a) Medium Workload.

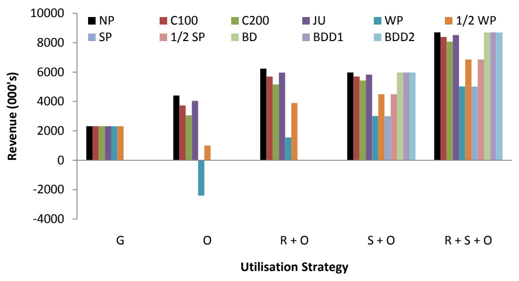


(b) High Workload.

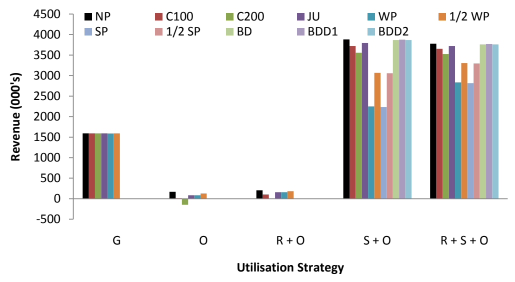
Figure 6.43: Total penalty value as a percentage of total system revenue using an available capacity pricing function.



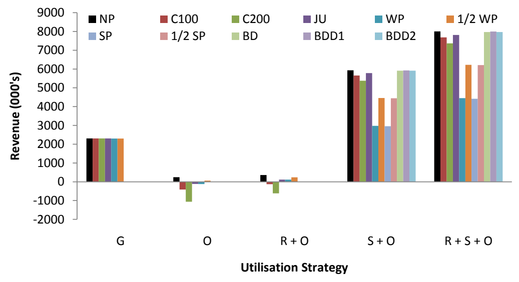
(a) Medium Workload - 20:1.



(b) High Workload - 20:1.



(c) Medium Workload - 10:1.



(d) High Workload - 10:1.

Figure 6.44: Total system revenue using a win/loss ratio pricing function.

Win/Loss Ratio Pricing Function

The total system revenue using different win/loss ratios is shown in Figure 6.44. The differences between high utilisation strategies is evident in both workloads as the win ratio is dependent on the particular strategy used. A ratio of 20:1 produces maximum revenue for the two substitute strategies on both workloads which is indicative of the ratio being too high. The ratio assumes that providers win 1 in 20 bids, however this is not the case when using substitutes as multiple providers can win a single auction. In addition providers do not bid (and lose) when they are fully utilised. Win/Loss Ratio (10:1) produces lower revenue for the substitute strategies indicating providers win closer to 1 in 10 auctions using these strategies. However, in the non substitute strategies this policy produces very low revenue as providers win far fewer than 1 in 10 auctions.

The penalties enforced exhibit similar characteristics to the previous strategies indicating each winning provider bids a similar value. Win price (WP) penalties have the greatest effect on revenue for the medium workload using non-substitute strategies. In the high workload constant penalties result in a loss due to the poor allocation rate. Win price and substitute price (SP) penalties are almost identical due to a lack of difference between provider bids. This is also reflected in the near maximum revenue when using difference penalties (BD, BDD1, BDD2).

Time based Pricing Function

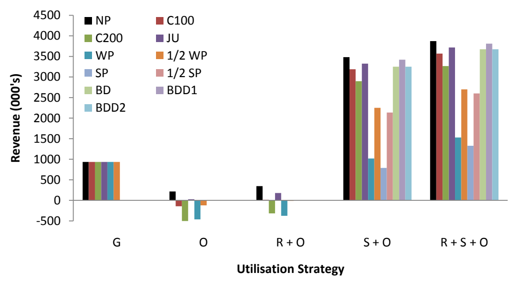
Figure 6.45 shows the total system revenue using time based pricing functions. The low revenue for the non substitute strategies is partly due to the low allocation rate and therefore the increased time between winning auctions, however it is also indicative of low bid values due to long delays between auction wins. Like the win ratio function the revenue generated is dependent on the strategy used. High revenue indicates auctions are won more frequently than the specified time interval and conversely low revenue indicates auctions are won less frequently. The medium workload submits approximately 519 jobs per hour, or almost 10 jobs per minute. Assuming a provider wins 1 out of every 20 auctions they should win an auction approximately every 2 minutes. Therefore with a time period of 30 seconds the unit price most of the time will be greater than 16 (especially in the substitute methods). The second time period of 5 seconds results in a unit price closer to 0 over this same period and represents a more sensitive model. This is confirmed in Figure 6.45 as the revenue using 30 second periods greatly exceeds that of the 5 second strategy for both workloads.

Due to the low allocation rate the non-substitute strategies operate at a loss for most of the penalty functions. With a 30 second time period, invoking win price (WP) penalties generates the same revenue as invoking constant C200 penalties. In the 5 second period the WP penalties do not result in such a loss as bid values are lower due to the shorter time between bids. The large bid space in the 5 second period is highlighted by the difference between win price and substitute price penalties, this difference is also reflected in the decreased revenue for bid difference penalties. This spread indicates providers use the full range of bid values as desired.

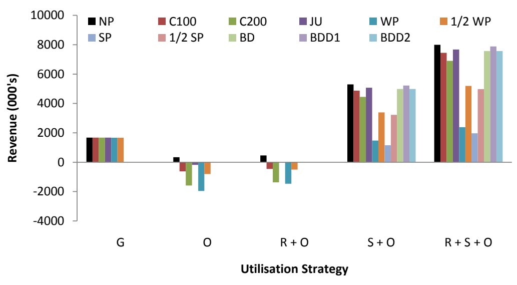
6.7.5 Summary

This section has examined the effect of different pricing and penalty functions on task allocation and system revenue using the proposed high utilisation strategies. Allocation rate was shown to be constant across most of the implemented pricing function, however the non-substitute strategies (O and O + R) exhibit drastically reduced allocation when providers bids are related to previous auction results. Total system revenue was shown to be heavily influenced by the pricing function and utilisation strategy used. The use of substitute providers for example effects revenue due to the number of providers winning and substituting auctions. Therefore the revenue generated is not entirely dependent on the allocation rate achieved by each strategy. The experiments conducted in this section use the same pricing functions for each provider to evaluate their effects on revenue, however in a real world scenario one would expect providers to implement different policies which may be favourable under particular constraints.

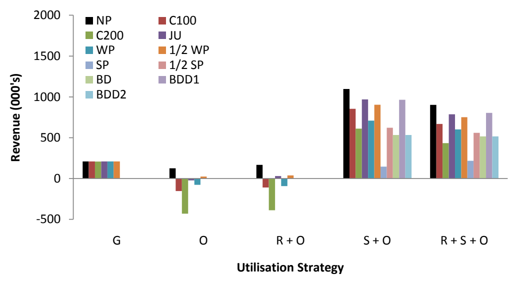
The random pricing function was shown to produce very low revenue due to the likelihood of at least one bidder bidding low. Constant pricing is essentially a price fixing model in which providers collude to raise prices, however in a competitive environment it may be advantageous for a provider to break the agreement as their is no means of enforcing collusion. Available capacity pricing is able to represent risk to providers, however as most jobs use almost 100% capacity



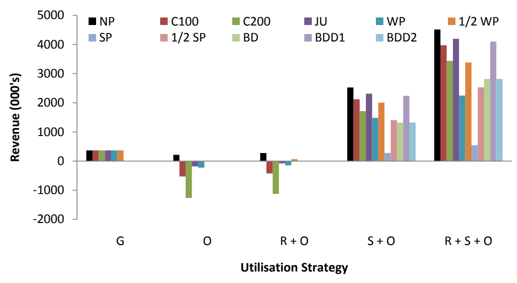
(a) Medium Workload - 30 seconds.



(b) High Workload - 30 seconds.



(c) Medium Workload - 5 seconds.



(d) High Workload - 5 seconds.

Figure 6.45: Total system revenue using a time based pricing function.

this function generates almost maximum revenue. This function would be most suitable when using large scale providers that can simulataionusly host multiple "average" jobs. Win loss and time based pricing were shown to be the most sensitive to the different strategies. The time based function is particularly inflexible due to the static declaration of a time period. To utilise such strategies in a real system a dynamic approach must be taken to allow the function to be tuned to current conditions.

The proposed penalty models provide a sample of possible metrics that could be considered in DRIVE. The constant and α penalties provide a simplistic way to encourage compliance, the advantage of these penalties is they are easy to implement and require very little data. β penalties are able to capture and distribute the effect of contract rejection over all defaulting parties, however they are only applicable when win price and substitute price are known, additionally for BDD2 all defaulting providers must be known. In the high workload many of the penalties imposed would make the system operate at a loss (if contract failures were also penalised). To avoid suffering loss penalty functions must be reduced or providers must incorporate knowledge of penalty functions when determining bid values.

6.8 Throughput

Throughput provides a measure of the maximum number of jobs that can be allocated within a period of time. Unlike centralised and queue based meta-schedulers, DRIVE relies on complex multi-partite negotiation protocols to establish the best match between jobs and providers. This increased protocol complexity and communication between parties is advantageous in untrusted environments, however it also reduces throughout. In an attempt to reduce communication and therefore increase auction performance the DRIVE architecture uses Publishing Services to distribute auction advertisements. This section analyzes the potential throughput improvements that can be obtained using distributed reverse discovery mechanisms rather than direct advertising.

In the following experiment throughput is calculated based on the total time taken to allocate 5000 jobs. The time is measured client side and includes the time to submit each job, conduct an auction, negotiate the resulting contract with the winning provider, and return the contract to the client. To simulate a realistic scenario a sealed bid second price auction protocol is used, the Auction Manager is allocated 1GB of RAM, and bidders are hosted in the full virtualised testbed. Auctions conclude when every bidder has bid on the auction. To examine the throughput using reverse discovery, bidders are configured to poll the publishing service(s) periodically. The jobs submitted do not use any provider resources so bidders can theoretically host every job, therefore there will be no auction failures. Bidders implement a random bidding policy irrespective of job requirements or available capacity.

Figure 6.46 shows the throughput (jobs per minute) for an increasing number of bidders when using targeted advertisements and reverse discovery. The throughput when directly advertising decreases exponentially with the number of bidders whereas the decrease in throughput using Publishing Services decreases linearly at a much more gradual rate. The advantage of publishing auctions and therefore almost halving Auction Manager communication is evident (advertisements are no longer sent to every potential bidder). Using a single Publishing Service with 50 bidders the throughput is triple that of direct advertising. Considering the complexity of the protocol and the relatively small scale Auction Manager used, a minimum 75 auctions per minute when advertising to 50 bidders represents reasonable throughput. Advertising overlaps with the publishing performance when the number of bidders is small as the communication is roughly the same and the burden on the Auction Manager is minimal. In a non LAN based deployment the time to communicate between Auction Manager and bidders would be increased and therefore throughput may be more balanced. Due to the distributed DRIVE architecture multiple Auction Managers can also operate simultaneously to increase overall system performance.

6.9 DRIVE Overhead

The final aspect of this evaluation focuses on the overhead of hosting DRIVE services. The DRIVE architecture is based upon a co-op model in which VO members are expected to contribute obliga-

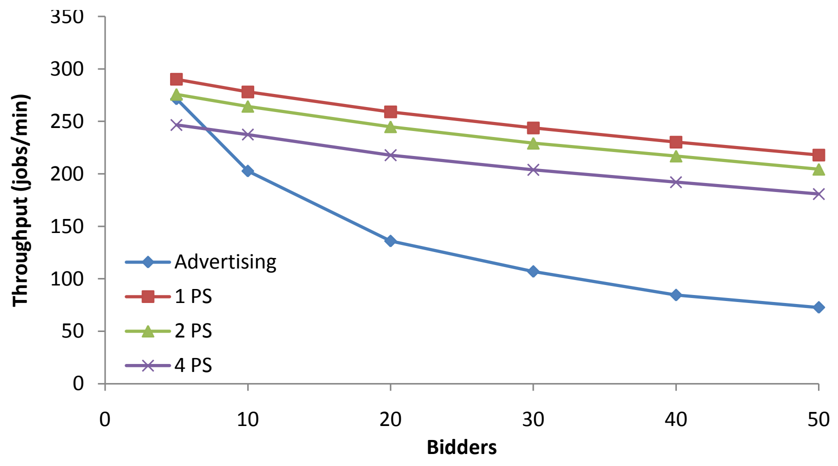


Figure 6.46: DRIVE total throughput using direct advertising and publishing. The throughput is inclusive of auction and contract creation.

tion services to provide core meta-scheduling functionality. To determine the requirements of the core DRIVE services each has been individually monitored under two workloads to determine the CPU, Memory and Input/Output (I/O) usage of the service. Only the core services required to conduct auctions and create contracts are considered in this analysis, the Auction Component is not considered as its functionality is dependent on the implemented protocol, the Publishing Service has not been included as it is not a necessary service to conduct an auction.

The following experiments are based on a virtualised testbed deployment, each of the services analysed is hosted on a dedicated host so as to minimise competition for resources. 20 bidders have been deployed to simulate the requirements of a realistic auction scenario and the bidders all implement a random bidding policy. Auctions have a duration of 30 seconds. The substitute and overbidding (S + O) strategy is used to model a realistic usage scenario. Two workloads have been developed to test increasing stress on each of the services. Workload A (Figure 6.47) submits a group of jobs simultaneously, ranging from 5 to 50 jobs. Workload B (Figure 6.48) spreads submission over a period of 30 seconds submitting the same number of jobs (5 to 50). To analyse the performance of using substitute providers the job requirements expressed in these workloads consume the entire capacity of the testbed.

CPU Usage

Figure 6.49 and Figure 6.50 show the CPU usage of each service for Workload A and B respectively. The CPU usage for Workload A exhibits sharper higher peaks whereas Workload B is flatter and wider due to the sparse job submission model. In both graphs as the number of jobs submitted increases the CPU usage of all services is both higher and wider. Both graphs highlight the order of events between services: as jobs are submitted the CPU usage of the Auction

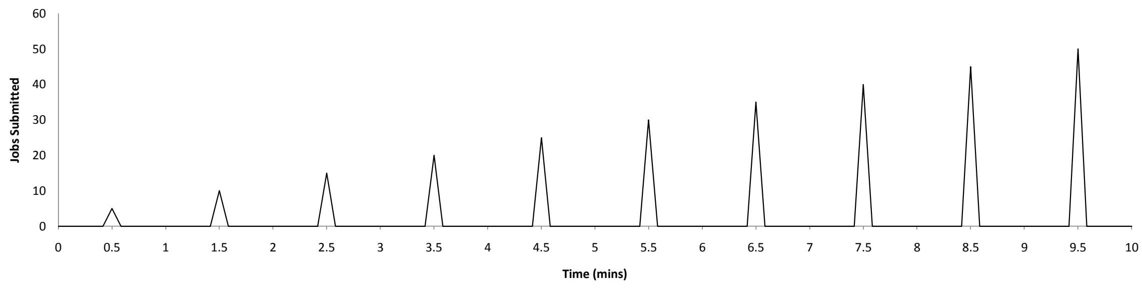


Figure 6.47: Workload A.

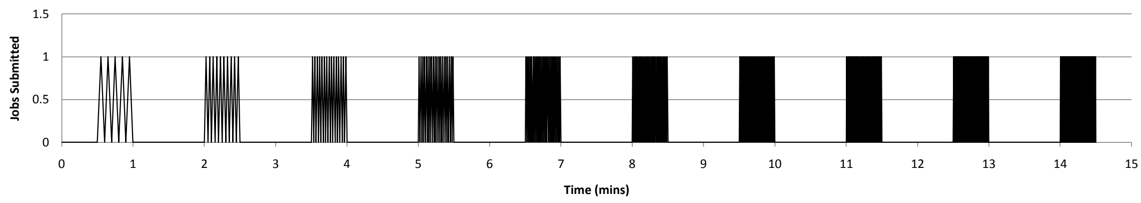


Figure 6.48: Workload B.

Manager peaks, which in turn generates work for MDS (to discover bidders) and the Bidding Agent (computing a bid), after the auction period of 30 seconds has elapsed the Auction Manager computes a winner and the Contract Manager CPU creates a contract. As the Contract Manager is operating there is an additional peak in the Auction Manager due to result retrieval and the computation of substitute providers. The heavy workload on the Auction Manager and Contract Manager makes them both good candidates for distribution.

The Auction Manager uses the most CPU of the services examined, approaching 100% in Workload A. Figure 6.49 shows the utilisation for each auction group increases steadily up until 6 minutes where the peaks remain constant but get wider, this is possibly due to a limit placed on the Auction Manager (for example the number of auction or container threads). Creating and advertising the auctions along with soliciting of bids consumes most CPU, winner determiniation and substitute computation in this case are small due to the fact a simple protocol is used and few bidders participate in the auction.

Contract Manager utilisation appears to be more spread in both Workload A and B, this may be due to asynchronous contract creation. In these experiments the broker initiates contract creation when it receives a notification from the Auction Manager that the auction has completed, if no broker threads are free then contract creation is queued. The spread seen in Figure 6.49 is

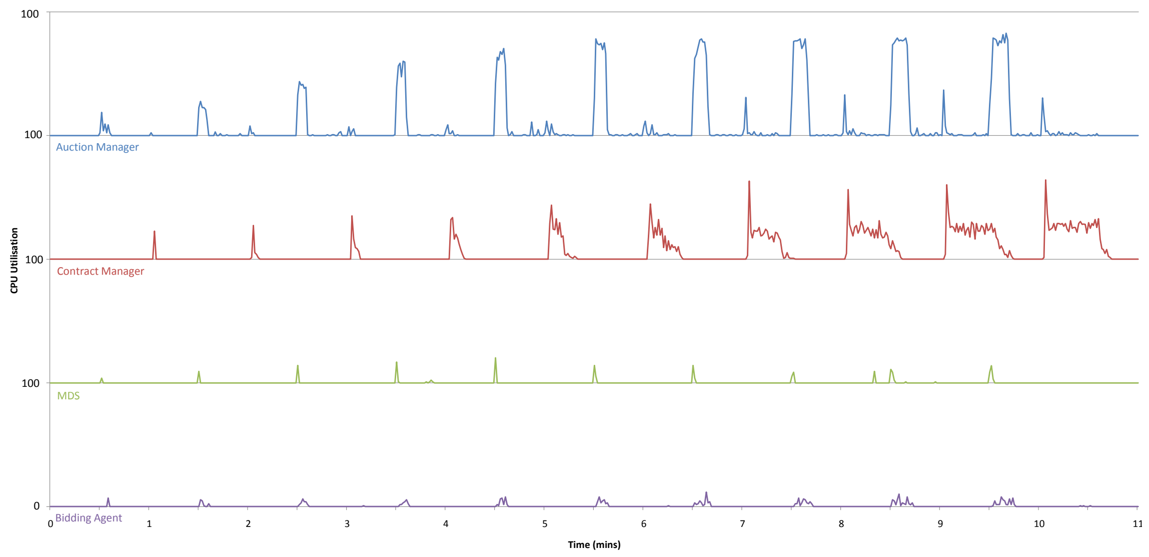


Figure 6.49: CPU Usage for workload A.

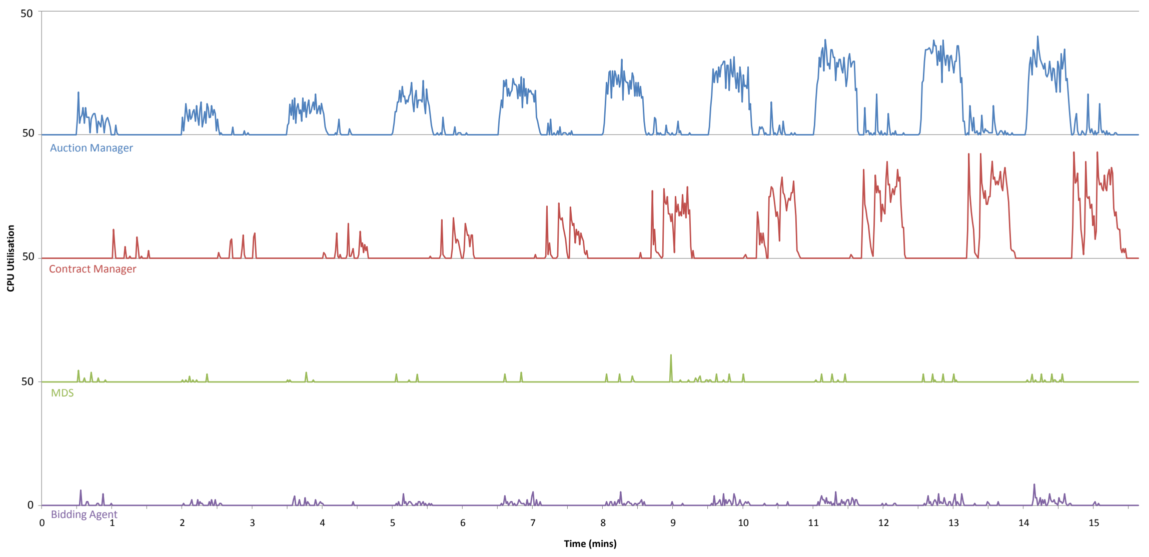


Figure 6.50: CPU Usage for workload B.

exaggerated due to the computation of substitute providers as all auctions are run concurrently, this results in all providers bidding on all 50 auctions when in reality they can only host a single task each. In this case the Contract Manager iteratively attempts to confirm contracts with all potential substitute providers for each job. The total usage is small due to blocking – waiting for substitute computation and bidder confirmation. Figure 6.50 shows a similar maximum peak but the time for which the Contract Manager is running is shorter as fewer bidders bid on auctions that cannot be hosted.

MDS usage for both workloads is minimal, as it is only used to query for 20 registered bidders. Figure 6.50 shows the effect of batching requests as the peaks do not increase substantially even though the number of auctions increases from 5 to 50. Both graphs exhibit increased MDS CPU usage between 8 and 9 minutes, this is due to bidder registration as the bidders in the system are configured to periodically register metadata every 10 minutes.

The Bidding Agent computation is shown to be minimal for both workloads with a maximum of 23.4% and 17.2% for workload A and B respectively. In both workloads the breadth of usage increases as the number of auctions increases, this is due both to bid computation and contract confirmation (with increased substitutes). In this experiment the bidders are configured to use random single good bidding policies irrespective of the current load, which explains the small CPU usage when calculating a bid. Using complex valuation functions and bidding policies would obviously increase this usage.

Memory Usage

The memory usage of each service shown in Figures 6.51 and 6.52 is similar for both workloads, indicating memory increase is related to the number of jobs represented rather than the number of simultaneous jobs. Neither MDS or the Bidding Agent go much beyond their initial allocation with maximum memory usage of approximately 50 MB. This is expected as the Bidding Agent stores no state directly and MDS only stores metadata for 20 registered bidders.

Both the Auction Manager and Contract Manager show increased memory usage with the number of jobs submitted. Memory usage peaks slightly higher in Workload A (125 MB, 85 MB) than Workload B (102 MB, 72 MB). The increase in memory usage is due to the auction and contract state stored by each service. Due to the limited duration of this experiment memory usage does not decrease as resources are not removed within this time frame. DRIVE includes user configurable Resource lifetimes [246] for auction and contract resources, however by default they are both set to one hour which is well beyond this sample range.

I/O Usage

The final aspect monitored is the I/O usage of each service. I/O includes all data transferred between the service and any other device. The measured I/O is the total bytes read and written per second. All service logging and monitoring has been disabled so core I/O usage can be measured. Figure 6.53 and Figure 6.54 show the I/O usage of each service for Workload A and B. IO usage generally models the CPU usage for each service, this is due to the fact that most functionality in the services involves communicating with other services to trigger computation. The same peak in MDS I/O usage for both workloads is evident between 8 and 9 minutes due to

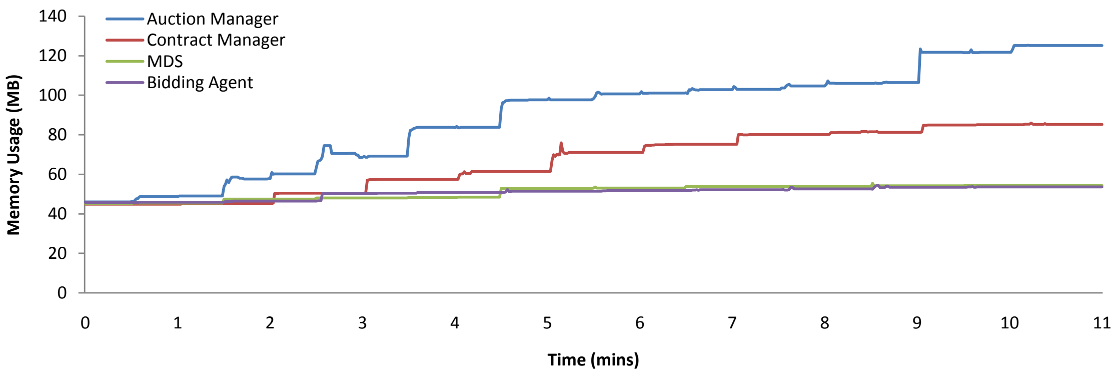


Figure 6.51: Memory Usage for Workload A.

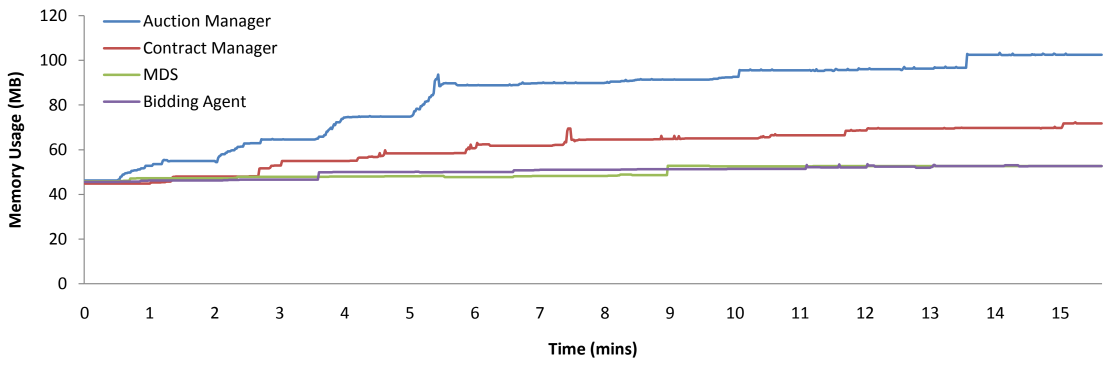


Figure 6.52: Memory Usage for Workload B.

periodic refreshments. The reason for the earlier peaks is unknown however as it corresponds to the first auctions hosted in both workloads, this may be due to indexing or caching.

6.10 Performance comparison with existing technology

Direct performance comparison between DRIVE and existing meta-schedulers is difficult as focuses and goals differ between projects. Chapter 2 summarised the criteria used to evaluate similar meta-schedulers. Many meta-schedulers take an application perspective to evaluation through execution of a specific application and comparison to non-scheduled or simplified approaches. Others take an economic approach to validation by monitoring profit and budget constraints as this is the focus of their research, however this differs from the pricing and penalty focus in this

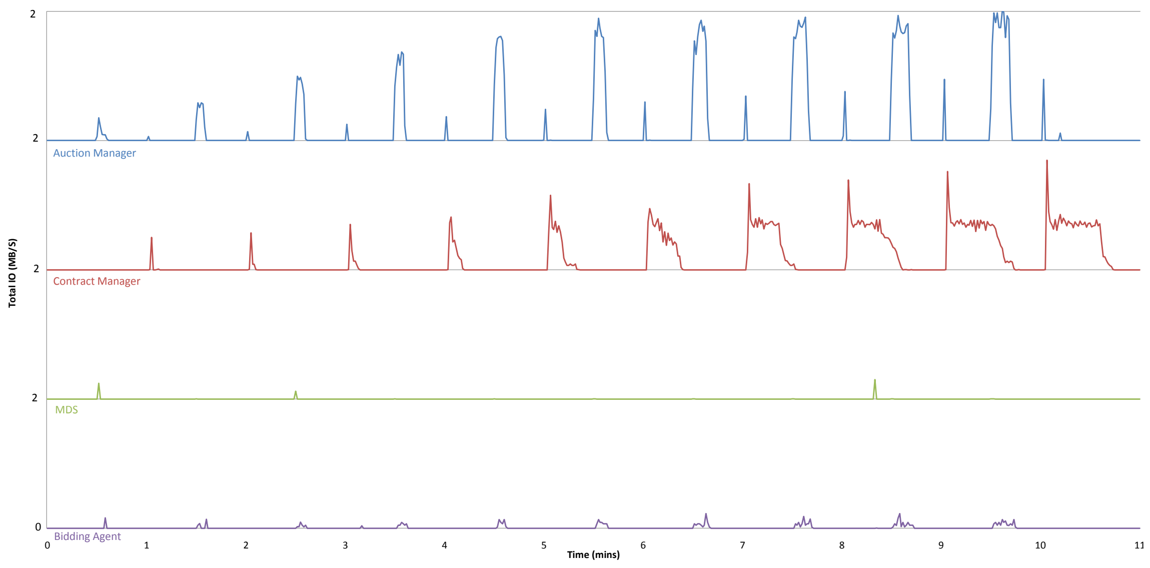


Figure 6.53: I/O Usage for Workload A.

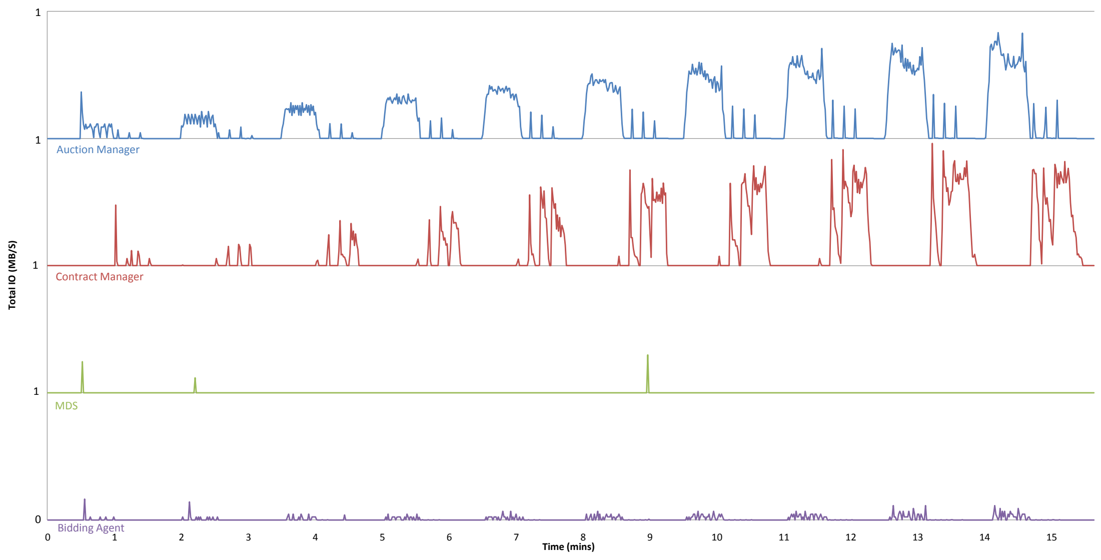


Figure 6.54: I/O Usage for Workload B.

thesis. Another common approach is through the use of simulation environments like GridSim or PlanetSim, however these approaches look at specific attributes like delay and deadlines. Most experiments used to evaluate meta-schedulers are in some way contrived and often results are based on the policies used for scheduling, valuation and bidding.

One metric that is common to all meta-schedulers however, is throughput. While throughput is not a major focus of DRIVE it can be used to compare the overhead of the economic protocols used. GridWay is one of the most commonly deployed performance based Grid meta-schedulers. When configured in a LAN based testbed with 4 providers and a dedicated GridWay server approximately 2,500 jobs can be submitted per minute [247]. In comparison DRIVE can process approximately 300 jobs per minute with 5 providers using a sealed bid second price auction. While this may seem like a drastic difference GridWay uses a centralised allocation structure in which the scheduler sends jobs to hosts based on simple metrics. DRIVE on the other hand conducts fully distributed auctions and creates complete WS-Agreement based SLAs.

Chapter 7

gRAVI

gRAVI (Grid Remote Application Virtualisation Interface)[113, 248, 249] has been developed in collaboration with members of the Computation Institute, University of Chicago and Argonne National Laboratory. The gRAVI toolkit aims to simplify Grid and Web service creation, deployment, and use in a service oriented environment. The toolkit was initially developed to help realise Service Oriented Science (SOS) [112], however gRAVI is much more generic as it provides the ability to wrap arbitrary applications as services and deploy them over a wide range of environments. For this thesis, gRAVI has been extended such that it is the first service wrapping toolkit able to create economically enabled services that can be traded in a DRIVE market. This chapter presents the gRAVI toolkit and describes the integrated DRIVE extensions. The chapter is partially derived from [248].

Service Oriented Architecture (SOA) is a design paradigm designed to encapsulate application functionality behind a well defined self describing service interface, thus abstracting the underlying implementation and technology from users. Service oriented approaches are widely used in distributed environments to abstract physical location and provide parallel execution paths. SOA is particularly valuable in the scientific domain as experimentation can be modularised into small tasks and orchestrated into workflows that model the experimentation process. This service based approach to science, termed SOS, refers to the enablement of scientific research by using distributed networks of interoperating services [112].

Grid and Cloud infrastructures are typically large and complex making exploitation notoriously difficult for users and developers alike. There are substantial barriers for entry as developers need to understand the underlying technologies to create, develop, deploy, and utilise services. Service development typically requires manual writing of service code, configuration files, description files, and deployment scripts for each supported environment and container. Developers must also explicitly manage features such as security, data staging, distributed resource scheduling, allocation protocols, persistence, notifications, and WS addressing. The task of service creation is even more difficult when considering utility models or economic markets such as those created by DRIVE. To participate in a market the service must adhere to the protocols used and implement the required functionality (discovery, advertising, allocation, contracts, and usage). The creation of service oriented infrastructures is not trivial and many users do not have the technical expertise to implement service based applications without large scale investment.

DRIVE is designed to create dynamic service markets, however few applications are web enabled and those that are, are not economically enabled. The difficulty creating services presents a substantial barrier preventing exploitation of DRIVE markets by application providers. To overcome this limitation gRAVI has been extended to automatically generate services with integrated DRIVE capabilities, therefore supporting transparent participation in DRIVE markets. The DRIVE architecture presented in Chapter 4 implicitly assumed separation between traded services and representative DRIVE components as it is unrealistic to assume service providers will alter existing implementations. However this separation is not necessary, in fact integration of DRIVE mechanisms within services has several advantages, for example, it provides tight coupling between the valuation process and service state, reduces security implications as DRIVE components must access service information, provides a single point of contact, and can simplify management. The use of a DRIVE-enabled service wrapping toolkit can drastically reduce the barriers for entry into a DRIVE market, allowing users the ability to rapidly expose a non web-enabled application as a tradable service.

While there are multiple examples of existing Web service wrapping toolkits available none of them meet the specific requirements of DRIVE. Briefly the requirements of a base wrapping toolkit for creating DRIVE-enabled services are:

- Open, standardised, and extensible such that DRIVE extensions can be added to the toolkit
- GUI-oriented to ease use for non-technical users
- Creates WSRF compliant services that meet the requirements of the DRIVE Agent (for example similar interfaces, stateful resources, persistence, WS-notifications, and GSI security) and can be transparently integrated with other DRIVE services
- Creates standardised, customisable, self contained services so that users can alter service implementations and move or deploy to non-proprietary environments
- Creates services that are not bound to a single data staging infrastructure or scheduling model
- Supports a wide range of invocation options, deployment environments, and service containers

The gRAVI toolkit has been developed to satisfy all of these requirements and therefore it provides a suitable base from which to integrate DRIVE functionality. Figure 7.1 shows a high level view of the capabilities of gRAVI. gRAVI provides a GUI-oriented means of rapidly Web enabling existing applications, including service creation, modification and deployment. Using gRAVI any arbitrary application, executable, or script can be wrapped and exposed through a WSRF execution interface. Unlike other wrapping toolkits gRAVI produces independent Web/Grid services that do not require any particular Grid infrastructure or specialised hosting environment. Any gRAVI generated service can be traded in a DRIVE market, scheduled on Grid resources using GRAM, used as a component in a workflow, or deployed to Cloud infrastructure without requiring users to write any code. In addition gRAVI can be used to create strongly typed services from worklfow definitions. All gRAVI services include GSI security, state notifications, persistence, an AJAX based web interface, and a range of data staging options.

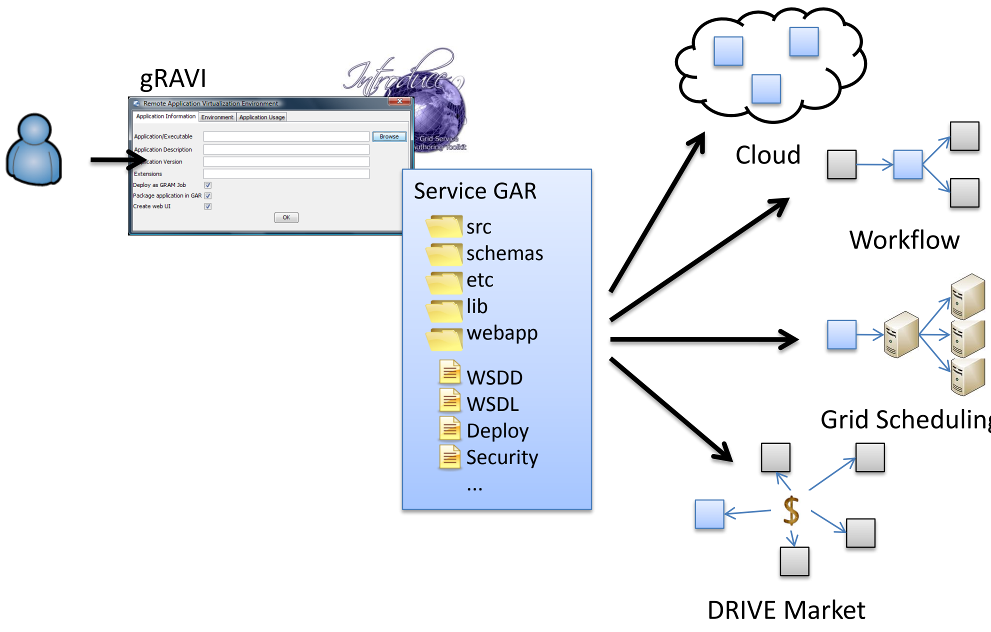


Figure 7.1: gRAVI provides a GUI based mechanism in Introduce to create fully functional WSRF services. The generated service includes all source code, schemas, descriptors, properties, and Web service related files required in the resulting Grid Archive File (GAR). gRAVI also supports deployment to a range of environments including Cloud environments, Workflows, Grid schedulers, and DRIVE markets

To complete this overview of gRAVI two example gRAVI-based infrastructures are presented in bioinformatics and high energy physics respectively. These examples show workflow composition using gRAVI generated services orchestrated using the Taverna workbench [250] and highlight the improved time to deployment using the techniques outlined in this chapter.

7.1 Related Tools

The concept of exposing legacy applications as Web services is not a new idea, there are many examples of service wrapping toolkits and development environments which facilitate rapid service development. Even within the Grid world the idea has been previously explored, the

Java COG kit [75] provided a primitive way to create services using service map files, however this technique is not flexible and lacks scheduling, data management, and security mechanisms. SoapLab [251] uses a set of generic Web services to access applications on remote machines. It uses Apache Axis to create Java based services and CORBA for discovering and starting applications. SoapLab has no support for Grid resources nor any security mechanisms.

Grid Execution Management for Legacy Code Architecture (GEMLCA) [252] provides an environment to expose applications as services. A GEMLCA service translates execution requests and uses Grid scheduling services to execute the application on remote hosts. Applications are described in Legacy Code Interface Description Files (LCID) which can be created using a simple portlet. Unlike gRAVI these services are not independent and cannot be moved easily, execution relies on applications being installed and available on compute nodes and Grid scheduling services are required to execute the application. Another approach to wrapping applications is through the use of dedicated hosting environments such as the Application Hosting Environment (AHE) [253, 254, 255]. AHE is a Perl based hosting environment which exposes unmodified applications as WSRF services on heterogeneous resources in a Grid. This approach differs from gRAVI in that it relies on proprietary middleware deployed to resources in order to deploy applications which therefore limits the mobility of applications. This approach is beneficial within administrative domains as AHE can be deployed locally to add additional transparency to computational resources.

Opal [140, 256] is an application wrapping toolkit designed to wrap scientific applications as Web services without requiring any code from users. Opal offers scheduling to distributed resources, GSI based security, persistence and data management. Opal2 [257] is a complete rewrite of the Opal toolkit which offers various improvements from the first version. In particular it supports third party transfer through HTTP, persistence through a lightweight object/relational service, additional job manager support (GRAM, DRMAA, CSF), and can be deployed inside a virtual machine or as a Rocks rolls for cluster deployment. The task of creating Opal services is limited to writing application configuration files which define the application binary location, arguments, and associated metadata. In order to deploy the service the developer manually alters the default WSDD to point to the appropriate configuration file and gives the service a unique name. Every Opal service uses the same WSDL file as the interface is static and only one method is exposed. This invocation method consists of an XML string defining the arguments with which to invoke the application and any input files (Base64). Each invocation instance receives a job ID that can be used to query status and retrieve outputs. There are several major differences between Opal and gRAVI; gRAVI services are created entirely through a GUI based approach, all gRAVI services support WSRF and use standardised resources to represent invocation and data staging, these resources are used to maintain state, provide persistence and support notifications in simple and standardised ways. gRAVI supports a wide range of third party transfer options and does not require external services or databases to maintain state. Opal also does not provide any customised API, command line invocation tools, or GUI based user interface.

The Generic Service Toolkit (GST) [141, 258] takes a different approach to creating Web service interfaces for legacy applications. GST is a complete infrastructure that includes a service container, discovery mechanism, workflow composer, and a front-end portal. Within the toolkit there is a Generic Factory Service (GFac) that is able to create service interfaces to legacy applications. In GFac a single Web service acts as a request conductor by mapping invocations to the appropriate application module. XML Service Utility Library (XSUL) SOAP libraries are used to provide Web service support. The XSUL message processor performs the mapping from SOAP invocation to a generic handling class, this generic class consults a user defined service map file and invokes the appropriate binary application. Like gRAVI, GFac provides three ways for users to access GFac services; it has an integrated Portlet interface, a shell script to invoke services via the command line, and also a Java API to invoke services programmatically. GFac supports Grid-based systems using Globus GRAM. Security mechanisms are based on GSI with an additional proprietary authorisation system called XPOLA. GFac also supports notifications through a messenger that complies with WS-notification specifications. Unlike gRAVI there is considerable infrastructure required to support GFac services and the generated services are not standard Web services able to be deployed on any container. Due to the non standardised nature of the services, discovery must be performed through the proprietary GST discovery mechanisms. Reliance on XSUL libraries makes widespread interoperability difficult as many organisations prefer to use commodity toolkits such as Apache Axis.

Although there has been much research into market oriented Grids and utility models used in Cloud providers there is no literature relating to integration of economic protocols within services or service wrappers. The DRIVE extensions to gRAVI are unique in that they generate a service with the ability to participate in a DRIVE market without requiring additional code from users. This simplifies the time to deployment required to expose applications in an arbitrary service market therefore reducing the barriers for entry into an economic service-based Grid or Cloud environment.

7.2 Introduce

gRAVI is implemented as a plug-in extension to Introduce [259], leveraging the GUI to guide users through service creation and exploiting the base service creation processes. Introduce is an open source toolkit designed to support development and deployment of strongly typed, secure Grid services.

Introduce exposes a GUI through which developers can create a skeleton WSRF service; including features such as GSI security, metadata registration and WSRF resource properties. Having created a service in Introduce the job of the developer is reduced to implementing the desired functionality in the Java skeleton. Introduce can then be used to deploy the service to a number of commonly used Grid service containers (Globus Toolkit, Apache Tomcat, or JBoss) and allow users to specify index services with which to register.

Introduce follows a three step model of service development:

- **Creation**: Developers describe attributes of the service, such as the name and namespace. A base service is created from which the developer can implement the desired functionality.
- **Modification**: Developers add, remove, and modify service contexts, service methods, resource properties, and security configurations.
- **Deployment**: Any Introduce service can be deployed to a range of supported service containers using the Introduce GUI.

Introduce maintains invocation state through stateful services following the WSRF-ResourceProperties (WSRF-RP) specification. Services implement the factory pattern using a Service Context as a backend service to a primary service. Each context has a single resource type relating to a specific invocation state. When the primary service is invoked it performs an action, creates an instance of the appropriate resource, and returns an Endpoint Reference (EPR) to the client. Using the EPR, subsequent access transparently references the appropriate resource through the respective Service Context. Introduce supports a number of resource framework options such as resource lifetime, persistence and notifications each adhering to the respective WS specification. Introduce creates secure Grid services using GSI to provide both service level and method level security. Secure communication channels can be used and Introduce ensures both client and service have the appropriate functionality to use the channel. Services can use community credentials inherited from the container, delegated credentials from other services, or client credentials using certificates/keys or Grid proxies depending on specific requirements.

The Introduce framework is highly extensible and is designed with a flexible extension plug-in model in which third party developers can create powerful extensions to the framework. Service extensions allow customisation of the service creation and modification steps providing increased control over the development process. Within a service extension new creation and modification GUIs can also be added and custom code can be executed at various stages of the development, modification, and deployment process.

Introduce was chosen as the basis for gRAVI for a number of reasons. In particular Introduce:

- is the most fully functional WSRF development tool available,
- is under active development and is easily extensible,
- provides a GUI based interface which suits non-technical users,
- supports many of the key concepts required in the vision of tradable services such as fine grained GSI security mechanisms, stateful services, persistence, and notifications,
- has a clearly defined model of creation, modification, and deployment which presents users with a range of actions through the same GUI and facilitates extension of services to meet changing requirements.

7.3 gRAVI

gRAVI (Grid Remote Application Virtualisation Interface) allows users to wrap binary applications as secure (tradable) WSRF compliant Web/Grid services without requiring the developer to write any implementation code, description files, or deployment scripts. The resulting service is encapsulated in a standard Grid Archive (GAR) file which is able to be deployed to any WSRF container without any requirement for gRAVI or Grid infrastructure. gRAVI extends the Introduce creation and modification steps adding new code creation processes and specific gRAVI configuration windows within the Introduce GUI, this removes the need for users to run scripts or create/modify any files.

gRAVI services can be automatically economically enabled such that any service can participate in a DRIVE market. The required DRIVE interfaces are generated such that the resulting service supports arbitrary economic protocols and bidding functions using the DRIVE protocol and bidding, plug-in interfaces. All gRAVI services offer blocking and non-blocking methods of invocation as well as the option of submitting workload to Grid schedulers via GRAM. Mechanisms are also included to deploy the service to a range of containers, Cloud infrastructure, or imported directly into a workflow. Each gRAVI service has methods to stage data using GridFTP, caGrid Transfer, base 64 encoded binary data, or HTTP. Additionally each service exposes interfaces to monitor and manage a running application.

Following the Introduce model, gRAVI services implement the factory pattern to create and manage resources. This separation is illustrated in Figure 7.2. The primary service is used to stage data and invoke the application, storing only metadata relating to the service. The secondary context service manages access to the pool of stateful resources and provides monitoring functionality. When the primary service is called the application is started and a resource is created to represent the invocation, subsequent monitoring and control of the application is conducted through the context service. The context service has methods to monitor the process through explicit polling or WS notifications, retrieve standard output, standard error, and files created by the application, kill the running process, and destroy the resource.

7.3.1 Service Creation using gRAVI

When creating a new gRAVI service the developer is guided through a series of customisation windows allowing specification of generic properties such as the service name, target namespaces, service location, and any required Introduce extensions for example caGrid Transfer. This initial configuration is shown in Figure 7.3. A skeleton for the service is created and the gRAVI modifications take place. The developer must then select the application they wish to wrap in the gRAVI specification window (Figure 7.4). There are a number of other options that can be specified at this point including application metadata, Grid scheduling support, packaging the executable in the service archive, and creating a web interface. Grid scheduling is supported by creating GRAM jobs in the gRAVI service and submitting them to a user defined GRAM service

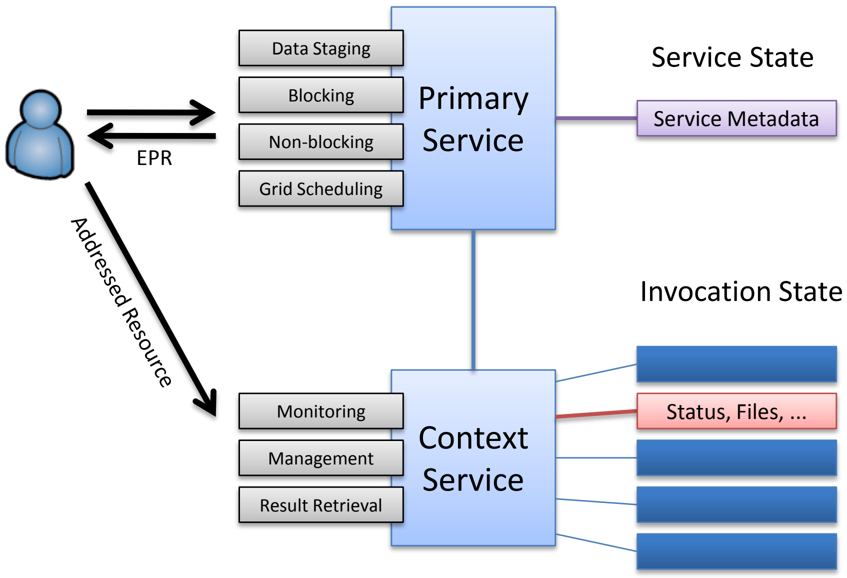


Figure 7.2: Structure of a gRAVI service. The primary service is used to start the application and stage data, it has a single resource which stores service metadata. The context service manages invocation resources exposing management, monitoring, and result retrieval interfaces.

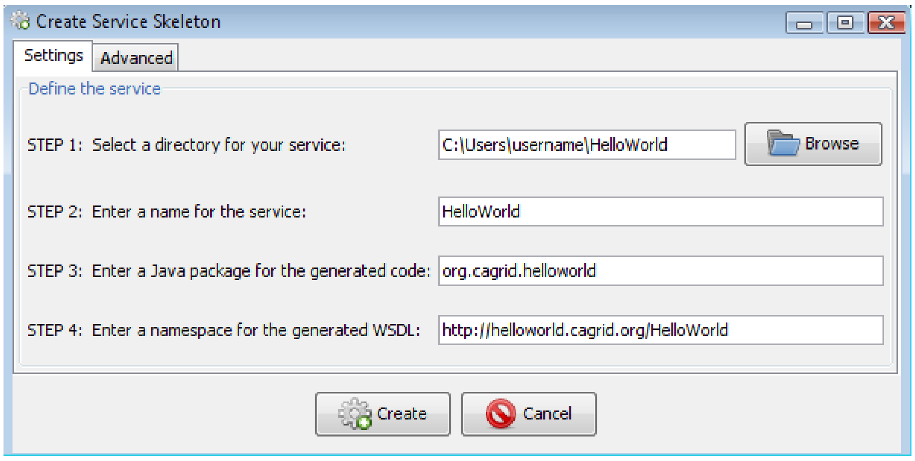
using the GRAMJob API. Packaging the application in the deployment archive is desirable in some scenarios so that the remote machine does not need to have the application pre-installed in order to run the service. However care must be taken to ensure the remote environment is suitable for application execution. If the web interface is selected an AJAX based web application is created, along with the altered deployment scripts required to deploy both the Web application and the gRAVI Web service. The creation process also has additional options allowing users to specify metadata describing the application, input parameters and general application use.

Having specified all requirements gRAVI produces a GAR file that can be deployed to any one of the supported containers. If specified the GAR file includes the application binary which is extracted to the service location when deployed and an associated Web Archive (WAR) file containing the web interface. All Web service artefacts such as deployment scripts, descriptors, WSDL, WSDD are created automatically.

7.3.2 DRIVE Extensions

While economic resource allocation has long been proposed as an efficient means of allocating resources, very few production environments have adopted economic principles. One of the major reasons for this limited adoption of economic allocation in distributed systems is the lack of economically enabled services and the difficulties creating such services.

gRAVI includes a set of DRIVE extensions that enable the creation of an economically aware



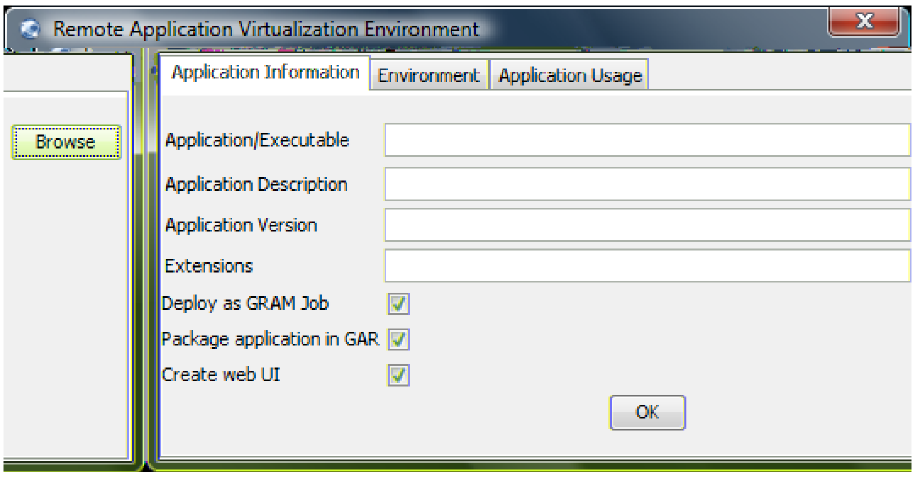
STEP 1: Select a directory for your service: C:\Users\username\HelloWorld

STEP 2: Enter a name for the service: HelloWorld

STEP 3: Enter a Java package for the generated code: org.cagrid.helloworld

STEP 4: Enter a namespace for the generated WSDL: http://helloworld.cagrid.org/HelloWorld

Figure 7.3: gRAVI initial service configuration.



Application/Executable

Application Description

Application Version

Extensions

Deploy as GRAM Job ✓

Package application in GAR ✓

Create web UI ✓

Figure 7.4: gRAVI service construction.

DRIVE-enabled service. These extensions allow the service to participate in a DRIVE market by implementing the required methods for service registration, economic negotiation, contract creation/management, and execution. DRIVE specific parameters can be configured using the associated properties file without modifying any service code. Each service includes registration metadata that can be configured by the user and periodically updated to represent the current capacity of the service. This metadata conforms to the prototype DRIVE resource profile schema and allows the service to be discovered and considered in the allocation process. The generated service implements a bidding interface that facilitates bid generation and submission following the auction protocols defined in the DRIVE prototype. The bidding process uses plug-in user defined DRIVE bidding policies and valuation functions to create a bid for a specific request.

The complete DRIVE contract interface is included to confirm or reject contracts resulting from winning an economic negotiation. Finally the generated gRAVI service also includes a DRIVE execution interface to start the application when required. This interface is designed to ensure only allocated users can access the service and like the DRIVE Execution Service (Section 5.3.5) requires presentation of the contract to invoke the application.

The generated gRAVI service essentially includes a DRIVE Bidding Agent. Each generated service includes the DRIVE Bidding Agent plug-in interfaces for economic protocols, bidding policies, and valuation functions (Section 5.3.1). Users can therefore use existing plug-ins or implement their own in a Java jar file. The location of plug-ins are specified in the gRAVI service properties file and the appropriate class is dynamically loaded at run time in the same way as the Bidding Agent. The valuation function is responsible for parsing the task description, validating that it conforms to the provider requirements/schema (in the prototype implementation this is JSDL), checking capacity, and computing a bid. The plug-in bidding protocol is used to convert a string representation of a bid into the protocol specific language, these plug-ins are developed by protocol developers and shared amongst participants in the same way as they are in the DRIVE prototype.

The current DRIVE extensions to gRAVI act as a proof of concept verifying the feasibility of creating DRIVE-enabled services. However, various improvements are required to fully incorporate DRIVE functionality in gRAVI services. In particular, economic participation is currently limited to push advertisements from the Auction Manager and the service does not support advanced reservations or domain specific capacity monitoring. Further extension is required to include reverse discovery and integration with the DRIVE reservation service to schedule advanced reservations. Capacity monitoring is currently based on system utilisation, however in many service-based domains this is not a good measure of capacity. Therefore a generic capacity monitoring interface must be created and incorporated such that users can define metrics used to indicate capacity in different domains.

7.3.3 gRAVI Service Features

Persistence and Notifications

One major advantage when using WSRF Web services is the ease with which developers can provide notifications and persistence. All gRAVI services are created with customised code to allow users to register for (and receive) state change notifications. This functionality is particularly useful when using asynchronous invocation or Grid scheduling as the application can be started and the consumer is able to receive notifications of state changes asynchronously. In distributed environments persistence of particular information is also important in the event of failure. All gRAVI services persist information relating to the invocation, including input/output files and the state of the application. While gRAVI does not explicitly maintain application state if the machine fails, it will keep information which may be used to determine where the application stopped. Unlike other wrapping toolkits persistence is implemented using standard WSRF resource persistence and therefore avoids the need for cumbersome administration dependent infrastructure such as databases.

Data Staging

Typically scientific applications have requirements for large data staging capabilities. Scientific communities also commonly have existing data transfer infrastructure available to move large amounts of data. gRAVI is designed both to quickly create services that facilitate small scale transfer for individual users while also providing the ability to leverage existing infrastructure for large scale transfers as is often seen in scientific collaborations.

gRAVI supports data transfer of base 64 encoded binary objects without any requirement for specialised infrastructure. These transfers are suitable for small scale movement (less than 10MB) as the overhead when wrapping binary objects in SOAP is substantial and does not scale. For larger data movement HTTP, GridFTP, and caGrid Transfer are supported. The use of GridFTP is a deployment property in which available GridFTP services are specified to the gRAVI service. caGrid transfer can be included in any gRAVI service as an extension in Introduce, this staging uses a Servlet to perform data transfer parallel to the gRAVI service.

Security

gRAVI makes use of GSI based security leveraging the existing support in Introduce. The Introduce security infrastructure supports authentication and authorisation through Grid credentials and can be customised for each service. Authorisation can be granted at the service or method level depending on requirements. Currently access to GRAM and GridFTP in gRAVI services requires the use of community credentials at the container level.

Deployment

Each gRAVI service is a completely independent entity that can be moved and deployed on any supported Web service container. Each service has deployment configurations and scripts that are used to deploy the service to these containers, alternatively the introduce GUI can be used to guide the user through the deployment process. If a web interface is selected it is also deployed automatically to a separate Servlet container when the gRAVI service is deployed.

When deploying the service the user may use the Introduce GUI (or configuration files) to alter service names and specify registration parameters. The standard Globus MDS Index Service is supported for registration. Introduce based deployment supports user defined periodic refreshments if required. Application metadata specified in the gRAVI service creation process along with DRIVE resource profiles are also registered with MDS in order to allow clients (or DRIVE services) to discover services. gRAVI specific properties can be customised by altering the gRAVI properties file. The properties file includes application properties, DRIVE properties, GRAM servers, GridFTP servers and credentials to be used on the Grid. Another unique deployment option in gRAVI is the ability to deploy services to the Cloud this is discussed further in section 7.3.5.

Invocation

gRAVI offers a number of mechanisms to invoke services; each service includes a Java Client API specific to the service, a customisable web interface, and a generated command line interface. The Java API is customised for the service and includes example use of the major functionality of the service. The customisable web interface is AJAX based and can be easily modified with limited web development knowledge as all the AJAX handling is abstracted in a generic JSP class and the look and feel of the page can be altered using CSS/HTML. With each of the invocation methods users can stage input data using any of the supported methods, start the application in blocking, non-blocking, and Grid scheduling modes, pass appropriate input parameters, monitor the status of the application including stdin and stdout, retrieve output files, kill the running application, and destroy the resource.

Each gRAVI service is a standalone Web service and as such it can be discovered and used by third party applications and workflow engines. In order to transparently exploit the WSRF resources used in gRAVI services the external client application must support WSRF WS-Addressing. The Taverna workbench has been extended such that it can transparently utilise gRAVI services. When a gRAVI service is invoked it creates a unique resource to maintain state about the invocation, clients identify the particular instance using the unique key of the resource, this key is encapsulated in a reference object making access to the resource transparent. If the client application does not support WS-addressing the gRAVI service methods can be altered to include explicit resource addressing information in the invocation as is shown in [113].

7.3.4 Scientific Workflows using gRAVI

A major focus of the design of gRAVI has been the vision of using services in scientific workflows and ultimately, realising Service Oriented Science. Scientific workflows enable scientists to collect and analyse data in a web-scale manner, rather than doing it in a laboratory. The adoption of scientific workflows enables scientists to publish work alongside the data and experimental procedures used to obtain their results. Scientists can even publish the workflows which describe their research procedure as orchestrated services, further facilitating scientific collaboration, extension and verification. An example of this vision is the caGrid project that currently holds over 100 services related to cancer research and uses both Taverna and BPEL [260] to orchestrate services. Another example is the MyExperiment platform in which scientists publish workflows that perform and share various bioinformatics experiments.

There are two factors that hamper the adoption of workflow technology in SOS. The first is that, many legacy applications are not web enabled; the second, workflows which represent scientific exploration procedures are not easily shared because each workflow has its own format and depends on specific execution mechanisms. gRAVI provides a solution to overcome both of these limitations. First, gRAVI wraps individual applications as services which are consumable by workflows; secondly, gRAVI can also be used to wrap workflows as a service so that scientists can easily reuse the workflow without having the execution engine installed in their own environment.

Exposing workflows as services provides a simple way to share (or trade) workflows with others and also leverage computational resources by executing the workflow remotely. Workflows are often created and shared with others to verify or build upon existing research and may be used independently or possibly as sub-workflows in larger scientific workflows. While gRAVI can be used to wrap workflow execution scripts directly there is a vast amount of information contained in workflow definition files that can be used to further customise the service. Typically the workflow definition contains information on each processor and data flow. The information regarding input and outputs can be extracted and used to create strongly typed interfaces to workflow services. Strongly typed services are favourable as they can be easily added to other workflows without requiring extra data flow parsing between processors.

gRAVI-t (gRAVI for Taverna) is an extension to gRAVI that parses Taverna workflow definitions and creates fully functional strongly typed WSRF services. The executeWorkflow method signature is constructed using the input port names and data structures, the output ports each map to a gRAVI service resource variable which is able to be retrieved using an individual method (there is one output method per output port). When the service is invoked the input parameters are collated and the workflow is executed using the Taverna API. All output files are collected in the gRAVI working directory. The output files map to a single output port and are read and returned to the consumer using the appropriate output retrieval method. This approach provides a simple way for users to take a Taverna workflow and expose (or sell) it to the wider community as a fully functional and independent Web service.

7.3.5 gRAVI Services in the Cloud

Cloud environments provide a scalable platform in which resources can be dynamically provisioned to meet requirements. By deploying applications in a Cloud, an application provider can avoid infrastructural costs associated with running a datacenter and offload problems such as scalability, availability, and fault tolerance to the Cloud provider. Like Grid platforms the task of creating and deploying services to the Cloud is difficult. gRAVI goes some way to simplifying these problems by creating command line deployment scripts for deploying services to Science Clouds. Science Clouds provide an open Cloud environment for sharing computing cycles in scientific communities, one example is Nimbus.

gRAVI services include a collection of Ant scripts that contextualise an existing Nimbus VM image and deploy the gRAVI service. Contextualisation of VMs requires importing a collection of libraries to configure the container on the image before deploying the service. The tools are dependent on utilising an SSH identity key to access the instance. By default during VM creation the Nimbus client sends the users default public SSH Key to the Nimbus service, the service in turn inserts this key into the root users *authorised keys* therefore enabling remote access to the running VM instance.

Client operations are executed by using standard gRAVI invocation tools, secure interaction with the deployed services requires use of valid local credentials. These tools provide a proof of concept mechanism for deploying gRAVI services to a Cloud.

7.4 Example Applications

gRAVI has been used in a number of example applications in a variety of scientific fields. Two such examples of differing use are presented in this section: first a workflow composed of gRAVI services geared toward finding functionality of sequences in the human genome; and secondly the gRAVI-based processing stages used in a tomography system for high energy physics.

gRAVI was used to wrap several independent applications used in the transposon bioinformatics project discussed in Chapter 3. The services were then combined to form a Taverna workflow modelling the process. The workflow is based on a multi-step comparative analysis performed between a genome inserted with mutant libraries and the original genome. The sites of random insertions can be sequenced to determine gene function. In this workflow sequences are iteratively compared against known databases in order to compare sequences. The process of wrapping the separate applications and orchestrating a workflow was shown to work with minimal developer intervention. The resulting workflow replaced the script based approach previously used and added modularity, scalability, and a GUI based perspective of the system. This workflow highlights the integration of gRAVI services and Taverna as a proof of concept. Further information about this workflow and the process of wrapping the applications using gRAVI is described in [113]

Tomography at the Advanced Photon Source (APS) uses x-rays fired over a range of angles to create projections through a sample. The projections are analysed to determine how the x-rays are absorbed through the sample. This data is processed and collated in a 3 dimensional density map called a *reconstruction* which is further analysed. The application produces large sets of data – typically the sample includes 1440 images which is reconstructed into a 3D image of approximately 35 GB. There is also a large computation burden incurred to create a single reconstruction on the order of hundreds of CPU hours. The process of creating reconstructions has been implemented in a Service Oriented manner at the APS using gRAVI. gRAVI was used to wrap many components of the workflow, specifically the parallel reconstruction stage which makes use of GRAM scheduling, the standalone cross-correlation code, data rescaling and other stand alone applications. Originally some services were developed without the use of gRAVI, the comparison between the two approaches show that time to deployment for one simple service without using gRAVI was over 1 month, the deployment time was much improved when using gRAVI with less than 2 days for a service with data transfer, security, status notifications, and GRAM scheduling [261]. This example highlights the successful use of gRAVI in a more complex environment involving large scale data movement using GridFTP, scheduling to existing clusters using GRAM, and GSI security for authentication and authorisation.

7.5 Summary

The task of developing Service based architectures is inherently complex due to the nature of the underlying infrastructure. Developers are expected to master a range of technologies, languages, and tools in order to create, deploy, and trade a service in a Grid environment. With the addition of WSRF, security configurations, Grid scheduling, economic protocols, data staging, web based user interfaces, and other requirements such as persistence and notifications the time to deployment for relatively complex services is immense. gRAVI provides a way to greatly reduce the cost of creating such services by simplifying the development and deployment process. Using gRAVI, creation and deployment is conducted automatically without requiring the developer to manually write any code, scripts, or files as all configuration can be done through a GUI. The DRIVE extensions to gRAVI can be used to overcome the difficulties creating and trading applications in an open market. gRAVI provides the ability to automatically generate elastic, metered DRIVE-enabled services, thereby reducing the task of developers to customising simple DRIVE plug-ins. This approach makes the DRIVE infrastructure available to a wide range of users without requiring in depth knowledge of services, DRIVE infrastructure, or the protocols used in the market. To our knowledge this is the first such example of a wrapping toolkit capable of producing economically enabled services.

Unlike similar wrapping toolkits gRAVI produces a completely independent service that is able to be deployed on any compatible container making use of standard Grid registration, discovery, and security mechanisms. Existing data movement infrastructures such as GridFTP and caGrid transfer can also be exploited to optimise large scale data movement. By leveraging Introduce, gRAVI is able to create these services following a GUI-based approach. In order to create widespread adoption of SOS and service based economies, tools such as gRAVI are required to simplify the task of development for users. Using gRAVI a user can rapidly expose an application, data set, or workflow as a service and share or trade the service with others either independently or orchestrated into a workflow. gRAVI has been used successfully in a number of projects and has been shown to drastically reduce deployment time for service oriented infrastructures.

Chapter 8

Social Cloud Computing

The concept of a Social Cloud was developed in collaboration with members of the Distributed Collaborative Computing Group at the University of Cardiff. The Social Cloud architecture and implementation are based on the DRIVE prototype presented in Chapter 5. Due to the different focus of the service-based Social Cloud it has been included as a separate chapter. The following chapter is partially derived from [262].

DRIVE is designed to create open service markets in which users trade access to arbitrary services in any domain. To demonstrate this flexibility a unique Social Cloud computing environment has been developed utilising DRIVE to create an open market in which users trade Cloud services amongst friends in a Social network. This chapter defines a *Social Cloud* - a Cloud computing environment which allows dynamic resource sharing between members of a Social Network (friends). A prototype Social Storage Cloud has been implemented as a proof of concept using DRIVE to create an open storage market. The initial DRIVE prototype has been altered and extended to represent Cloud requests rather than Grid jobs. Specifically the description/term language is based on EJSDL (rather than JSDL) with extensions for Cloud QoS parameters (Availability, Error rate). The Bidding Agent has been altered so that it interacts with a generic Storage service to determine capacity and pricing. A new resource profile has been developed to represent typical Cloud service capabilities for registration and discovery. Facebook credentials are used for authorisation and VO management rather than the CA/VOMS approach taken in the DRIVE prototype. Finally the co-op nature of DRIVE is not used in this deployment, rather DRIVE services are deployed on dedicated infrastructure to serve the entire community.

8.1 Introduction

Social networking has become an everyday part of many peoples lives as evidenced by huge individual user communities. Some Social network communities even exceed the population of large countries, for example Facebook has over 500 million active users[1]. Social networks provide a platform to facilitate communication and sharing between users, therefore modelling real world relationships. Social networking has also extended beyond communication between friends, for instance, there are a multitude of integrated applications and some organisations even utilise a users Facebook credentials for authentication rather than requiring their own credentials (for example the Calgary Airport authority in Canada uses Facebook Connect[2] to grant access to their WiFi network).

The structure of a Social network is essentially a dynamic virtual organisation with inherent trust relationships between friends. The trust established can be used as a foundation to share resources (information, hardware, services) in the context of a Cloud. This architecture is termed a Social Cloud. Cloud environments are typically focused on providing low level abstractions of computation or storage. Computation and Storage Clouds are complementary and act as building blocks from which high level service Clouds and mash-ups can be created. Storage Clouds are particularly valuable as they offer a way to extend the capabilities of storage-limited devices such as phones, provide access to data from anywhere, and can be used to backup data. As such this chapter focuses on the creation of a proof of concept Social Storage Cloud that facilitates dynamic storage trading between Facebook friends.

A Social Cloud is a scalable computing model in which virtualised resources contributed by users are dynamically provisioned amongst a group of friends. Compensation for use is optional as users may wish to share resources without payment, and rather utilise a reciprocal credit (or barter) based model [263]. In both cases guarantees are offered through customised SLAs. In a sense, this model is similar to a Volunteer computing approach, in that friends share resources for little to no gain. However, unlike Volunteer models there is inherent accountability through existing friend relationships. There are a number of advantages gained by leveraging Social networking platforms, in particular access to huge user communities, exploitation of existing user management functionality, and reliance on pre-established trust formed through existing user relationships. This chapter outlines our vision of, and experiences with, creating a prototype Social Storage Cloud using DRIVE to facilitate an open service market.

8.2 Background

The concept of a Social Cloud pulls together multiple threads of current distributed computing research. Social networking and Cloud computing concepts form the basis for this research, however concepts from Volunteer computing, economic resource allocation, and Service Level Agreements (SLAs) are also used. Cloud computing has grown rapidly due to the publicity surrounding large scale providers. In addition, there are a multitude of commercial Cloud providers such as Amazon EC2/S3, Google App Engine, Microsoft Azure and also many open Clouds like Nimbus and Eucalyptus.

[1]`http://www.facebook.com/press/info.php?statistics`
[2]`http://www.developers.facebook.com/connect.php`

There are many instances of Social network and Cloud computing integration. However, these projects either use a Cloud platform to host an entire Social network or utilise Cloud services to create scalable applications within the Social network. For example, Facebook users can build scalable Cloud based applications hosted by Amazon Web Services [264]. There is no prior literature related to creating a Cloud infrastructure leveraging Social networking as a means of dynamic user management, authentication, and user experience.

Automated Service Provisioning ENvironment (ASPEN) [265] takes an enterprise approach to integrating Web 2.0, Social networking and Cloud Computing by exposing applications hosted by Cloud providers to user communities in Facebook. The focus of ASPEN is exposing applications and sharing data within an enterprise through an intuitive and integrated environment. Clusters are used to provide a scalable Cloud infrastructure for their demonstrator application. Application participation is by invitation only, as it is aimed at individual enterprises. There are similar efforts in the Grid community to leverage Social networking concepts, communities, and mechanisms. For example PolarGrid [266] extracts Social data using the OpenSocial [267] interface and relies on OpenID [268] for identification. Different Social networking functions (friendship, groups, information sharing) are then incorporated in an application specific portal.

An alternative approach involves building a Social network around a specific application domain such as MyExperiment [3] for biologists and nanoHub [4] for the nanoscience community. MyExperiment provides a virtual research environment where collaborators can share research and execute scientific workflows remotely. Similarly nanoHub allows users to share data as well as transparently execute applications on distributed resource providers such as TeraGrid and OSG. These platforms are focused on the communities they serve and lack the sizable user bases of Social networking platforms, however they highlight the type of collaborative scientific scenarios possible in such networks.

The concept of sharing resources in a Social Cloud is similar to a Volunteer computing model. Volunteer computing is a distributed computing model in which users donate computing resources to a specific (academic) project. The first volunteer project was the Great Internet Mersenne Prime Search[5] in 1996, however the term gained much exposure through the SETI@Home [269] and Folding@home [270] projects in the late 90's. These projects showed the enormous computing power available through collaborative systems. One of the most relevant Volunteer computing efforts is Storage@Home [271] which is used to back up and share huge data sets arising from scientific research. The focus of Volunteer computing has since shifted towards generic middleware designed to provide a distributed infrastructure independent of the type of computation, for example the Berkeley Open Infrastructure for Network Computing (BOINC) [35]. Most Volunteer platforms do not define SLAs, typically users are anonymous and are not accountable for their actions (they are rewarded with different incentives however). In a Social Cloud context this does not suffice as users need to have some level of accountability. A more realistic model for this type

[3]http://www.myExperiment.org
[4]http://www.nanoHub.org
[5]http://www.mersenne.org

of open sharing is a credit based system in which users earn credits by contributing resources and then spend these credits when using other resources. This type of policy is used in systems such as PlanetLab [272].

8.3 Application Scenarios

Social Clouds provide a base platform from which various applications and scenarios can be realised. In particular Social Clouds may be especially valuable to the scientific community for sharing not only information but also resources (computation, storage) in VOs. Increasingly scientific collaborations are turning to social networking concepts to share information and resources within user communities for example MyExperiment and nanoHub. However, these approaches are limited as administrators need to create and manage proprietary social infrastructures and users require credentials for each network they participate in. The same functionality can be realised using a Social Cloud deployed in an existing Social network. Social Computational Clouds can be used to share large scale resources, execute workflows, or perform experiments in such communities. Social Storage clouds can be used to store/share data and information (for example academic papers, scientific workflows, datasets, and analysis). Social Software Clouds extend the concept of Service Oriented Science, by sharing access to particular scientific applications within a community for collaboration, extension, or verification of research. A Social Cloud approach is advantageous as there is no requirement for dedicated infrastructure or management, few barriers for entry for scientific communities, and users can leverage existing Social network credentials.

Without extension Social Storage Clouds provide a tangible use for users, allowing them to share, store, back-up, or replicate data and access it anywhere. Transparency could be increased by accessing storage through a traditional operating system by mapping the storage Cloud as a network drive. One obvious use for Social Storage is storing and sharing photos. Online photo storage is increasingly popular due to the size of photos, requirements to share with with friends, and cameras/phones with internet connectivity. While most Social networks already store photos, the burden for hosting them could be moved from the network to the distributed network of users to increase scalability and reduce infrastructural requirements. Essentially creating a distributed Flickr-like[6] Cloud service based upon a Social Storage Cloud. The security implications of such an infrastructure are limited as photos are commonly shared with friends.

8.4 Architecture

A Social Cloud provides a way for users to trade virtualised resources amongst one another utilising the relationships represented in the Social network as a basis for trust. The architecture presented in this section is implemented as a Facebook application as Facebook is the most widely

[6]`http://www.flickr.com`

used Social network and has a comprehensive application development environment and API. A similar architecture could be realised using other Social networks or OpenSocial.

At a high level a Social Cloud offers transparent access to a generic type of service provided by users. The Cloud must also allow users to select the services with which they are willing to interact. This decision may be made for any number of reasons, for example price, capacity, location, latency, or relationship with the user. In a Social Cloud, services can be mapped to particular users through Facebook identification, this allows for definition of unique policies regarding the interactions between users. For example, a user could limit trading with close friends only, users in the same country/network/group, all friends, or even friends of friends.

A credit-based system is used to encourage contribution to the system, so users must contribute resources in order to be able to use resources. This virtualised currency creates a *closed* economy which requires book balancing to ensure validity. A specialised banking component manages the transfer of credits between users whilst also storing information relating to current reservations. Contracts are established to represent agreed upon service levels and multiple market mechanisms may be used concurrently in such a model. A high level architecture of a Social Cloud is shown in Figure 8.1.

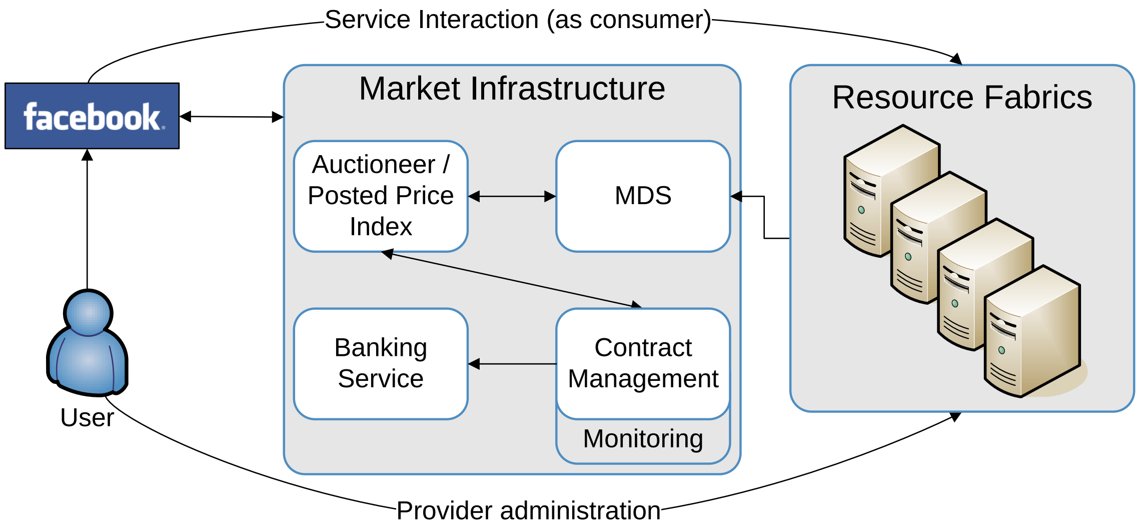


Figure 8.1: Social Cloud Architecture. Users register Cloud services and friends are then able to provision and use these resources through the Social Cloud Facebook application. Allocation is conducted by the underlying DRIVE-based market infrastructure(s).

8.4.1 Facebook Applications

Facebook provides a comprehensive application development environment. The Facebook API uses a REST-like interface and is therefore accessible in multiple programming languages. The

API includes methods to get a range of data from the Facebook server such as friends, events, groups, application users, profile information, and photos. Facebook Markup Language (FBML) enables the creation of applications that integrate completely with the Facebook "look and feel". Integration points include the profile page, an application canvas, or in feeds. FBML is a subset of HTML with proprietary extensions (for example *fb:user* – given a user ID will display the user name and provide a link to their profile). Facebook JavaScript (FBJS) is Facebook's version of JavaScript – rather than sandboxing JavaScript, FBJS is parsed when a page is loaded. The parsing process ensures a virtual scope for every application by prepending the application ID to any JavaScript identifiers. Using these techniques developers can create integrated Facebook applications with access to diverse Social network information.

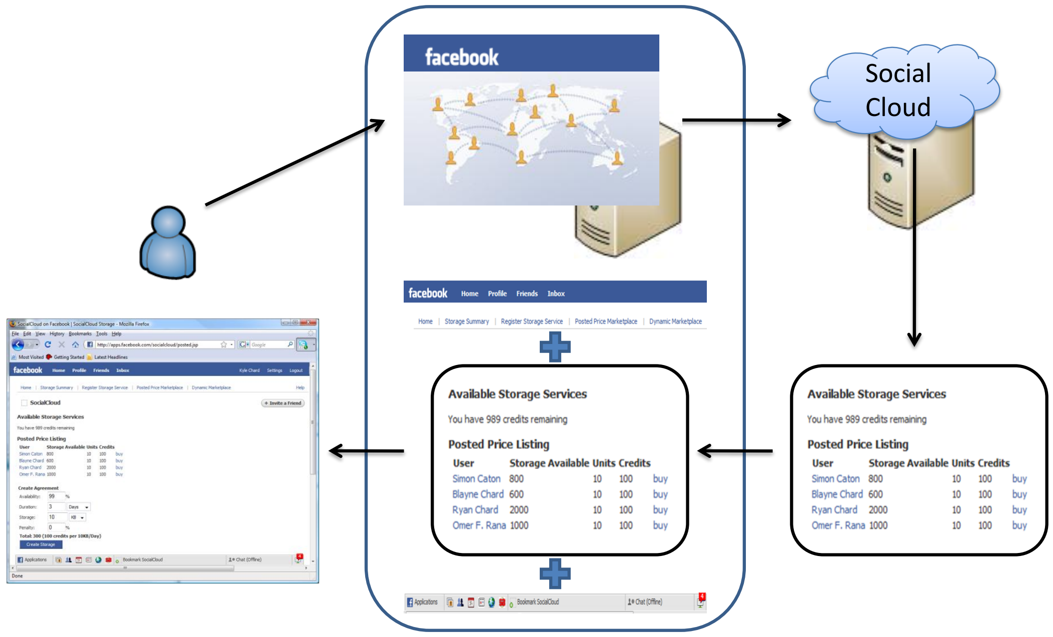
Facebook applications are hosted independently not within the Facebook environment. A Facebook canvas URL is created for user access, this URL maps to a user defined callback URL which is hosted independently. The process of rendering an application page is shown in Figure 8.2. When a page is requested by the user through the Facebook Canvas URL (`http://www.apps.facebook.com/socialcloud/`) the Facebook server forwards the request to the defined callback URL. The application then creates a page based on the request and returns it to Facebook. At this point the page is parsed and Facebook specific content is added according to the FBML page instructions. The final page is then returned to the user. This routing structure presents an important design consideration in a Social Cloud context as access to the Cloud services would be expensive if routed through both the Facebook server and the callback application server in order to get data from the actual Cloud service. An AJAX model can be used to reduce this overhead by using FBJS to request data asynchronously from the specified service in a transparent manner without routing through the application server.

8.4.2 Virtualised Resources

Cloud computing relies on exposing virtualised resources as a service in a metered and elastic manner. In a Social Cloud context this service could represent *any* resource that users may wish to share, ranging from low level computation or storage services through to high level application services or service mashups such as photo storage. There are two generic requirements of this service: firstly, the interface needs to provide a mechanism to create a stateful instance for a reservation. In the Social Cloud model an agreement is passed to the service which is parsed and instantiates the required state. Secondly, in order to be discovered the service needs to advertise capacity so that it can be included in the market. Following the mechanisms used in DRIVE, advertised capacity is XML based metadata which is periodically refreshed and stored in Globus MDS.

8.4.3 Banking

A credit-based system has been implemented to reward users for contributing resources and charge users for consuming resources. The banking service is therefore required to regulate vir-


facebook
Social Cloud
facebook Home Profile Friends Inbox
Home | Storage Summary | Register Storage Service | Posted Price Marketplace | Dynamic Marketplace
Applications
Bookmark SocialCloud
Chat (Offline)

Available Storage Services

You have 989 credits remaining

Posted Price Listing

User	Storage Available	Units	Credits	
Simon Caton	800	10	100	buy
Blayne Chard	600	10	100	buy
Ryan Chard	2000	10	100	buy
Omer F. Rana	1000	10	100	buy

Figure 8.2: Facebook application hosting environment. The Social Cloud web application generates page content which is parsed by Facebook to create the page delivered to the user.

tual currency and facilitate transfer. The banking service registers every member of the cloud and stores their credit balance and all agreements they are participating (or have participated) in. Credits are exchanged between users when an agreement is made, prior to the service being used. To bootstrap participation in the Social Cloud, users are given an initial number of credits when joining the Cloud. While suitable for testing of the prototype Social Cloud, this initial credit policy is susceptible to inflation (adding users with initial credits devalues currency) and cheating (if fake users are created and the initial credits are transferred).

Currently there is no mapping between the credits users may purchase via Facebook and the credits used in the Social Cloud. A production version of a Social Cloud could use Facebook credits, however as Facebook credits are not presently convertible, there may be insufficient reward for storage providers. The use of a real currency on the other hand, would not as easily integrate into Facebook, but would provide better incentives. In any case, either option resolves the problem of cheating (as noted above), and obviates the need to regulate a closed economy [101].

8.4.4 Registration

Registration within a Social Cloud follows the same model as DRIVE registration – users first register themselves and then specify the Cloud services they are willing to trade. Figure 8.3 shows the registration process. This process assumes a user has already registered and authenticated with the Social network. The user registers with the banking service and registers associated services with MDS. Due to the fact users are pre-authenticated through Facebook, user instances can be transparently created in the banking service using the users Facebook ID. Having registered with the banking service, the user is presented with an MDS EndPoint Reference (EPR) and Cloud ID which they use to configure their service for registration (and refreshment) of resource capacity and pricing information. DRIVE market services utilise the MDS XQuery interface to discover suitable services based on user IDs and real time capacity.

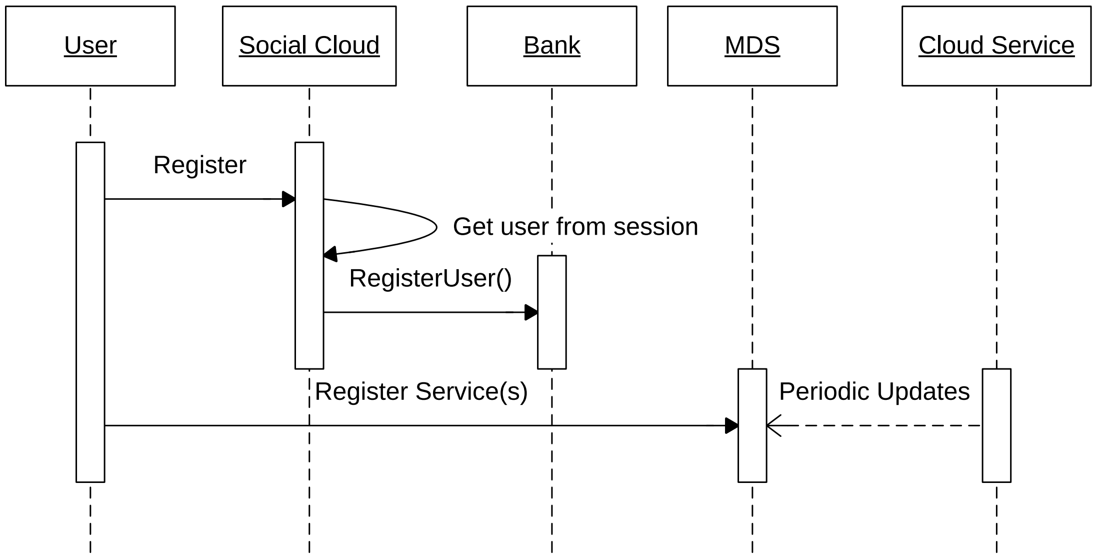


Figure 8.3: Registration in a Social Cloud.

8.4.5 Service Marketplaces

Social Cloud service usage and user credits may be exchanged within a DRIVE marketplace. The Social Cloud architecture includes two concurrent economic models used for trading: Posted Price and Auctions. *Posted Price* markets are the predominant model used by current Cloud providers. This model allows users to select a service (or service level) from a list of static offers. In a dynamic system like a Social Cloud this requires manual configuration to establish market prices and obtain supply and demand equilibrium. More dynamic mechanisms such as *Auctions* are able to quickly determine a market price based on local policies and valuation functions, however they are more computationally expensive. Both market mechanisms provide ways to create a contract between users which can then be used to instantiate the service provision and transfer credits between participants. DRIVE is used to facilitate the open market and create WS-Agreement based contracts as a result of a trade.

Posted Price

In a posted price model providers advertise offers relating to particular service levels for a predefined price or following a linear pricing function, consumers are then able to fine tune specific parameters to create a SLA. Creating such a market requires coordination between a number of the Social Cloud components to; discover Cloud services, create agreements, and transfer credits. Figure 8.4 shows the flow of events for a posted price trade in a Social Cloud. When a user requests posted price offers the Cloud application uses the user ID (from the session) to check the user is registered in the bank and they have sufficient credits available. A list of all the users friends is generated using the Facebook API, this list is used to compose a query to discover friends' Cloud services from MDS. The result of which populates the offer list that describes availability and pricing information. When the user selects a Cloud service, the Social Cloud application creates a contract which it sends to the Cloud Service. Assuming both parties accept the agreement it is then passed to the Bank to transfer credits between users.

Auctions

In an auction-based market trades are established through a competitive bidding process between users or services. Figure 8.5 details the auction process using common DRIVE services. A list of friends is discovered by the Social Cloud application and passed to a DRIVE Auction Manager to create and run the auction. The Auction Manager uses the list of friends to discover a group of suitable Cloud services based on capacity and relationships; these are termed the bidders in the auction. Each provider uses a DRIVE Agent to act on its behalf to value resource requests, determine a bid based on locally defined policies, and follow the auction protocol. The Auction Manager determines the auction winner and uses the Contract Manager to create an agreement between the auction initiator and the winning bidder. As in the posted price mechanism, the agreement is sent to the specified service for instantiation and the bank for credit transfer.

8.4.6 Risk

Although there is a level of trust between participants in a Social network, this trust may not be sufficient in some situations. Take for example a storage service, where consumers are risking loss, compromise, or corruption of files while providers are risking their own environment by hosting unknown files. Within the Cloud context users may want to take into consideration this lack of control over the corresponding users actions and attempt to minimise risk. In a storage scenario providers can alleviate risk through service design and sandboxing, on the other hand consumers can avoid compromising file content through encryption and reduce the impact of file loss through replication. This raises the possibility of automatically managing such approaches, and

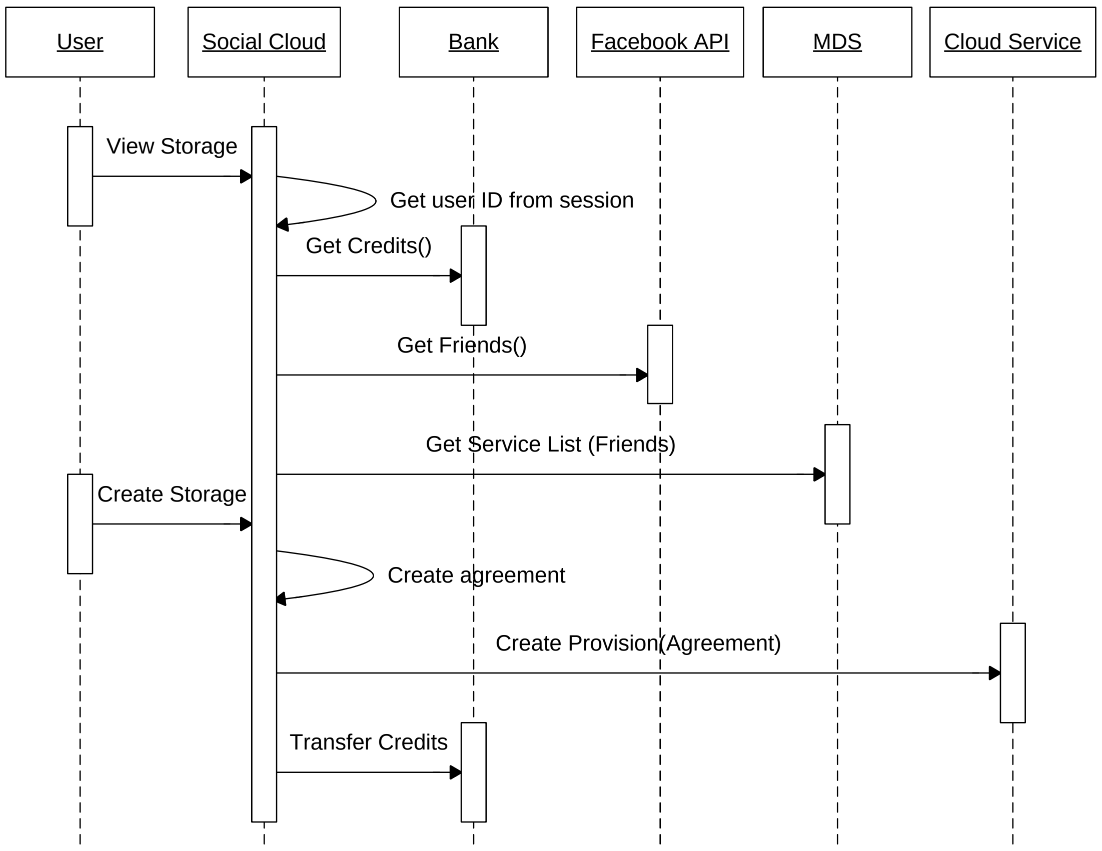


Figure 8.4: Posted Price marketplace in a Social Cloud.

offering premium differentiated services such as replication. For example Storage@home [271] includes a level of redundancy to minimise the risk of loss.

8.5 Implementation

To explore the Social Cloud concept a prototype Social Storage Cloud has been implemented as a Facebook application. In this prototype users are able to access storage provided by their friends using a virtual credit system to trade storage in a DRIVE market. A service based approach has been taken to creating the Social Storage Cloud as the infrastructure must be scalable, distributed and decentralised. All components are implemented as WSRF services and run on Globus WS-core and Tomcat 6. The Facebook application is a Java based web application built from a collection of JSP pages.

Two concurrent economic markets have been implemented to trade storage, both operate in-

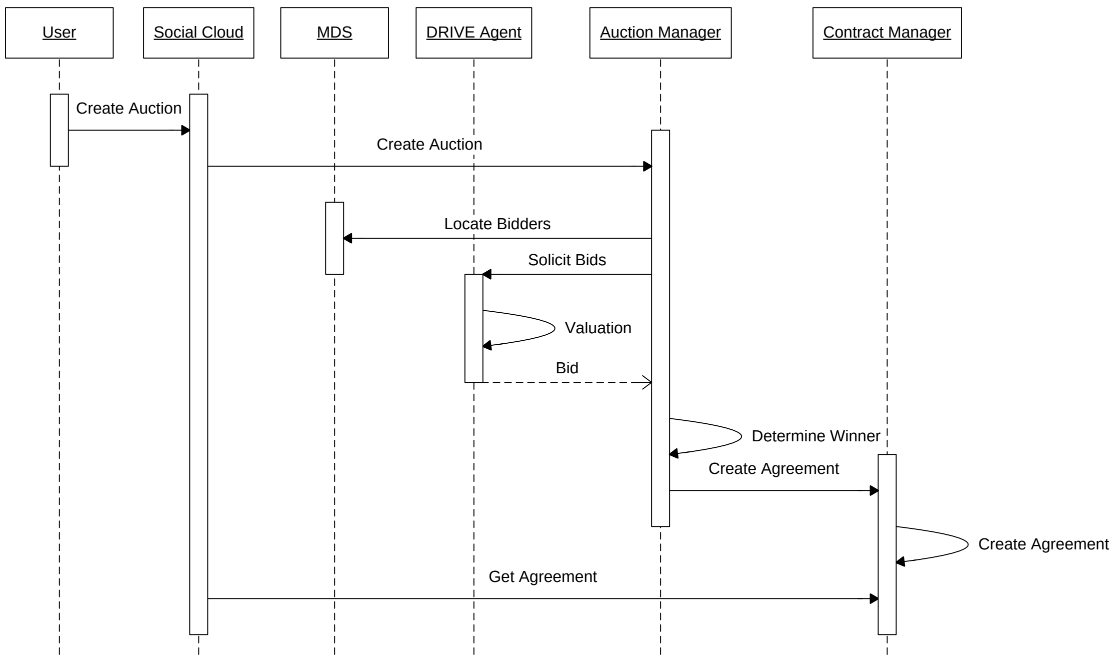


Figure 8.5: Auction marketplace in a Social Cloud. This diagram excludes the actions taken to find the users' ID, retrieve the users' friends, instantiate the Cloud service, and transfer Credits these actions are shown in Fig 8.4.

dependently and are designed to work simultaneously. In a posted price market users select storage from a list of friends' services. The service list includes information regarding pricing, maximum storage, available storage, and the user hosting the service. Fine grained details such as availability and reservation duration are then defined by the consumer and a contract is created between the two users. In the auction market, consumers outline specific storage requirements and pass this description to the Social Cloud infrastructure which in turn uses DRIVE to host an auction. Providers bid to host the storage. When the auction completes a contract is created between the consumer and the winning provider. Contracts from both markets are redeemed through the appropriate storage service which creates a storage instance. In these markets users are made aware of the friend with which they are trading, this is a design decision to provide accountability between friends. In traditional Cloud environments users are unaware of the location of their provision, the prototype Social Cloud could provide this transparency by removing user information from posted price listings, auction requests, and storage access.

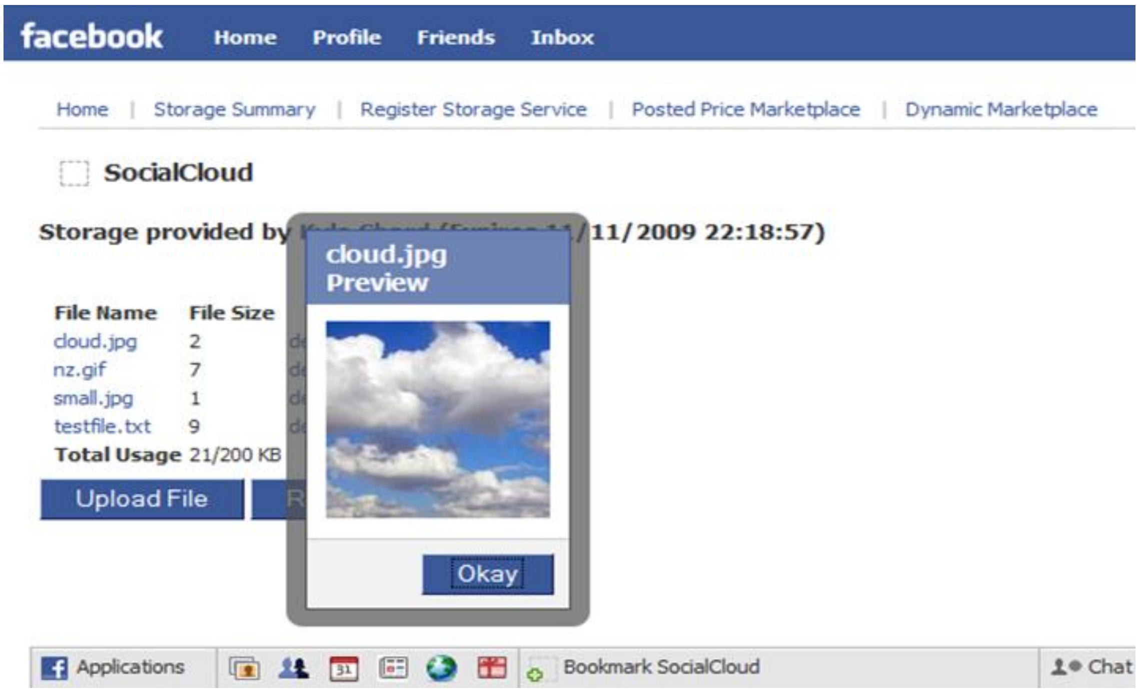
8.5.1 Storage as a Service

This Social Storage Cloud relies on a generic Storage service which provides an interface for users to access virtualised storage. This service exposes a set of file manipulation operations to users and maps their actions to operations on the local file system. gRAVI was used to create a base storage service, the base service has been enhanced by adding additional storage constraints and manipulation operations. Users create storage by passing an agreement to the storage service, this creates a mapping between a user ID, agreement ID, and the storage instance used for subsequent access. Instances are identified by the user ID and agreement ID allowing individual users to have multiple storage instances in the same storage service. The storage service creates a representative resource and an associated working directory for each instance. The resource keeps track of service levels as outlined in the agreement such as the data storage limit. Additionally the service has interfaces to list storage contents, retrieve the amount of storage used/available, upload, download, and delete files. By default the service is configured to use base64 encoded data to transfer files as this can be easily integrated with the Social Cloud Facebook application.

Each storage service relies on a Web application to deliver content to the Facebook application without routing data through the Social Cloud application. To do this the storage service has a collection of JSP pages that perform a specific action and deliver a response in the form of JSON (JavaScript Object Notation) – these actions are Create, Upload, Download, Delete, and List. This approach allows dynamic AJAX invocation of storage operations without requiring a callback or page reload of the Social Cloud Application. Figure 8.6 shows the storage page of the Facebook application, this page is passed a storage service EPR, agreement ID, and user ID in the Get request. A call is made to the specific storage service and a list of files currently stored is displayed. The figure also shows a file preview in a Facebook dialog, previews are supported for image and text based data and are achieved by creating a dynamic instance of the file linked through a preview JSP page hosted by the storage server.

8.5.2 Banking Service

The banking service manages user and agreement information. The service itself is composed of two associated context services each representing different instance data. The first context service records user resources while the second stores hardened agreements (contracts). The user resource stores the user's Facebook ID, current credits, agreement IDs the user has participated in, and auction references (EPR/ID). The agreement resource simply contains any agreements created in the system. These are used to get provision information as well as acting as a receipt. Figure 8.7 shows a summary page generated by querying the banking service, this page displays current and historical agreements with other users. It includes both storage provided and consumed, and information corresponding to each reservation.



Storage provided by /11/2009 22:18:57)

File Name	File Size
cloud.jpg	2
nz.gif	7
small.jpg	1
testfile.txt	9
Total Usage	21/200 KB

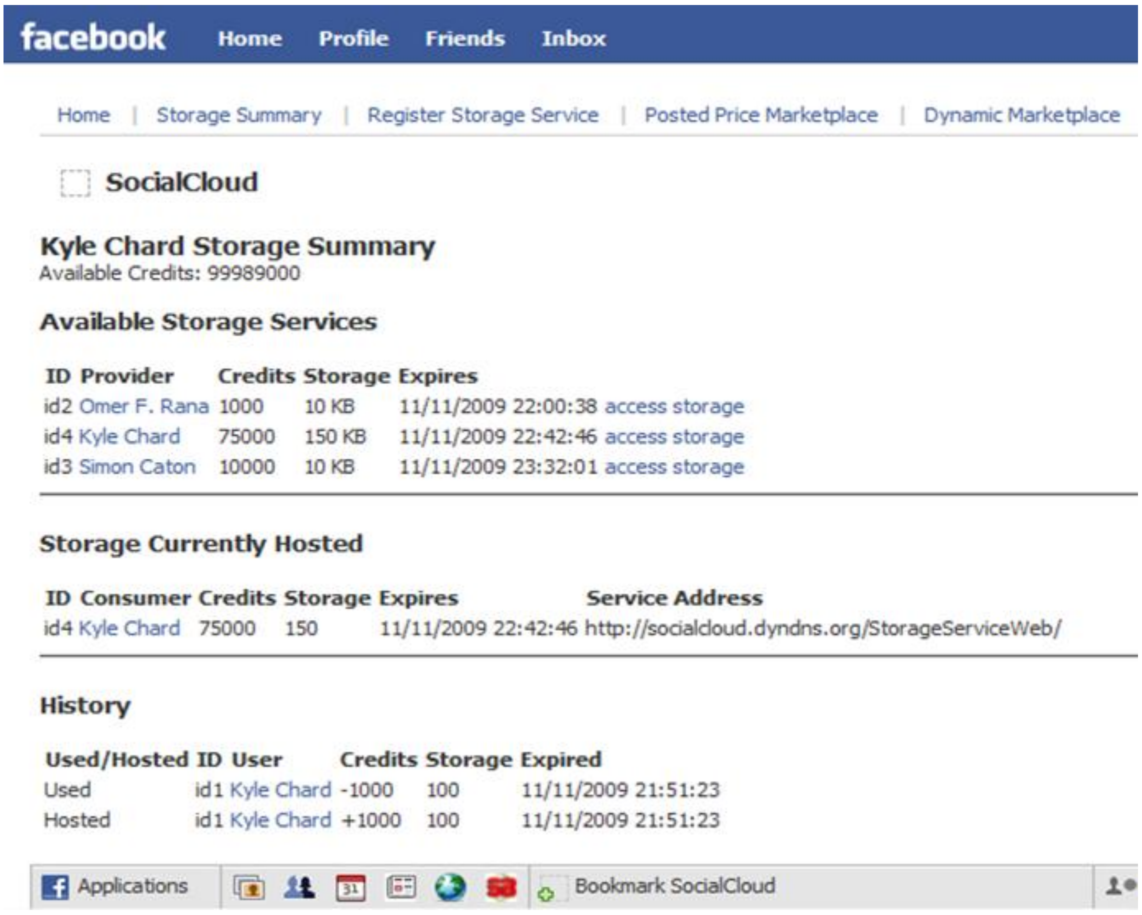
Figure 8.6: Accessing remote storage. Showing a preview of a stored image.

8.5.3 Registration

Facebook manages dynamic user groups in much the same way as VOMS manages VO membership in the DRIVE prototype. User registration in the Social Cloud relies on Facebook credentials and assumes users are existing members of the Social network. All interactions are through the Social Cloud application and therefore all users are pre-authenticated using their Facebook credentials. To access the Social Cloud, users must also be registered with the banking service, this is done transparently when users first access the Social Cloud application. Service registration occurs offline as users must configure their storage service to periodically update metadata describing capacity. The metadata schema used is presented in Listing 8.1. Registration metadata describes the total amount of storage, storage remaining, and pricing information based on a base price per unit of storage. The address of the service and the base JSP URL are also specified so that the Social Cloud application can access the Storage service using AJAX.

Listing 8.1: Social Storage Cloud metadata schema (Resource Profile).

```
<complexType name="SocialCloudMetadata">
  <sequence>
    <element type="string" name="userID"></element>
    <element type="int" name="totalStorage"></element>
    <element type="int" name="remainingStorage"></element>
```


facebook
Home
Profile
Friends
Inbox
Home | Storage Summary | Register Storage Service | Posted Price Marketplace | Dynamic Marketplace
SocialCloud

Kyle Chard Storage Summary
Available Credits: 99989000

Available Storage Services

ID	Provider	Credits	Storage	Expires	
id2	Omer F. Rana	1000	10 KB	11/11/2009 22:00:38	access storage
id4	Kyle Chard	75000	150 KB	11/11/2009 22:42:46	access storage
id3	Simon Caton	10000	10 KB	11/11/2009 23:32:01	access storage

Storage Currently Hosted

ID	Consumer	Credits	Storage	Expires	Service Address
id4	Kyle Chard	75000	150	11/11/2009 22:42:46	http://socialcloud.dyndns.org/StorageServiceWeb/

History

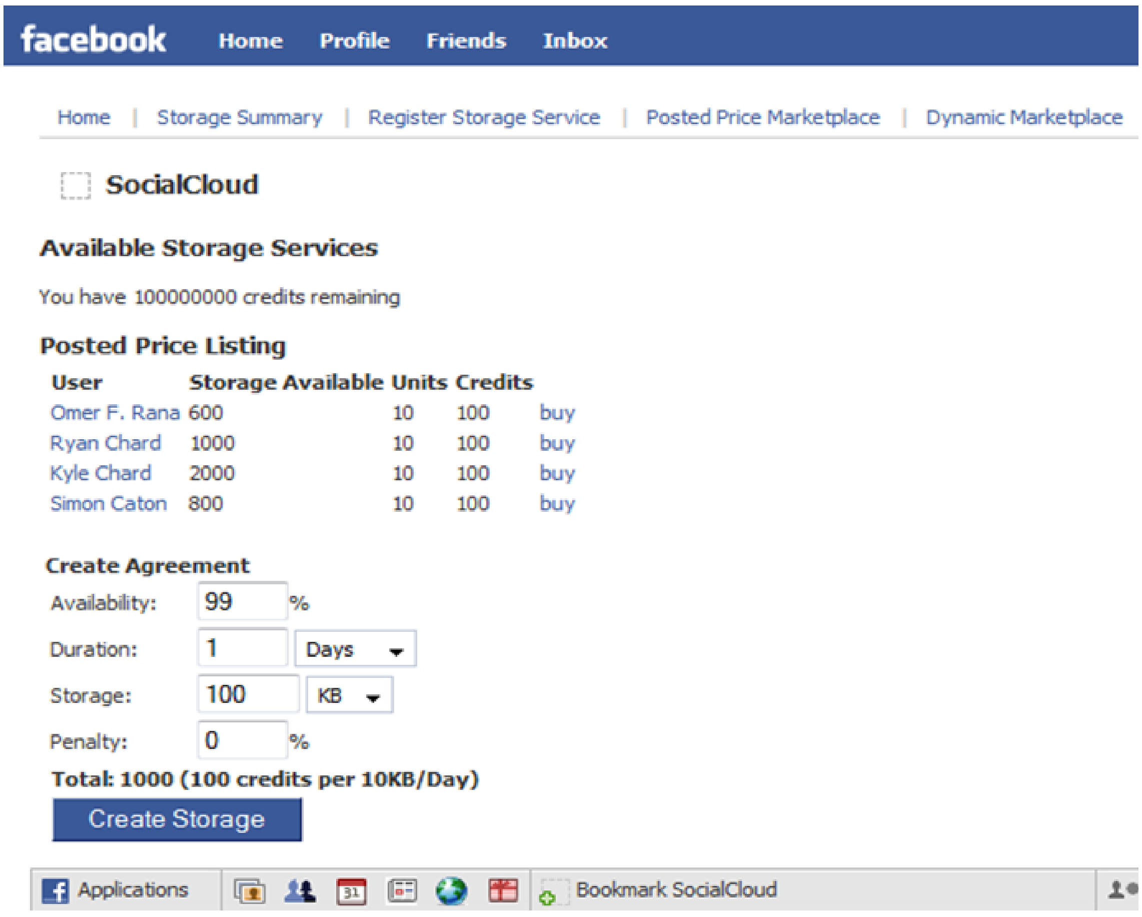
Used/Hosted	ID	User	Credits	Storage	Expired
Used	id1	Kyle Chard	-1000	100	11/11/2009 21:51:23
Hosted	id1	Kyle Chard	+1000	100	11/11/2009 21:51:23

Figure 8.7: Summary of services used and hosted.

```
    <element type="int" name="basePrice"></element>
    <element type="int" name="unitAmount"></element>
    <element type="string" name="Address"></element>
    <element type="string" name="EPRAddress"></element>
    <element type="string" name="JSPAddress"></element>
  </sequence>
</complexType>
```

8.5.4 Posted Price Marketplace

In a posted price marketplace a user can select any advertised service and define specific requirements (storage amount, duration, availability, and penalties) of the provision. Figure 8.8 shows the posted price marketplace page in the Social Cloud application. The service list is obtained by querying MDS for appropriate offers (depending on friends IDs). Registered provider metadata describes available capacity and pricing functions which is offered to users through the marketplace. When the user selects a service and chooses required service levels, a SLA is created using DRIVE. To do this the specific requirements are encoded into an EJSDL [72] (JSDL with economic extensions) document describing the storage request. The EJSDL document is included as the Service Description Term of the agreement and individual requirements are split into guarantee terms, pricing information is mapped to the respective Business Value Lists. An example Social Cloud agreement is presented in Appendix A. For the Social Cloud EJSDL has been extended to include two additional Cloud specific QoS terms: Availability and Error Rate, which are defined as JSDL ranges and are used to describe and monitor the availability of the storage service. Having created an agreement it is passed to the appropriate storage service to create a storage instance. The storage service determines if it will accept the agreement based on local policy and current resource capacity. Having created the storage the agreement is then passed to the banking service to exchange credits and store a copy as a receipt. If either the banking service or storage service decline the agreement both entities will remove the reservation.


facebook
Home
Profile
Friends
Inbox
Home | Storage Summary | Register Storage Service | Posted Price Marketplace | Dynamic Marketplace
SocialCloud

Available Storage Services

You have 100000000 credits remaining

Posted Price Listing

User	Storage Available	Units	Credits	
Omer F. Rana	600	10	100	buy
Ryan Chard	1000	10	100	buy
Kyle Chard	2000	10	100	buy
Simon Caton	800	10	100	buy

Create Agreement

Availability:	99	%
Duration:	1	Days
Storage:	100	KB
Penalty:	0	%

Total: 1000 (100 credits per 10KB/Day)

Create Storage
Applications
Bookmark SocialCloud

Figure 8.8: Posted Price marketplace.

8.5.5 Dynamic Marketplace

The prototype Social Storage Cloud uses DRIVE to provide a complete open market using arbitrary auction protocols. Any of the protocols included in the DRIVE prototype implementation can be used, however, by default the Social Cloud is configured to use a Vickrey protocol due to the existing trust relationships within the Social network. Figure 8.9 shows the dynamic auction marketplace, which lists all currently running auctions. New auctions can be started by specifying required service levels, which are used as the basis for valuation and bid computation. When the auction completes an agreement is created between the user and the winning provider, the EJSDL term language describes the requirements of the provision. The state of the auction/agreement and the final price paid is displayed on the list of current auctions.

The Auction Manager is responsible for creating the auction, soliciting bids, and determining a winner. Individual Bidding Agents act on behalf of the user to compute valuations according to local policy and valuation functions. The prototype DRIVE Bidding Agent has been modified to interact with the Storage Service to determine real time storage capacity (as a fraction of total storage) and compute valuations based on this information, simple linear pricing schemes are used to determine bid prices. The DRIVE Contract Manager is used to create a WS-Agreement based contract between the user and auction winner including the extended EJSDL term language to represent requirements of Cloud provisions.

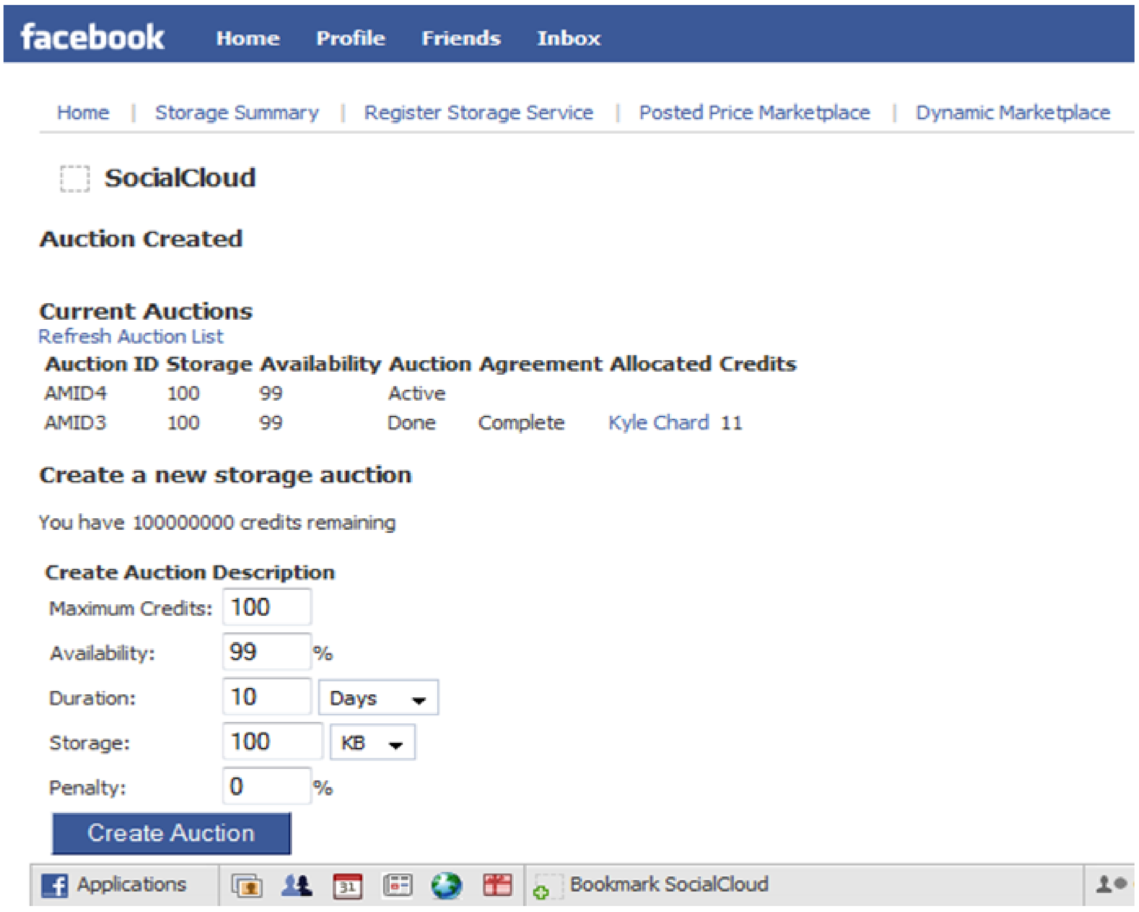
8.6 Evaluation

This section outlines measurements obtained from a deployed Social Storage Cloud focusing on the economic marketplaces used and the Facebook application load time. For the following experiments is is assumed an average Facebook user has 130 friends[7]. DRIVE services are deployed on a single trusted server from subset A (3.0 GHz Core 2 Duo machines, 4GB RAM, Windows Vista). The Facebook application and therefore the page load experiments are hosted on a single machine outside of the experimental testbed (2.2 GHz Dual Core processor, 2 GB memory, Windows Vista).

8.6.1 Posted Price Allocation

Posted price trading requires several steps: identification of storage requirements, generation of a contract, instantiation of a storage service, and registering the transaction with the banking service. The time taken to perform these operations is constant and generally small compared to the time taken to locate applicable storage offers, which is dependent on the registration service and the number of registered entries. The prototype uses MDS to register and discover XML-based metadata describing the real time capabilities of a storage service. Figure 8.10 shows the time taken to query MDS for an increasing number of registered entries. The time includes the cost

[7]`http://www.facebook.com/press/info.php?statistics`


facebook
Home
Profile
Friends
Inbox
Home | Storage Summary | Register Storage Service | Posted Price Marketplace | Dynamic Marketplace
SocialCloud

Auction Created

Current Auctions

Refresh Auction List

Auction ID	Storage	Availability	Auction	Agreement	Allocated	Credits
AMID4	100	99	Active			
AMID3	100	99	Done	Complete	Kyle Chard	11

Create a new storage auction

You have 100000000 credits remaining

Create Auction Description

Maximum Credits:	100	
Availability:	99	%
Duration:	10	Days
Storage:	100	KB
Penalty:	0	%

Figure 8.9: Auction marketplace.

of converting the XML result into a Java Object. Although both the metadata and the topology differ from the DRIVE Evaluation presented in Chapter 6 the results are very similar. With 1GB of memory over 2000 entries can be retrieved in less than 2 seconds. Therefore, MDS can be run even on a low specification server yet still support a small Social Cloud and its posted price market.

In a Social Cloud environment policies dictate the services with which a user is willing to interact (for example friends, friends of friends). Such policies can be implemented by querying for registered services matching particular user IDs (for instance all friends' IDs). Figure 8.11 reflects this situation by loading an increasing number of services in MDS and querying for a subset of registered services (representing friend's services). The query result ranges between 20 and 200 services, while the number of registered services is increased from 200 to 2000. The container is running with 1GB of memory. The time taken to retrieve entries is proportional to the number of registered services and also the number of services returned in the query. Assuming on average 130 friends per user, and the fact not all of these friends would be involved in a Social Cloud, this performance is acceptable – selecting 100 of 2000 registered entries takes approximately 2s.

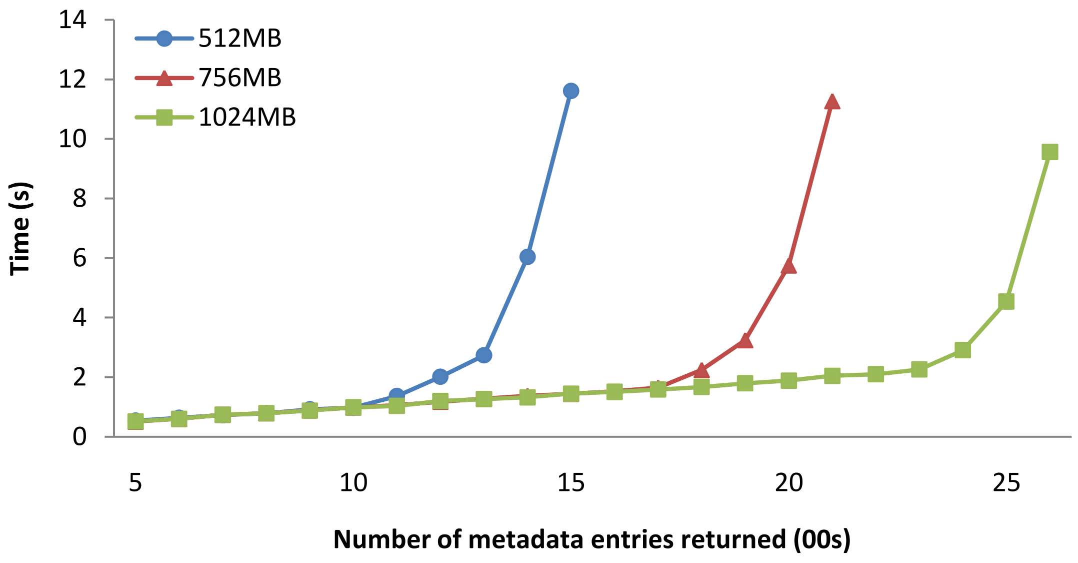


Figure 8.10: Time taken to retrieve service metadata from MDS with different amounts of container memory.

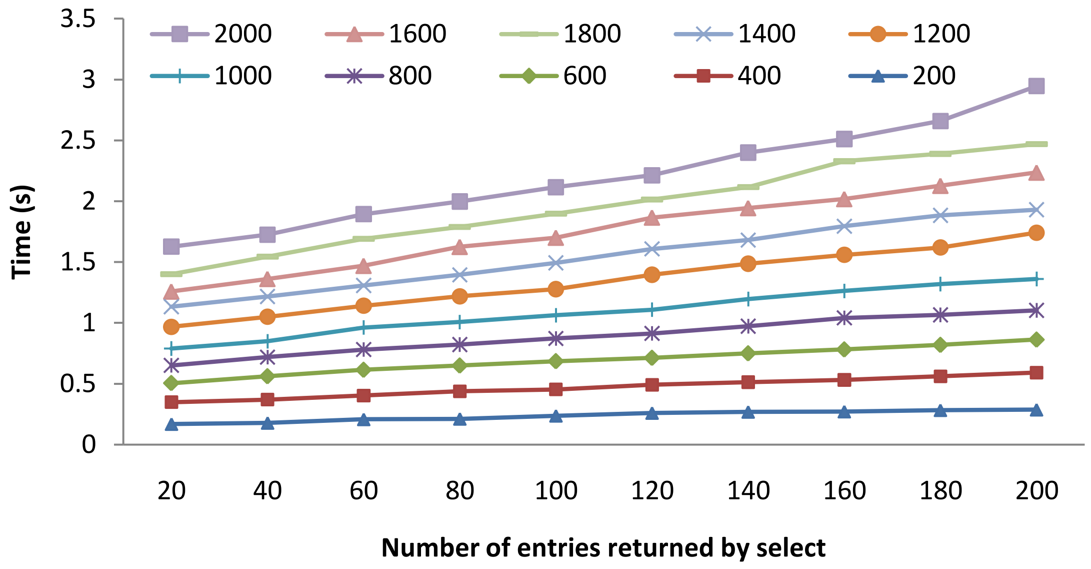


Figure 8.11: Time taken to select a subset of the registered service metadata from MDS with increasing number of total registrations.

8.6.2 Auction Allocation

The auction mechanism relies on a collection of Web services representing the parties involved in the marketplace. The prototype uses a single Auction Manager to conduct the auction and a single Contract Manager to create contracts as a result of the auction. Each storage service is represented by a Bidding Agent which consults local policies to determine a price based on pre-defined metrics. The major point of stress in this system is the Auction Manager as it is responsible for creating an auction, advertising the auction to suitable bidders, soliciting bids, and determining the result of the auction. Contract creation is much simpler as it only involves creation of a WS-Agreement and one call to the winning bidder to verify the agreement (substitute providers are not used in this deployment).

As described in Chapter 6 the performance of the Auction Manager is dependent on the number of concurrent auctions and the number of bids being placed in each auction. Figure 8.12 shows the system throughput with an increasing number of bidders in each auction. The number of auctions per minute is calculated based on the time taken for 500 auctions to complete, this time is measured on the client side. The time starts when the client submits the first task (of 500) through to the creation of the final agreement. Bidders are hosted in a virtualised environment on Subset A (containing 5 machines with 3.0 GHz Core 2 Duo machines, 4GB RAM, Windows Vista). The results match the throughput presented in Chapter 6, the slight difference is due to the different deployment environment and the bidder interactions with the storage service. Figure 8.12 represents the worst case situation when all auctions are started immediately; auctions close as soon as all bidders have bid. In a typical scenario auctions are created with a predefined deadline and users expect some latency between submission and agreement creation. Additionally in a storage context one would expect relatively long term stable reservations which implies users would not conduct auctions frequently. These results show that even with 50 bidders a small scale Auction Manager can complete 65 auctions per minute which, under our assumptions, would be capable of supporting a large scale Social Storage cloud. This number could easily be increased by adding additional Auction Managers to the system, which would be run independently on dedicated hosts.

8.6.3 Application Page Load Time

The Social Cloud Facebook application is made up of several different web pages, each page makes multiple Web service requests to generate page content. To measure the cost of this infrastructure the average page loading times can be compared between the Social Cloud pages and simple control HTML pages without Social Cloud Web service calls. The Facebook application is deployed to Tomcat 6 running on a single server (Windows Vista, 2.2 GHz Dual Core processor, 2 GB memory) located in New Zealand, dynamic DNS is used to map the application URL to the server. Each request is submitted from a separate machine (with the same specifications) also located locally and page load time is measured using Mozilla Firebug.[8]

[8] `http://www.getfirebug.com/`

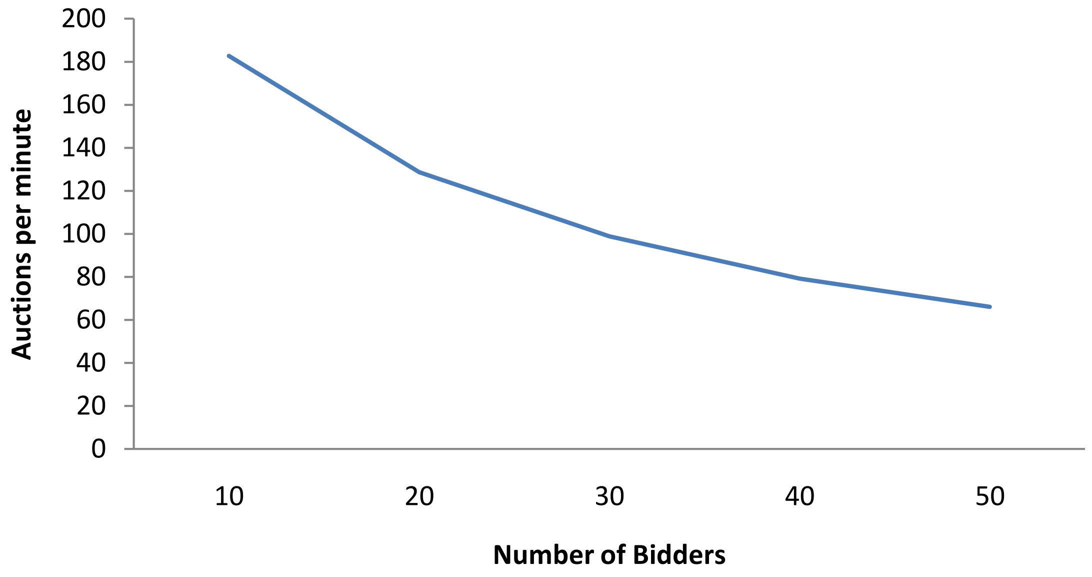


Figure 8.12: Auction throughput. Number of auctions completed per minute for an increasing number of bidders.

Figure 8.13 shows the time taken to load various pages in the application. Each page queries the appropriate Social Cloud services and renders a page displaying the results. Two additional control pages (Control JSP/HTML and simple JSP/FBML) are included to compare load times with the Social Cloud application pages. The bars show the time taken to load the main page content and the additional overhead of other requests for Facebook related content. The main request includes the application page content as well as Facebook content such as the navigation bar, header, and advertising. All results are measured using caching so as to replicate normal user experience

Figure 8.13 shows there is considerable time taken to load a simple page in a Facebook application hosted in this environment. Over 3 seconds are required to load a "hello world" HTML/JSP page containing only a few lines of code that display one line if the user is logged in (a single session variable check). The Simple FBML page is designed to analyse the cost of FBML parsing, in this page a single line of FBML is rendered if the user is logged in. This page is only slightly slower than the plain HTML page due to the additional Facebook parsing. The large delay between request and rendering is due to the cost of routing requests through the Facebook server, and latency between the client, Facebook server, and application server.

The time taken to load the Social Cloud pages is approximately 1 second more than the plain HTML pages due to the extra time taken to make Web Service requests for information. Posted Price (PP) listing and Storage Summary are the least expensive operations as they only make a single Web service call to retrieve a list of registered providers and a list of used storage services respectively. The Posted Price (PP) creation page takes over 4 seconds on average to load, due to the time taken to create an agreement, register it with the banking service, and then create the storage instance. The pages that list and create auctions take the longest time (over 5 seconds) as they involve multiple interactions with Social Cloud services. The auction marketplace pages are substantially slower than the other pages due to the complexity of page generation.

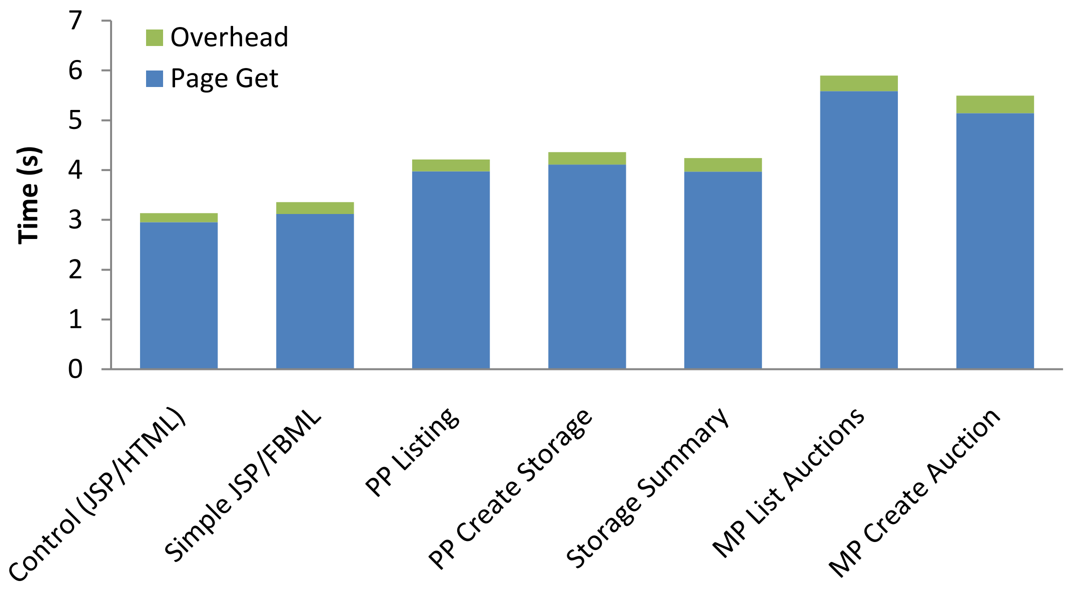


Figure 8.13: Page load time for various Social Cloud pages showing both the main page Get and also the overhead of additional requests.

The significant page load time in the initial prototype greatly effects user experience. However, most of this overhead is due to the routing and parsing process required to generate a Facebook page in this environment. The obvious solution to this problem is to reduce the frequency of page loading and make asynchronous direct calls for content using AJAX. Figure 8.14 shows the loading times for each of the main pages when converted to AJAX calls. The initial page load is also shown as it has increased from the previous results due to the additional time taken to parse and load FBJS. This graph shows that the time taken to perform operations is greatly improved, for example listing the posted price offerings takes approximately 1 second using AJAX whereas previously it took 4 seconds to load a new page without AJAX. Various actions have been included in a single page to amortise the initial loading overhead, however further improvement could be gained by using a single page and AJAX calls to render every action.

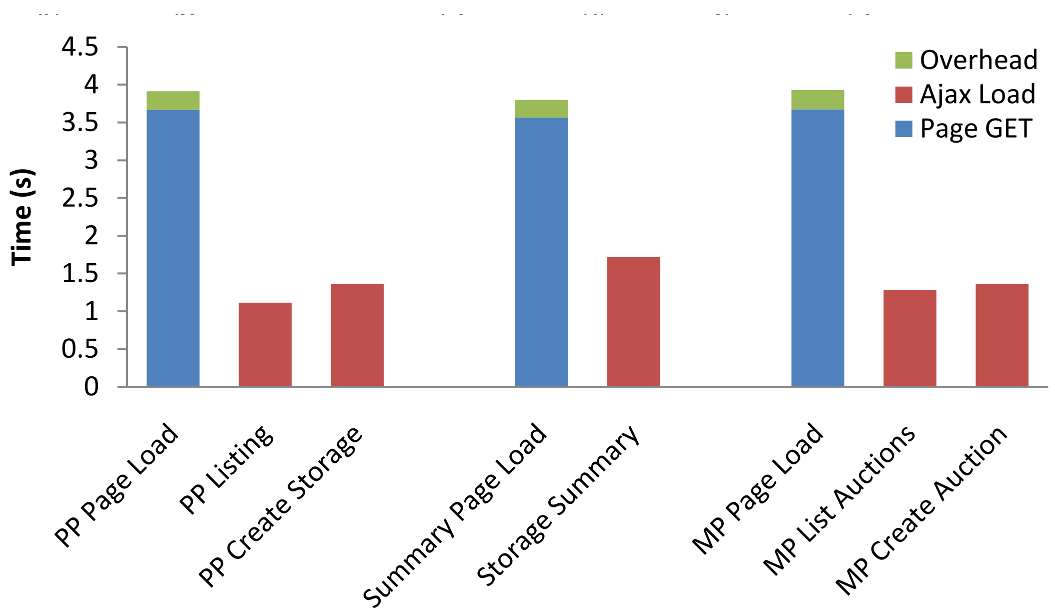


Figure 8.14: Page load time for pages in the Social Cloud Facebook Application showing the initial page load and the time taken to update the page using AJAX.

8.7 Summary

This chapter has defined the concept of a Social Cloud and presented the design and implementation of a functional prototype Social Storage Cloud. This development highlights the versatility of the DRIVE architecture, by using DRIVE as a basis for service trading in a Social Cloud infrastructure. A Social Cloud is an amalgamation of Cloud Computing, Volunteer Computing and Social networking. The prototype Social Storage Cloud allows Facebook users the opportunity to discover and trade storage services contributed by their friends, taking advantage of pre-existing trust relationships. In order to discourage free loading a credit-based approach has been adopted. Here, users can gain credits through services offered, as opposed to purchasing more credits. Users may trade with a specific member of their Social network using a posted price market, or trade storage dynamically through open auction based markets.

The DRIVE infrastructure provides a scalable base from which to construct an open market, providing protocol flexibility and mechanisms to create tailored Cloud specific contracts. The DRIVE-based market in the prototype Social Cloud was quickly and easily implemented using the existing DRIVE prototype. Various alterations and extensions have been made to tailor the DRIVE prototype to a Cloud service model highlighting the versatility and flexibility of the DRIVE architecture. In particular tasks are represented as service requests using an extended EJSDL term language to represent Cloud specific requirements, Bidding Agents have been extended to obtain capacity and pricing information from the storage service, a new resource profile is used to advertise capacity, and Facebook is used to provide dynamic VO management including authentication and authorisation of users.

It was shown empirically that the marketplaces used for trading and/or reciprocation of services could be hosted using small scale resources, based upon the observation that individual social networks are small in size (averaging 130 individuals). In addition, it was shown that even under load, the system can perform multiple concurrent auctions that would satisfy the requirements for a moderately sized Social network. Increasing the scale to millions of users would however require dedicated scalable infrastructure for these components. Analysis of page loading times highlighted the considerable overhead of the hosting environment. However the use of AJAX was shown to greatly improve loading times making the application more usable.

Chapter 9

Conclusions

This thesis has proposed a scalable meta-scheduling architecture for allocating resources in a federated environment. DRIVE is a co-op economic meta-scheduler in which VO members contribute services that collectively perform the tasks required for decentralised federated resource allocation. The contributed services are in effect membership "dues" required to participate in the VO. DRIVE is well suited to federating Grid and Cloud infrastructures as it is not bound to a single task domain, is distributed and scalable, does not require large scale infrastructure investment, is autonomous and self managing, and is also extensible through plug-in interfaces and user defined policies. The DRIVE architecture was designed to meet six clear goals, these goals are restated here from section 4.1 and are referred to directly in Section 9.1.

1. Provider and task independent
2. Support efficient flexible allocation mechanisms
    (a) Efficient allocation mechanisms
    (b) Protocol independent
3. Secure
    (a) General VO security
    (b) Fine grained security
    (c) Secure privacy preserving allocation
4. Scalable
    (a) Scalable architecture
    (b) Autonomous and self managing
    (c) Minimal infrastructure
5. Provide a strong contract management framework

6. Open, interoperable, and standardised

The DRIVE architecture achieves all of these stated goals. This chapter reviews these goals, highlights the contributions made in this thesis, and presents potential research directions.

9.1 Review

DRIVE is designed to create a competitive open market in which VO members can trade access to arbitrary services using a variety of different allocation protocols. Any service type can be offered in a DRIVE market, ranging from raw resources through to high level applications. A service oriented model has been adopted as most Grid and Cloud providers expose resources though service based interfaces. The DRIVE prototype has been deployed and evaluated in both Grid and Cloud domains to demonstrate the versatility of the architecture. This thesis has also defined the concept of Software as a Tradable Service (SaaTS) which refers to trading software in a service oriented market such that providers can charge consumers for service use. To aid this process gRAVI has been extended to automatically create DRIVE-enabled services from binary applications and scripts.

9.1.1 DRIVE

DRIVE is designed to be used in a number of scenarios ranging from small scale local Grids through to large scale federated environments. To achieve this goal DRIVE is independent of provider class (1) and allocation protocol (2b). Provider independence refers to the separation between DRIVE and a particular task model or class of provider. This is vital in a federated environment as different providers with different task models can participate in a single market. To provide this independence, DRIVE places no requirement on the task definition language used which allows users to describe a task in any domain specific representation. The allocation procedure is independent of this description and the resulting contract is formed using the domain specific language as the term language. Although task execution is outside the scope of this thesis, DRIVE is designed to interact with a range of execution environments and architectures. The DRIVE Reservation Service transcends a single domain as it has been designed to represent task independent reservations.

DRIVE relies on economic markets to provide efficient flexible allocation mechanisms (2) and support the allocation principles used by commercial providers (1). Economic protocols provide effective decentralised allocation in distributed systems (2a). However, due to the range of scenarios considered in the development of DRIVE (Chapter 3) a single allocation protocol is not sufficient to meet every users needs. For example secure protocols are necessitated in a global untrusted environment, however within a trusted environment more efficient protocols are more suitable. To facilitate protocol independence (2b) DRIVE includes a plug-in architecture that allows arbitrary protocols to be dynamically loaded and used. In addition DRIVE defines an Auction Component service which can be instantiated as any protocol specific entity to provide the ability to dynamically implement distributed protocols. Communication between plug-in components uses a generic message passing interface to overcome the limitations of typed web service interfaces.

Security is a difficult task in multi-domain (administration and ownership) distributed systems as users and providers are inherently distrustful. DRIVE is based on a VO model designed to represent the dynamic and distributed nature of federated environments. The VO membership service acts to authenticate and authorise users, providers, and DRIVE components (3a). Providers join the VO to trade services and users join in order to negotiate and consume services. As DRIVE has been designed to be deployed in a co-op model security implications are amplified. To overcome these difficulties all DRIVE services are defined according to a deployment trust model which describes their respective requirements. In addition each service supports fine grained security policies and communication between entities can be secured using SSL/TLS (3b). To provide trustworthy allocation in untrusted environments DRIVE is designed to support secure, privacy preserving allocation protocols (3c).

Scalability is a concern for all distributed architectures, this is especially the case in DRIVE due to the potential for large scale global federations. The basis for scalability in the DRIVE architecture is the decentralised distributed mechanisms used for discovery and allocation (4a), and the co-op deployment model (4c). The hint based discovery architecture is designed to reduce the number of allocation participants and also reduce the burden on the Allocation Manager. The decentralised allocation and contract architectures are composed of multiple identical services such that no single service is required to perform any of these actions. In addition, the DRIVE architecture is designed to be deployed in a co-op model therefore distributing the burden of allocation over participating providers and greatly reducing the infrastructural investment required (4c). As systems become large the increase in requirements for human management is often neglected, to limit this issue DRIVE has been designed with an explicit policy framework to reduce the level of management required (4b). For example, DRIVE Agents abstract the complexities of negotiation and allow complex administration through user defined policies.

Like real world economies, DRIVE establishes contracts at the conclusion of an allocation (5). Contracts specify the requirements and obligations of both the provider and the consumer. An incentive model is used to encourage compliance with the terms of the contract. Contracts are represented using WS-agreement as it is standardised, task-independent, and extensible (7). DRIVE uses a proprietary progressive contract negotiation protocol designed to encourage participation and minimise the effect of negotiation latency. Contract negotiation is a two-phased process featuring a tentative agreement reached through economic negotiation which reflects soft-state resource requirements. Prior to redemption this agreement must be converted into a binding contract by a Contract Manager.

Finally the DRIVE architecture has been designed to be completely open and integrate with standards and commonly used Grid services, mechanisms and protocols (7). For example DRIVE leverages MDS for service and provider discovery, and WS-Agreement to represent SLAs. The DRIVE prototype also uses standardised task representation through JSDL, and VO management using VOMS. The entire architecture is modular and can therefore be extended or altered to utilise different components.

9.1.2 DRIVE Prototype

The proof of concept DRIVE prototype includes functional implementations of all the core DRIVE services. The prototype is job based and aims to simulate a federated Grid scenario in which a range of providers are represented in an open market. Three reverse auction protocols (including a secure, trustworthy protocol) have been implemented to validate the plug-in allocation architecture. Each protocol allows users to submit resource requirements in the form of an auction request which is used to create a competitive auction. Providers then bid to satisfy these requirements. At the conclusion of the auction the winner is presented with a contract defining their obligations. JSDL is used to define the requirements of a job and therefore it is also used as the term language in the WS-agreement based contract. Globus GRAM provides the execution interface, however it has been wrapped with a specialised DRIVE Execution Service so that DRIVE protocols can be enforced. The prototype includes functional implementations of three separate policy suites used to verify the XACML policy architecture. VOMS is used to manage dynamic VO membership providing extended Grid credentials to authenticate users. The prototype demonstrates the feasibility of the DRIVE architecture and is the basis for performance evaluation of DRIVE.

To demonstrate the versatility of the DRIVE architecture the initial prototype has been extended to create a dynamic storage market for the prototype Social Cloud. Not only does this prototype serve to highlight the ease with which DRIVE can create open service markets in different environments, it is also a novel contribution in its own right. A Social Cloud is a scalable computing model in which virtualised resources contributed by friends are dynamically provisioned within a Social network. This model has value in a number of domains not least of which is establishing dynamic scientific communities. The DRIVE prototype was shown to be easily adapted to a Cloud domain, using different task representation (EJSDL), VO management (Facebook), and execution environment (storage service). It was also shown empirically that the marketplaces used for trading services could be hosted using small scale resources, based upon the observation that individual social networks are small in size.

9.1.3 Evaluation

The evaluation presented in Chapter 6 not only verifies the operation of the DRIVE meta-scheduler as a whole but also quantifies key performance metrics and examines the benefits of the proposed high utilisation strategies. These high utilisation strategies were implemented to overcome the the perceived limitations of economic resource allocation and in particular auction based allocation. The strategies are a major focus of the evaluation and have been shown to increase occupancy, utilisation, and profit using three synthetic fine grained workloads and a long duration batch model derived from a production Grid trace. DRIVE was shown to provide high throughput economic allocation considering the small scale resources on which the prototype was deployed and the complexity of the protocols used. Using publishing services DRIVE can achieve over 200 auctions (and completed contracts) per minute with 50 concurrent bidders. The co-op model was also shown to be plausible due to the reasonable resource requirements of each of the DRIVE services examined.

9.1.4 gRAVI

To participate in an open service market users must first create economically aware services that adhere to defined economic protocols and offer elastic metered capacity. To simplify this task gRAVI has been extended to address the significant difficulties creating services and interacting with economic markets. Using gRAVI developers can create a fully functional DRIVE-enabled service which can be completely customised using the DRIVE valuation and bidding protocol plug-in interfaces. This approach makes DRIVE accessible to a wide range of users without requiring knowledge of Grid technology, DRIVE infrastructure, or the economic protocols used in the DRIVE market. To our knowledge this is the first example of a wrapping toolkit capable of producing economically enabled services

9.2 Contributions

This thesis has made several research contributions in the fields of federated distributed computing, meta-scheduling, economic resource allocation, web service wrapping toolkits, and Cloud computing. The following contributions are restated here from Chapter 1. Specifically this thesis:

1. Presents the design and prototype implementation of DRIVE – a distributed economic meta-scheduler designed to facilitate large scale resource trading in federated distributed environments. DRIVE is unique for a number of reasons:

   - DRIVE is a co-op architecture in which core components of the meta-scheduler are hosted on (potentially untrusted) participating providers. Trust is established through secure economic protocols.
   - DRIVE is allocation protocol independent to provide the option of simultaneously using both (efficient) insecure and secure protocols in different scenarios. Independence is provided by a generic protocol plug-in interface exposed by DRIVE market services. In addition a generic Allocation Component has been designed that can be instantiated as any protocol dependent entity.
   - DRIVE is domain and provider independent. This separation makes DRIVE a suitable platform from which to create a federated Grid and Cloud environment while also facilitating arbitrary service markets. Task independence is achieved by supporting arbitrary term languages for task submission, allocation, reservations and contracts.

- DRIVE uses a unique two phase SLA negotiation mechanism to encourage negotiation participation and reduce the impact of latency in distributed economies. Soft tentative agreements are created as the result of a negotiation, upon redemption tentative agreements are hardened into binding contracts containing SLAs.
- A hint based reverse discovery mechanism is used to reduce the burden on allocation entities. Providers receive direct advertisements for suitable negotiations, they also have the option of discovering published negotiations through Publishing Services.
- The DRIVE architecture includes a task-independent flexible advanced reservation model therefore providing consumers with flexibility when describing task requirements and interdependencies. This model has been shown to be advantageous to providers due to the ability to schedule tasks.
- DRIVE components are extensible through user defined policies and plug-in interfaces. For example the DRIVE Agent dynamically loads risk policies and valuation functions at runtime to value requests.
- DRIVE is based on a dynamic VO model to represent consumers, providers, and DRIVE services distributed over multiple administrative domains with proprietary security configurations.

2. Analyses the performance of the DRIVE prototype in the context of auction based allocation. To conduct this analysis several synthetic workloads have been developed to model situations where economic allocation typically performs poorly. A variety of experiments are presented focused on bidding policies, auction throughput, DRIVE service overhead, and economic (profit/penalty) analysis. The results verify the implementation and also serve to quantify the salient performance measurements of DRIVE.

3. Identifies and quantifies the performance benefits of four High Utilisation strategies which can be used to overcome some of the perceived limitations of economic mechanisms. The four strategies studied are overbidding and overbooking, second chance substitute providers, flexible advanced reservations, and just-in-time bidding. These strategies can be employed both through allocation protocols and by participants to increase resource occupancy and therefore optimise overall utilisation. They are shown to improve the performance of economic (auction) protocols in the allocation of resources in a high performance computing environment using DRIVE.

4. Presents the design and implementation of gRAVI – an economically enabled Grid and Web service wrapping toolkit. gRAVI is capable of rapidly web enabling legacy applications such that they can be transparently integrated in a DRIVE-based service market. The generated WSRF services are independent from a particular hosting platform and support GSI security, Grid scheduling, state notifications, persistence and data staging. All service code, scripts and definition files are created automatically without any developer input. gRAVI is able

to deploy services directly to the open science cloud Nimbus, while also being able to parse workflow definition files to create strongly typed services. To our knowledge this is the only example of a service wrapping toolkit capable of creating economically enabled services

5. Defines a Social Cloud – a Cloud computing infrastructure utilising virtualised resources contributed by members of a Social network. Additionally the design and implementation of a proof of concept Social Storage Cloud based on DRIVE is presented which highlights the versatility of the DRIVE architecture. The prototype Social Cloud is implemented as a Facebook application which allows "friends" to trade storage with one another in an open DRIVE market. Performance measurements for the deployed posted-price and auction based marketplaces are shown to exceed the requirements of a small Social Cloud. This is the first example of a Cloud computing environment that leverages the trust relationships established in a Social network.

9.3 Future Work

Due to the size and scope of DRIVE there are many potential areas for extension and further research.

9.3.1 Reputation

In real world economies consumers purchase products or services based on many factors, one of the primary factors being brand awareness or reputation. Reputation can be established in a number of ways for example through previous interactions, word of mouth, or even implied through third party relationships. DRIVE too can benefit from such knowledge, for example reputation can be used to estimate the likelihood of an entity meeting their obligations (risk assessment). In addition, in a global utility scenario reputation can be used to augment trust, for example an entity can infer a level of trust in another entity based on its perceived reputation.

From a DRIVE perspective, a reputation framework is required such that entities can record previous experiences (both positive and negative) with other entities (consumers, providers, DRIVE services) such that this information can be used in the future to consider reputation and infer trust. Within a VO it would also be valuable to aggregate reputation information between participants, to establish "brand awareness" amongst entities which have not previously interacted. Secure protocols and mechanisms are required to store, aggregate and share this information in the context of a DRIVE VO.

Such a reputation architecture could transcend DRIVE into other online domains. For example currently individuals are mapped to multiple online identities in a number of different domains (email, social networks, online auctions, banks, Cloud providers), each of which may have an associated reputation measure (for example trader feedback in eBay and trademe). The online reputation of an individual or entity could therefore be aggregated from any of these online identities essentially creating a global reputation measure.

9.3.2 SLA Monitoring and Enforcement

DRIVE includes a complete SLA architecture and implementation using WS-Agreement to represent the requirements and obligations of participants. While the DRIVE architecture defines financial incentives it does not include mechanisms to monitor or enforce agreements. The Contract Manager stores individual guarantee terms as resource properties allowing future application of the WS-Agreement state model to each monitorable term. Using this model each property can be used to reflect the current state of the term (honoured, breached). However a monitoring layer is required to determine if terms have been satisfied.

A simple monitoring layer could be established using a self monitoring philosophy in which participants themselves determine if terms have been met. However, in a commercial environment there is incentive to lie for financial gain, therefore necessitating trusted monitoring techniques. The mechanisms by which provisions are monitored is also dependent on the task domain. For example in a service environment, users may wish to ensure the task is completed by the stated time, however in a Cloud scenario, users may wish to ensure the VM allocated has the advertised capacity. The monitoring layer must therefore be application aware as SLA terms are generally application specific. Perhaps the best approach to this problem is to deploy trusted domain specific monitors within the provider environment (VM, Grid service) this way the monitor can periodically notify if contract terms are honoured. To complete this architecture a best effort Contract Manager is no longer suitable as the task of Contract Monitoring is potentially sensitive and could be subverted. In this case the Contract Manager requires either trusted core deployment, or it could utilise secure protocols (similar to secure verifiable auctions) to ensure results are valid.

Enforcement of SLAs is an even more difficult task due to a lack of control over distributed heterogeneous resources. It is impossible to force compliance as this would require complete control through integration at the resource level, in addition it is unreasonable to expect that providers will not fail (network outage, power failure). However, enforcing the rewards and penalties specified in the agreement is more achievable. This requires integration with a currency provider (bank) or a trusted third party to store a "bond" and then allocate it after contract completion.

Finally, one aspect of SLAs that is not often considered is the legal implications. Typically SLAs are vaguely defined which raises questions about the legal validity. For example the Amazon EC2 terms and conditions state that they do not accept liability for:

"any unanticipated or unscheduled downtime or unavailability of any portion or all of the Services for any reason, including as a result of power outages, system failures or other interruptions" [1]

As yet these terms have not been tested in court to determine their legal validity. In addition as many providers span international borders the legal implications may be even more ambiguous.

[1] `http://www.aws.amazon.com/agreement/`

9.3.3 Protocol Repository

DRIVE is designed to support runtime protocol selection and configuration using a plug-in architecture and predefined DRIVE protocol interfaces. The limitation with the current approach is that all protocol libraries must be distributed and specified to DRIVE services prior to allocation. As the size of a DRIVE VO increases it is difficult to ensure all parties have access to every possible protocol and that the libraries are compatible (versions). A better approach is to use a dynamic protocol repository which allows protocol developers to submit and share protocols through an online mechanism. DRIVE services can then discover and use economic protocols when required. Additionally, the DRIVE prototype currently supports three protocols, to explore the full capabilities of DRIVE a suite of protocols must be developed to reflect the difference scenarios in which DRIVE can be used.

9.3.4 Global Utility Environment

DRIVE provides a step towards creating a global utility environment by facilitating an open market in which any type of consumer or provider can participate. However, provider middleware differs and therefore jobs must be configured for a particular hosting environment. Due to the complexity and cost of porting implementations between hosting environments this will lead to restrictions on participants in the market. To create a truly global computing environment common standards and protocols are required to abstract heterogeneity, this may be accomplished through a middleware layer or common interface. Already there is some "standardised" development taking place for example Eucalyptus and Nimbus have implemented Amazon EC2 interfaces so that any EC2 image can run on any one of these providers. In the Grid world Globus has taken a generic middleware approach using GRAM to abstract provider heterogeneity. Either one of these approaches could be used to overcome heterogeneity in a global computing utility.

9.3.5 High Utilisation Strategies

This thesis proposed and evaluated several high utilisation strategies with respect to a single class of economic allocation. However, strategies such as overbooking, second chance substitution, flexible advanced reservations and Just-In-Time bidding are just as applicable in other domains. For example in any protocol consumer no shows and underestimated resource requirements are likely to be common, overbooking in such a scenario could be used in a similar fashion to increase occupancy at the expense of potential penalties. Second chance substitute providers can be used in any scenario effected by latency which is the case for most distributed allocation protocols. Finally JIT bidding could be generalised by leaving allocation decisions to the last possible moment to ensure accurate capacity prediction. There is potential to explore more general use of these techniques as a means of increasing occupancy, utilisation and allocation performance in different economic and non-economic domains.

9.3.6 Economic Analysis

DRIVE provides a platform on which different economic protocols and functions can be examined. The pricing and penalty functions presented in this thesis aimed to study their effect on allocation and revenue. The pricing functions defined are based on provider data only, these could be extended to incorporate market conditions obtained by analysing previous allocations from Publishing Services. For example strategies such as ZIP and Q-Bidding could be applied. It would also be valuable to compare strategies against one another when deployed concurrenlty in the same market. Likewise penalty functions could be examined in greater detail to analyse their effect on different aspects of the system. In particular pricing functions must become penalty aware in order to consider possible penalties in valuation. These different aspects could potentially be implemented and evaluated in an economic Grid or Cloud simulation toolkit such as GridSim/CloudSim [84]

9.3.7 gRAVI

gRAVI has been successfully used in a number of domains, however there are various improvements that could be made to extend its capabilities. The DRIVE-extensions include most Bidding Agent functionality such as bidding, valuation, contract creation, and advertising. However, there is no support for reverse discovery, reservation management, or generic capacity monitoring. Bidding Agent mechanisms for reverse discovery and reservation support could be integrated such that services are enabled to discover auctions from publishing services and store/schedule reservations using the DRIVE Reservation Service. Currently capacity monitoring is based on system utilisation like the DRIVE prototype, however in many service-based domains this is not a good measure of capacity. A generic capacity monitoring interface is required so users can define metrics used to measure capacity in different domains. A plug-in interface could be used so that developers can customise data gathering and aggregation to determine capacity.

gRAVI currently produces generic invocation interfaces in which arguments and files are transferred as character data, in a number of domains (for example workflows) it is desirable to offer strongly typed interfaces. Introduce includes the notion of data type repositories, it would be beneficial for gRAVI to leverage these repositories in a way that maps defined data types to application input parameters. The Introduce GUI could be used to select and match application parameters to existing service data types. In the area of data staging gRAVI relies heavily on external mechanisms to move medium to large files. It is desirable for generated services to support SOAP attachments as a more scalable way of moving large data sets without the use of external infrastructure. In order to do this, attachment support is required from the Globus Web service container. For Grid scheduling and GridFTP transfer gRAVI relies on community credentials which are not suitable in some scenarios as administrators require accountability of user actions. A better solution would be to support delegation of credentials in a similar manner to that used in GRAM.

9.3.8 Social Cloud

The Social Cloud presents a unique Cloud environment with a number of different areas of potential research. One of the biggest problems with the Social Cloud prototype is the susceptibility to inflation and cheating, new trading policies and mechanisms are required to combat these problems. In addition these problems could be alleviated by integrating the Social Cloud with Facebook credits, this therefore ties the virtual currency used to a real currency. Using Facebook credits providers would be rewarded with real currency for contributing resources to the system. There is also potential to use different types of economies, for example a reciprocation economy [263] which is analogous to car pooling is well suited to sharing resources in a Social Cloud. In such a model peers decide who to donate resources to based on local history, for instance previous interactions with peers.

The Social Cloud architecture presented is generic in that it could represent any type of Cloud service, to verify this assumption the prototype could be generalised so that different service types can be represented in a complex marketplace. This may include a fashioned yellow pages service or ontology in order to capture "similar" utility services. These services may also be provided outside the Social Cloud, for example Amazon S3 storage could be included in the open storage market thus creating a federated Social Cloud architecture. Perhaps the most valuable use for a Social Cloud is in the growing scientific communities. To demonstrate this value a scientific Social Cloud could be created in which collaborators can share research and resources (for example a workflow engine) between friends (in scientific communities).

9.3.9 Additional Directions

In addition to the future work presented in the previous sections there are a number of other avenues for future research around DRIVE. Briefly these potential directions include:

- **Combinatorial JSDL representation:** JSDL has limited support for combinatorial representation through ranges. A full study of JSDL is required to determine if it is capable of representing adequate combinatorial requirements. In addition mappings need to be developed to convert JSDL to common combinatorial representations.
- **High utilisation extensions:** The proposed high utilisation strategies have been shown to be advantageous in an auction scenario. Further research is required to determine appropriate policies to optimise their use in production environments. For example overbidding and overbooking ratios need to be defined based on observed behaviour and second chance substitute limits need to be defined for individual protocols so as not to violate protocol security. Policies regarding selective contract rejection could also be used to maximise provider profit.
- **Function-based Resource Profiles:** The proof of concept resource profiles describing provider capacity define discrete property values. Continuous mathematical functions can be used

to more accurately describe capacity with respect to time. Expressing capacity with functions has been proposed in SLA negotiation [224], in this work SLA terms are represented as multidimensional functions (or systems of functions) rather than constant values or variable ranges. This approach may only be applicable in non-commercial environments as capacity prediction may be commercially sensitive to commercial providers.

- **Reducing trusted core:** Finally a major focus of our future work is reducing the trusted core services so that the infrastructural service requirements are a minimal subset of the overall services. For example secure distributed protocols could potentially be used for contract creation, management, and enforcement. These functions could therefore be executed on untrusted VO members. It might also be possible to similarly distribute an encrypted VOMS architecture

Appendix A

Appendix A: DRIVE Contract

Listing A.1 presents a WS-Agreement based contract generated by DRIVE in the Social Cloud prototype. An extended EJSDL is used as the term language. The extended EJSDL language includes availability and error rate to model Cloud specific requirements.

Listing A.1: DRIVE Agreement

```
<Agreement ws:AgreementId="consumer1-provider1-1256771619664"
 xmlns="http://schemas.ggf.org/graap/2007/03/ws-agreement"
 xmlns:ws="http://schemas.ggf.org/graap/2007/03/ws-agreement"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:bean="http://www.sormaproject.eu/message/ejsdl/beans"
 xmlns:jsdl="http://schemas.ggf.org/jsdl/2005/11/jsdl"
 xmlns:add="http://schemas.xmlsoap.org/ws/2004/03/addressing">

 <Name>Agreement Name</Name>
  <Context>
    <AgreementInitiator xsi:type="bean:Context_Type">
      <bean:PartipantId>consumer1</bean:PartipantId>
    </AgreementInitiator>
    <AgreementResponder xsi:type="bean:Context_Type">
      <bean:PartipantId>provider1</bean:PartipantId>
    </AgreementResponder>
    <ServiceProvider>AgreementResponder</ServiceProvider>
  </Context>
  <Terms>
    <All>
      <ServiceDescriptionTerm ws:ServiceName="jobId" ws:Name="JSDL">
        <bean:ServiceDescription>
          <bean:serviceType>WEBSERVICE</bean:serviceType>
          <jsdl:JobDefinition id="jobId">
            <jsdl:JobDescription>
              <jsdl:JobIdentification>
          <jsdl:JobName>JobName</jsdl:JobName>
          <jsdl:Description>Description..</jsdl:Description>
        </jsdl:JobIdentification>
        <jsdl:Application>
          <jsdl:Description>Appl description..</jsdl:Description>
        </jsdl:Application>
        <jsdl:Resources>
          <jsdl:CandidateHosts>
            <jsdl:HostName>www.drive.com</jsdl:HostName>
          </jsdl:CandidateHosts>
          <jsdl:IndividualDiskSpace>
            <jsdl:UpperBoundedRange>55.0</jsdl:UpperBoundedRange>
            <jsdl:LowerBoundedRange>0.0</jsdl:LowerBoundedRange>
          </jsdl:IndividualDiskSpace>
        </jsdl:Resources>
      </jsdl:JobDescription>
    </jsdl:JobDefinition>
  </bean:ServiceDescription>
</ServiceDescriptionTerm>
<ServiceProperties ws:ServiceName="jobId">
  <VariableSet>
    <Variable ws:Name="CandidateHosts"
              ws:Metric="jsdl:CandidateHosts">
      <Location>/JobDescription/Resources/CandidateHosts</Location>
    </Variable>
    <Variable ws:Name="IndividualDiskSpace"
              ws:Metric="jsdl:IndividualDiskSpace">
      <Location>
          /JobDescription/Resources/IndividualDiskSpace
      </Location>
    </Variable>
    <Variable ws:Name="Reservation" ws:Metric="Reservation">
      <Location>/ReservationDocument/ReservationType</Location>
    </Variable>
    <Variable ws:Name="Availability" ws:Metric="bean:Availability">
      <Location>/ServiceQoSTerms/Availability</Location>
    </Variable>
    <Variable ws:Name="ErrorRate" ws:Metric="bean:ErrorRate">
      <Location>/ServiceQoSTerms/ErrorRate</Location>
    </Variable>
  </VariableSet>
</ServiceProperties>
<GuaranteeTerm ws:Name="CandidateHosts" ws:Obligated="ServiceProvider">
  <ServiceScope ws:ServiceName="jobId"/>
  <ServiceLevelObjective>
    <KPITarget>
      <KPIName>jsdl:CandidateHosts</KPIName>
      <CustomServiceLevel xsi:type="jsdl:CandidateHosts_Type">
        <jsdl:HostName>www.hostname.com</jsdl:HostName>
      </CustomServiceLevel>
    </KPITarget>
  </ServiceLevelObjective>
  <BusinessValueList>
    <Importance>1</Importance>
    <Penalty>
      <AssessmentInterval>
        <TimeInterval>PT30S</TimeInterval>
      </AssessmentInterval>
      <ValueExpression xsi:type="bean:Penalty_Type">
        <bean:functionName>DefaultPenalty</bean:functionName>
        <bean:normalizationConstant>
          10.0
        </bean:normalizationConstant>
      </ValueExpression>
    </Penalty>
    <Reward>
      <ValueExpression xsi:type="bean:MarketMessage_Type">
        <bean:clearingPrice>20.0</bean:clearingPrice>
        <bean:currency xsi:nil="true"/>
        <bean:paymentType>EITHER</bean:paymentType>
      </ValueExpression>
    </Reward>
  </BusinessValueList>
</GuaranteeTerm>
<GuaranteeTerm ws:Name="IndividualDiskSpace"
               ws:Obligated="ServiceProvider">
  <ServiceScope ws:ServiceName="jobId"/>
  <ServiceLevelObjective>
    <KPITarget>
      <KPIName>jsdl:IndividualDiskSpace</KPIName>
      <CustomServiceLevel xsi:type="jsdl:RangeValue_Type">
        <jsdl:UpperBoundedRange>55.0</jsdl:UpperBoundedRange>
        <jsdl:LowerBoundedRange>0.0</jsdl:LowerBoundedRange>
      </CustomServiceLevel>
    </KPITarget>
  </ServiceLevelObjective>
  <BusinessValueList>
    <Importance>1</Importance>
    <Penalty>
      <AssessmentInterval>
        <TimeInterval>PT30S</TimeInterval>
      </AssessmentInterval>
      <ValueExpression xsi:type="bean:Penalty_Type">
        <bean:functionName>DefaultPenalty</bean:functionName>
        <bean:normalizationConstant>
          10.0
        </bean:normalizationConstant>
      </ValueExpression>
    </Penalty>
    <Reward>
```

```
      <ValueExpression xsi:type="bean:MarketMessage_Type">
        <bean:clearingPrice>20.0</bean:clearingPrice>
        <bean:currency xsi:nil="true"/>
        <bean:paymentType>EITHER</bean:paymentType>
      </ValueExpression>
    </Reward>
  </BusinessValueList>
</GuaranteeTerm>
<GuaranteeTerm ws:Name="Reservation"
               ws:Obligated="ServiceProvider">
  <ServiceScope ws:ServiceName="jobId"/>
  <ServiceLevelObjective>
    <KPITarget>
      <KPIName>Reservation</KPIName>
      <CustomServiceLevel xsi:type="res:reservation_type"
            xmlns:res="http://ws.sormaproject.eu/eerm/reservation">
        <res:request>
          <res:interval>
            <res:startFrom>2008-11-10T12:00:00-05:00</res:startFrom>
            <res:finishUntil>
                  2008-11-10T12:00:00-05:00
            </res:finishUntil>
            <res:duration amount="10.0" units="hours"/>
          </res:interval>
        </res:request>
      </CustomServiceLevel>
    </KPITarget>
  </ServiceLevelObjective>
  <BusinessValueList>
    <Importance>1</Importance>
    <Penalty>
      <AssessmentInterval>
        <TimeInterval>PT30S</TimeInterval>
      </AssessmentInterval>
      <ValueExpression xsi:type="bean:Penalty_Type">
        <bean:functionName>DefaultPenalty</bean:functionName>
        <bean:normalizationConstant>
          10.0
          </bean:normalizationConstant>
      </ValueExpression>
    </Penalty>
    <Reward>
      <ValueExpression xsi:type="bean:MarketMessage_Type">
        <bean:clearingPrice>20.0</bean:clearingPrice>
        <bean:currency xsi:nil="true"/>
        <bean:paymentType>EITHER</bean:paymentType>
      </ValueExpression>
    </Reward>
  </BusinessValueList>
</GuaranteeTerm>
```

```
<GuaranteeTerm ws:Name="Availability" ws:Obligated="ServiceProvider">
  <ServiceScope ws:ServiceName="jobId"/>
  <ServiceLevelObjective>
    <KPITarget>
      <KPIName>bean:Availability</KPIName>
      <CustomServiceLevel xsi:type="jsdl:RangeValue_Type">
        <jsdl:LowerBoundedRange>90.0</jsdl:LowerBoundedRange>
      </CustomServiceLevel>
    </KPITarget>
  </ServiceLevelObjective>
  <BusinessValueList>
    <Importance>1</Importance>
    <Penalty>
      <AssessmentInterval>
        <TimeInterval>PT30S</TimeInterval>
      </AssessmentInterval>
      <ValueExpression xsi:type="bean:Penalty_Type">
        <bean:functionName>DefaultPenalty</bean:functionName>
        <bean:normalizationConstant>
          10.0
        </bean:normalizationConstant>
      </ValueExpression>
    </Penalty>
    <Reward>
      <ValueExpression xsi:type="bean:MarketMessage_Type">
        <bean:clearingPrice>20.0</bean:clearingPrice>
        <bean:currency xsi:nil="true"/>
        <bean:paymentType>EITHER</bean:paymentType>
      </ValueExpression>
    </Reward>
  </BusinessValueList>
</GuaranteeTerm>
<GuaranteeTerm ws:Name="ErrorRate" ws:Obligated="ServiceProvider">
  <ServiceScope ws:ServiceName="jobId"/>
  <ServiceLevelObjective>
    <KPITarget>
      <KPIName>bean:ErrorRate</KPIName>
      <CustomServiceLevel xsi:type="jsdl:RangeValue_Type">
        <jsdl:LowerBoundedRange>90.0</jsdl:LowerBoundedRange>
      </CustomServiceLevel>
    </KPITarget>
  </ServiceLevelObjective>
  <BusinessValueList>
    <Importance>1</Importance>
    <Penalty>
      <AssessmentInterval>
        <TimeInterval>PT30S</TimeInterval>
      </AssessmentInterval>
      <ValueExpression xsi:type="bean:Penalty_Type">
        <bean:functionName>DefaultPenalty</bean:functionName>
```

```
              <bean:normalizationConstant>
                10.0
              </bean:normalizationConstant>
            </ValueExpression>
          </Penalty>
          <Reward>
            <ValueExpression xsi:type="bean:MarketMessage_Type">
              <bean:clearingPrice>20.0</bean:clearingPrice>
              <bean:currency xsi:nil="true"/>
              <bean:paymentType>EITHER</bean:paymentType>
            </ValueExpression>
          </Reward>
        </BusinessValueList>
      </GuaranteeTerm>
      <ServiceReference
            ws:Name="EERMEndpoint" ws:ServiceName="JobExecutionALL">
        <add:EndpointReference>
          <add:Address>
                http://somehost:8081/SocialCloud/StorageService
          </add:Address>
        </add:EndpointReference>
      </ServiceReference>
    </All>
  </Terms>
</Agreement>
```

Appendix B

Appendix B: Bidding Policy

Listing B.1 shows an example DRIVE policy used to determine risk based on a reputation value. The policy compares the given reputation value with a predefined value (10) to determine if the reputation is high risk.

Listing B.1: Reputation XACML Policy

```
<Policy PolicyId="ReputationHighRiskPolicy"
RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:
rule-combining-algorithm:ordered-permit-overrides">
  <Description>
    This policy determines if the given reputation value is perceived
    to be high risk (i.e not trustworthy)
  </Description>
  <Target>
    <Subjects>
      <Subject>
        <SubjectMatch
         MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
          <AttributeValue
           DataType="http://www.w3.org/2001/XMLSchema#string">
               Reputation
          </AttributeValue>
          <SubjectAttributeDesignator
           AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
           DataType="http://www.w3.org/2001/XMLSchema#string"/>
          </SubjectMatch>
        </Subject>
      </Subjects>
      <Resources>
      <Resource>
        <ResourceMatch
         MatchId="urn:oasis:names:tc:xacml:1.0:function:anyURI-equal">
```

```
          <AttributeValue
           DataType="http://www.w3.org/2001/XMLSchema#anyURI">
               ReputationHighRisk
          </AttributeValue>
          <ResourceAttributeDesignator
           AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"
           DataType="http://www.w3.org/2001/XMLSchema#anyURI"/>
        </ResourceMatch>
      </Resource>
    </Resources>
    <Actions>
      <AnyAction/>
    </Actions>
  </Target>
  <Rule RuleId="ReputationHighRiskRule" Effect="Permit">
    <Target>
      <Subjects>
        <AnySubject/>
      </Subjects>
      <Resources>
        <AnyResource/>
      </Resources>
      <Actions>
        <Action>
          <ActionMatch
           MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
            <AttributeValue
             DataType="http://www.w3.org/2001/XMLSchema#string">
              ReputationAction
            </AttributeValue>
            <ActionAttributeDesignator
             AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
             DataType="http://www.w3.org/2001/XMLSchema#string"/>
          </ActionMatch>
        </Action>
      </Actions>
    </Target>
    <Condition FunctionId="greater-than">
      <ResourceAttributeDesignator
       AttributeId="reputation:value"
       DataType="http://www.w3.org/2001/XMLSchema#integer"
       MustBePresent="true"/>
      <AttributeValue
       DataType="http://www.w3.org/2001/XMLSchema#integer">
        10
      </AttributeValue>
    </Condition>
  </Rule>
  <Rule RuleId="DefaultRule" Effect="Deny"/>
</Policy>
```

Appendix C

Appendix C: DRIVE Schemas

This appendix presents schema definitions and code segments relating to the DRIVE prototype implementation. The listings are referred to directly in Chapter 5.

Listing C.1: DRIVE Task (Job) Description Schema.

```
<complexType name="AuctionRequest">
  <sequence>
    <element type="string" name="AuctionID"></element>
    <element type="string" name="AuctionProtocol"></element>
    <element type="string" name="AuctionManager"></element>
    <element type="string" name="ContractManager"></element>
    <element ref="PublishingServices"></element>
    <element ref="JobWindow"></element>
    <element ref="CoallocationNumber"></element>
    <element ref="AuctionOptions"></element>
    <element ref="AuctionProtocolProperties"></element>
    <element ref="AuctionComponents"></element>
    <element ref="AuctionDescription"></element>
    <element ref="EconomicProperties"></element>
    <element ref="AnyType"></element>
  </sequence>
</complexType>

<complexType name="AuctionOptions">
  <sequence>
    <element type="dateTime" name="AuctionEndTime"></element>
    <element type="int" name="MaxNumberBidders"></element>
    <element type="string" name="AuctionOption"
                           maxOccurs="unbounded" minOccurs="0">
    </element>
  </sequence>
</complexType>
```

Listing C.2: DRIVE Auction State.

```
<simpleType name="AuctionState">
  <restriction base="string">
    <enumeration value="NotStarted" />
    <enumeration value="Pending" />
    <enumeration value="Advertising" />
    <enumeration value="RequestingBids" />
    <enumeration value="Active"  />
    <enumeration value="Finalising"  />
    <enumeration value="Done" />
    <enumeration value="Failed" />
  </restriction>
</simpleType>
```

Listing C.3: Reservation Window Schema.

```
<complexType name="ReservationWindow">
   <sequence>
      <element type="dateTime" name="StartTime"></element>
      <element type="dateTime" name="EndTime"></element>
      <element type="duration" name="Duration"></element>
   </sequence>
</complexType>
```

Listing C.4: Reservation Schema.

```
<complexType name="Reservation">
   <sequence>
      <element type="string" name="ReservationID"></element>
      <element ref="ReservationType"></element>
      <element ref="ReservationWindow"></element>
      <element ref="Contract"></element>
      <element ref="TaskDescription"></element>
      <element type="string" name="ResourceID"></element>
      <element type="string" name="ResourceAddress"></element>
      <element type="string" name="AuctionManager"></element>
      <element type="string" name="ContractManager"></element>
      <element ref="tns:AnyType"></element>
   </sequence>
</complexType>
```

Listing C.5: VOMS proxy info (Note: The CA path has been reduced for brevity).

```
#voms-proxy-info -all
```

```
subject  : /O=Grid/OU=DRIVE/OU=simpleCA/OU=dsrg.mcs.vuw.ac.nz/CN=Kyle
issuer   : /O=Grid/OU=DRIVE/OU=simpleCA/Globus Simple CA
identity : /O=Grid/OU=DRIVE/OU=simpleCA/Globus Simple CA
type     : unknown
strength : 512 bits
path     : /tmp/x509up_u509
timeleft : 11:59:55
=== VO DRIVE extension information ===
VO : DRIVE
subject : /O=Grid/OU=DRIVE/OU=simpleCA/OU=dsrg.mcs.vuw.ac.nz/CN=Kyle
issuer :  /O=Grid/OU=DRIVE/OU=simpleCA/Globus Simple CA
attribute : /DRIVEVO/Role=NULL/Capability=NULL
timeleft : 11:59:55
```

Listing C.6: DRIVE provider metadata schema (Resource Profile).

```
<complexType name="DRIVEMetadata">
  <sequence>
    <element ref="Categories"></element>
    <element ref="AuctionProtocols"></element>
    <element type="int" name="Computation"></element>
    <element type="int" name="Memory"></element>
    <element type="int" name="IO"></element>
    <element type="string" name="Address"></element>
  </sequence>
</complexType>

<complexType name="Categories">
  <sequence>
    <element ref="Category"
             maxOccurs="unbounded" minOccurs="0"></element>
  </sequence>
</complexType>

<simpleType name="Category">
  <restriction base="string">
    <enumeration value="Computation"/>
    <enumeration value="Memory"/>
    <enumeration value="IO"/>
  </restriction>
</simpleType>
```

Listing C.7: DRIVE Provider XPath Query. This query will retrieve all DRIVE metadata objects with computation greater than 10 and categorised as memory intensive and supporting the Garbled Circuits (GC) auction protocol.

```
//*[local-name()='DRIVEMetadata'][Computation>10]
 /*[local-name()='Categories'][.='memory']
 /*[local-name()='AuctionProtocols'][AuctionProtocol='GC']/..
```

Bibliography

[1] M. Dias de Assunção, R. Buyya, and S. Venugopal, "Intergrid: a case for internetworking islands of grids," *Concurrency and Computation: Practice & Experience*, vol. 20, no. 8, pp. 997–1024, 2008.

[2] K. Keahey, I. Foster, T. Freeman, and X. Zhang, "Virtual workspaces: Achieving quality of service and quality of life in the grid," *Scientific Programming*, vol. 13, no. 4, pp. 265–275, 2005.

[3] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov, "The eucalyptus open-source cloud-computing system," in *Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID '09)*, (Washington, DC, USA), pp. 124–131, IEEE Computer Society, 2009.

[4] J. O. Callaghan, "A national grid infrastructure for australian researchers," *Cyberinfrastructure Technology Watch Quarterly*, vol. 2, no. 1, pp. 1–6, 2006. http://www.ctwatch.org/quarterly/articles/2006/02/a-national-grid-infrastructure-for-australian-researchers/ [Accessed May 2010].

[5] E. Laure and B. Jones, "Enabling grids for e-science: The egee project," in *Grid Computing: Infrastructure, Service, and Applications* (L. Wang, W. Jie, and J. Chen, eds.), CRC, 2009.

[6] C. Catlett, "The philosophy of teragrid: Building an open, extensible, distributed terascale facility," in *Proceedings of the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID '02)*, (Washington, DC, USA), p. 8, IEEE Computer Society, 2002.

[7] C. Catlett, P. Beckman, D. Skow, and I. Foster, "Creating and operating national-scale cyberinfrastructure services," *Cyberinfrastructure Technology Watch Quarterly*, vol. 2, no. 2, pp. 2–10, 2006.

[8] M. Livny, R. Pordes, K. Blackburn, P. Avery, and I. Foster, "Open science grid annual report 2007-2008," tech. rep., Open Science Grid Consortium, 2008. http://osg-docdb.opensciencegrid.org/cgi-bin/RetrieveFile?docid=770&extension=pdf [Accessed May 2010].

[9] R. Pordes, D. Petravick, B. Kramer, D. Olson, M. Livny, A. Roy, P. Avery, K. Blackburn, T. Wenaus, F. Würthwein, I. Foster, R. Gardner, M. Wilde, A. Blatecky, J. McGee, and R. Quick, "The open science grid status and architecture," *Journal of Physics: Conference Series*, vol. 119, no. 5, p. 052028, 2007.

[10] R. Bolze, F. Cappello, E. Caron, M. Daydé, F. Desprez, E. Jeannot, Y. Jégou, S. Lanteri, J. Leduc, N. Melab, G. Mornet, R. Namyst, P. Primet, B. Quetier, O. Richard, E.-G. Talbi, and I. Touche, "Grid'5000: A large scale and highly reconfigurable experimental grid testbed," *International Journal of High Performance Computing Applications*, vol. 20, no. 4, pp. 481–494, 2006.

[11] K. Chard and K. Bubendorfer, "Using secure auctions to build a distributed meta-scheduler for the grid," in *Market Oriented Grid and Utility Computing* (R. Buyya and K. Bubendorfer, eds.), Wiley Series on Parallel and Distributed Computing, pp. 569–588, New York, USA: Wiley Press, 2009.

[12] K. Chard and K. Bubendorfer, "A distributed economic meta-scheduler for the grid," in *Proceedings of the 2008 8th IEEE International Symposium on Cluster Computing and the Grid (CCGRID '08)*, (Washington, DC, USA), pp. 542–547, IEEE Computer Society, 2008.

[13] I. Foster, C. Kesselman, and S. Tuecke, "The anatomy of the grid: Enabling scalable virtual organizations," *International Journal of Supercomputer Applications*, vol. 15, no. 3, pp. 200–222, 2001.

[14] C. Smith, "Open source metascheduling for virtual organizations with the community scheduler framework," Technical Whitepaper May 14, Platform Computing, 2003. http://www.cs.virginia.edu/ grimshaw/CS851-2004/Platform/CSF_architecture.pdf [Accessed May 2010].

[15] X. Wei, Z. Ding, S. Yuan, C. Hou, and H. Li, "Csf4: A wsrf compliant meta-scheduler," in *Proceedings of International Conference on Grid Computing & Applications (GCA '06)*, (Las Vegas, USA), pp. 61–67, 2006.

[16] Z. Ding, X. Wei, Y. Luo, D. ma, P. W. Arzberger, and W. W. Li, "Customized plug-in modules in metascheduler csf4 for life sciences applications," *New Generation Computing*, vol. 25, no. 4, pp. 373–394, 2007.

[17] R. Ranjan, A. Harwood, and R. Buyya, "A case for cooperative and incentive based coupling of distributed clusters," *Future Generation Computing Systems*, vol. 24, no. 4, pp. 280–295, 2008.

[18] R. Ranjan, *Coordinated Resource Provisioning in Federated Grids*. Phd thesis, Department of Computer Science and Software Engineering, The University of Melbourne, Australia, July 2007.

[19] E. Huedo, R. S. Montero, and I. M. Llorente, "A framework for adaptive execution in grids," *SoftwarePractice & Experience*, vol. 34, no. 7, pp. 631–651, 2004.

[20] E. Huedo, R. S. Montero, and I. M. Llorente, "The gridway framework for adaptive scheduling and execution on grids," *Scalable Computing - Practice and Experience*, vol. 6, no. 3, pp. 1–8, 2005.

[21] E. Huedo, R. S. Montero, and I. M. Llorente, "A modular meta-scheduling architecture for interfacing with pre-ws and ws grid resource management services," *Future Generation Computer Systems*, vol. 23, no. 2, pp. 252–261, 2007.

[22] K. Czajkowski, D. F. Ferguson, I. Foster, J. Frey, S. Graham, I. Sedukhin, D. Snelling, S. Tuecke, and W. Vambenepe, "The ws-resource framework," tech. rep., Globus, 2004. http://www.globus.org/wsrf/specs/ws-wsrf.pdf [Accessed May 2010].

[23] I. E. Sutherland, "A futures market in computer time," *Communications of the ACM*, vol. 11, no. 6, pp. 449–451, 1968.

[24] C. A. Waldspurger, T. Hogg, B. A. Huberman, J. O. Kephart, and W. S. Stornetta, "Spawn: A distributed computational economy," *IEEE Transactions on Software Engineering*, vol. 18, no. 2, pp. 103–117, 1992.

[25] R. Buyya, D. Abramson, J. Giddy, and H. Stockinger, "Economic models for resource management and scheduling in grid computing," *Concurrency and Computation: Practice and Experience (CCPE)*, vol. 14, no. 13-15, pp. 1507–1542, 2002.

[26] C.-H. Chien, P. H.-M. Chang, and V.-W. Soo, "Market-oriented multiple resource scheduling in grid computing environments," in *Proceedings of the 19th International Conference on Advanced Information Networking and Applications (AINA '05)*, (Washington, DC, USA), pp. 867–872, IEEE Computer Society, 2005.

[27] M. P. Wellman, "A market-oriented programming environment and its application to distributed multicommodity flow problems," *Journal of Artificial Intelligence Research*, vol. 1, no. 1, pp. 1–23, 1993.

[28] K. Bubendorfer, *NOMAD: Towards An Architecture for Mobility in Large Scale Distributed Systems*. Phd thesis, School of Mathematics, Statistics, and Computer Science, Victoria University of Wellington, 2001.

[29] S. Krawczyk and K. Bubendorfer, "Grid resource allocation: allocation mechanisms and utilisation patterns," in *Proceedings of the sixth Australasian workshop on Grid computing and e-research (AusGrid '08)*, (Darlinghurst, Australia, Australia), pp. 73–81, Australian Computer Society, Inc., 2008.

[30] K. Bubendorfer, B. Palmer, and I. Welch, "Trust and privacy in grid resource auctions," in *Encyclopedia of Grid Computing Technologies and Applications* (E. Udoh and F. Wang, eds.), IGI Global, 2008.

[31] I. Raicu, Y. Zhao, C. Dumitrescu, I. Foster, and M. Wilde, "Falkon: a fast and light-weight task execution framework," in *Proceedings of the 2007 ACM/IEEE conference on Supercomputing (SC '07)*, (New York, NY, USA), pp. 1–12, ACM, 2007.

[32] M. Litzkow, M. Livny, and M. Mutka, "Condor - a hunter of idle workstations," in *Proceedings of the 8th International Conference of Distributed Computing Systems (ICDCS '98)*, pp. 104–111, IEEE Computer Society, 1988.

[33] I. Foster and C. Kesselman, "Globus: A metacomputing infrastructure toolkit," *The International Journal of Supercomputer Applications and High Performance Computing*, vol. 11, no. 2, pp. 115–128, 1997. Globus: A Metacomputing Infrastructure Toolkit.

[34] D. P. Anderson, J. Cobb, E. Korpela, M. Lebofsky, and D. Werthimer, "SETI@home: an experiment in public-resource computing," *Communications of the ACM*, vol. 45, no. 11, pp. 56–61, 2002.

[35] D. P. Anderson, "BOINC: A system for public-resource computing and storage," in *Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing (GRID '04)*, (Washington, DC, USA), pp. 4–10, IEEE Computer Society, 2004.

[36] B. Nitzberg, J. M. Schopf, and J. P. Jones, "Pbs pro: Grid computing and scheduling attributes," in *Grid resource management: state of the art and future trends* (J. Nabrzyski, J. M. Schopf, and J. Weglarz, eds.), pp. 183–190, Kluwer Academic Publishers, 2004.

[37] M. Q. Xu, "Effective metacomputing using lsf multicluster," in *Proceedings of the 1st International Symposium on Cluster Computing and the Grid (CCGRID '01)*, (Washington, DC, USA), p. 100, IEEE Computer Society, 2001.

[38] S. Zhou, "LSF: load sharing in large-scale heterogeneous distributed systems," in *Proceedings of the 1st Workshop on Cluster Computing*, (Tallahassee, FL), 1992.

[39] H. Shan, L. Oliker, and R. Biswas, "Job superscheduler architecture and performance in computational grid environments," in *Proceedings of the 2003 ACM/IEEE conference on Supercomputing (SC '03)*, (Washington, DC, USA), p. 44, IEEE Computer Society, 2003.

[40] D. Abramson, J. Giddy, and L. Kotler, "High performance parametric modeling with nimrod/g: Killer application for the global grid?," in *Proceedings of the 14th International Symposium on Parallel and Distributed Processing (IPDPS '00)*, (Washington, DC, USA), p. 520, IEEE Computer Society, 2000.

[41] R. Buyya, D. Abramson, and J. Giddy, "Nimrod/g: an architecture for a resource management and scheduling system in a global computational grid," in *Proceedings of the 4th International Conference on High Performance Computing in Asia-Pacific Region (HPC Asia 2000)*, (Beijing, China), pp. 283–289, IEEE Computer Society Press, 2000.

[42] S. Venugopal, R. Buyya, and L. Winton, "A grid service broker for scheduling e-science applications on global data grids: Research articles," *Concurrency and Computation: Practice & Experience*, vol. 18, no. 6, pp. 685–699, 2006.

[43] I. Foster and C. Kesselman, *The Grid: Blueprint for a New Computing Infrastructure*. San Francisco, CA, USA: Morgan Kaufmann Publishers, 1998.

[44] I. Foster, "What is the grid? - a three point checklist." GRIDToday - www.mcs.anl.gov/ itf/Articles/WhatIsTheGrid.pdf [Accessed May 2010], July 20 2002.

[45] W. Gentzsch, "Sun grid engine: Towards creating a compute power grid," in *Proceedings of the 1st International Symposium on Cluster Computing and the Grid (CCGRID '01)*, (Washington, DC, USA), p. 35, IEEE Computer Society, 2001.

[46] K. Czajkowski, I. T. Foster, N. T. Karonis, C. Kesselman, S. Martin, W. Smith, and S. Tuecke, "A resource management architecture for metacomputing systems," in *Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing (IPPS/SPDP '98)*, vol. 1459 of *Lecture Notes In Computer Science*, (London, UK), pp. 62–82, Springer-Verlag, 1998.

[47] M. Feller, I. Foster, and S. Martin, "GT4 GRAM: a functionality and performance study," in *TeraGrid Conference*, (Madison, WI), 2007.

[48] S. Fitzgerald, I. Foster, C. Kesselman, G. von Laszewski, W. Smith, and S. Tuecke, "A directory service for configuring high-performance distributed computations," in *Proceedings of the 6th IEEE International Symposium on High Performance Distributed Computing (HPDC '97)*, (Washington, DC, USA), p. 365, IEEE Computer Society, 1997.

[49] K. Czajkowski, S. Fitzgerald, I. Foster, and C. Kesselman, "Grid information services for distributed resource sharing," in *Proceedings of the 10th IEEE Symposium on High Performance Distributed Computing (HPDC '01)*, (San Francisco, CA, USA), pp. 181–194, 2001.

[50] L. M. Vaquero, L. Rodero-Merino, J. Caceres, and M. Lindner, "A break in the clouds: towards a cloud definition," *ACM SIGCOMM Computer Communication Review*, vol. 39, no. 1, pp. 50–55, 2009.

[51] A. Kertész and P. Kacsuk, "A taxonomy of grid resource brokers," in *Distributed and Parallel Systems*, pp. 201–210, Springer, 2007.

[52] C. S. Yeo and R. Buyya, "A taxonomy of market-based resource management systems for utility-driven cluster computing," *SoftwarePractice & Experience*, vol. 36, no. 13, pp. 1381–1419, 2006.

[53] D. Neumann, J. Stößer, A. Anandasivam, and N. Borissov, "Sorma - building an open grid market for grid resource allocation," in *Proceedings of the 4th International Workshop on Grid Economics and Business Models (GECON '07)*, vol. 4685 of *Lecture Notes in Computer Science:*, (Rennes, France), pp. 194–200, 2007.

[54] D. Abramson, R. Buyya, and J. Giddy, "A computational economy for grid computing and its implementation in the nimrod-g resource broker," *Future Generation Computer Systems*, vol. 18, no. 8, pp. 1061–1074, 2002.

[55] K. Lai, L. Rasmusson, E. Adar, L. Zhang, and B. A. Huberman, "Tycoon: An implementation of a distributed, market-based resource allocation system," *Multiagent and Grid Systems*, vol. 1, no. 3, pp. 169–182, 2005.

[56] A. AuYoung, B. Chun, A. Snoeren, and A. Vahdat, "Resource allocation in federated distributed computing infrastructures," in *Proceedings of the 1st Workshop on Operating System and Architectural Support for the on demand IT InfraStructure (OASIS 2004)*, 2004.

[57] Y. Fu, J. Chase, B. Chun, S. Schwab, and A. Vahdat, "Sharp: an architecture for secure resource peering," *SIGOPS Operating Systems Review*, vol. 37, no. 5, pp. 133–148, 2003.

[58] L. Adzigogov, J. Soldatos, and L. Polymenakos, "EMPEROR: An ogsa grid meta-scheduler based on dynamic resource predictions," *Journal of Grid Computing*, vol. 3, no. 1-2, pp. 19–37, 2005.

[59] G. Moltó, V. Hernández, and J. M. Alonso, "A service-oriented wsrf-based architecture for metascheduling on computational grids," *Future Generation Computer Systems*, vol. 24, no. 4, pp. 317–328, 2008.

[60] C. Daval-Frerot, M. Lacroix, and H. Guyennet, "Resource balancing using trader federation," in *Proceedings of the 5th IEEE Symposium on Computers and Communications (ISCC '00)*, (Washington, DC, USA), p. 647, IEEE Computer Society, 2000.

[61] C. Daval-Frerot, M. Lacroix, and H. Guyennet, "Federation of resource traders in object-oriented distributed systems," in *Proceedings of the International Conference on Parallel Computing in Electrical Engineering (PARELEC '00)*, pp. 84–88, 2000.

[62] D. Ouelhadj, J. Garibaldi, J. MacLaren, R. Sakellariou, and K. Krishnakumar, "A multi-agent infrastructure and a service level agreement negotiation protocol for robust scheduling in grid computing," in *Advances in Grid Computing (EGC '05)*, vol. 3470 of *Lecture Notes in Computer Science*, pp. 651–660, Springer-Verlag, 2005.

[63] N. Andrade, W. Cirne, and F. B. P. Roisenberg, "Ourgrid: An approach to easily assemble grids with equitable resource sharing," in *Proceedings of the 9th Workshop on Job Scheduling Strategies for Parallel Processing*, vol. 2862 of *Lecture Notes in Computer Science*, pp. 68–86, Springer, 2003.

[64] W. Cirne, F. Brasileiro, N. Andrade, L. B. Costa, A. Andrade, R. Novaes, and M. Mowbray, "Labs of the world, unite!!!," *Journal of Grid Computing*, vol. 4, no. 3, pp. 225–246, 2006.

[65] L. Peterson, T. Anderson, D. Culler, and T. Roscoe, "A blueprint for introducing disruptive technology into the internet," *ACM SIGCOMM Computer Communication Review*, vol. 33, no. 1, pp. 59–64, 2003.

[66] R. Byrom, L. Cornwall, A. Djaoui, L. Field, S. Fisher, S. Hicks, M. Soni, and A. Wilson, "R-GMA: an information integration system for grid monitoring," in *Proceedings of the 11th International Conference on Cooperative Information Systems (CoopIS '03)*, 2003.

[67] J. Yu, S. Venugopal, and R. Buyya, "A market-oriented grid directory service for publication and discovery of grid service providers and their services," *The Journal of Supercomputing*, vol. 36, no. 1, pp. 17–31, 2006.

[68] M. L. Massie, B. N. Chun, and D. E. Culler, "The ganglia distributed monitoring system: design, implementation, and experience," *Parallel Computing*, vol. 30, no. 7, pp. 817–849, 2004.

[69] R. Ranjan, A. Harwood, and R. Buyya, "Peer-to-peer based resource discovery in global grids: A tutorial," *IEEE Communications Surveys and Tutorials*, vol. 10, no. 2, pp. 6–33, 2008. Peer-to-Peer Based Resource Discovery In Global Grids: A Tutorial.

[70] D. Oppenheimer, J. Albrecht, D. Patterson, and A. Vahdat, "Design and implementation tradeoffs for wide-area resource discovery," in *Proceedings of the 14th IEEE International Symposium on High Performance Distributed Computing (HPDC '05)*, (Washington, DC, USA), pp. 113–124, IEEE Computer Society, 2005.

[71] A. Anjomshoaa, F. Brisard, M. Drescher, D. Fellows, A. Ly, S. McGough, D. Pulsipher, and A. Savva, "Job submission description language (JSDL) specification, version 1.0," Grid Forum Document GFD.56, Open Grid Forum, 2005. http://www.gridforum.org/documents/GFD.56.pdf.

[72] N. Borissov, S. Caton, O. Rana, and A. Levine, "Message protocols for provisioning and usage of computing services," in *Proceedings of the 6th International Workshop on Grid Economics and Business Models (GECON 09)*, vol. 5745 of *Lecture Notes in Computer Science*, pp. 160–170, Springer, 2009.

[73] "The globus resource specification language RSL v1.0," specification, Globus Alliance.

[74] H. Rajic, R. Brobst, W. Chan, F. Ferstl, J. Gardiner, J. P. Robarts, A. Haas, B. Nitzberg, D. Templeton, J. Tollefsrud, and P. Tröger, "Distributed resource management application api specification 1.0," Specification GFD-R-P.22, DRMAA Working Group, The Global Grid Forum, 2003.

[75] G. von Laszewski, J. Gawor, S. Krishnan, and K. Jackson, "Commodity grid kits - middleware for building grid computing environments," in *Grid Computing: Making the Global Infrastructure a Reality* (F. Berman, G. Fox, and T. Hey, eds.), Wiley Series in Communications Networking & Distributed Systems, Wiley, 2003.

[76] T. Goodale, S. Jha, H. Kaiser, T. Kielmann, P. Kleijer, A. Merzky, J. Shalf, and C. Smith, "A simple api for grid applications (saga)," Grid Forum Document GFD.90, Open Grid Forum (OGF), 2008. http://www.ogf.org/documents/GFD.90.pdf [Accessed May 2010].

[77] H. Kaiser, A. Merzky, S. Hirmer, and G. Allen, "The saga c++ reference implementation - lessons learnt from juggling with seemingly contradictory goal," in *Proceedings of the 2nd International Workshop on Library-Centric Software Design (LCSD '06) at Object-Oriented Programming, Systems, Languages and Applications conference (OOPSLA '06)*, pp. 101–106, 2006.

[78] A. Andrieux *et al.*, "Web services agreement specification (WS-Agreement)," Grid Forum Document GFD-R-P.107, Open Grid Forum, 2007.

[79] I. Foster, C. Kesselman, C. Lee, R. Lindell, K. Nahrstedt, and A. Roy, "A distributed resource management architecture that supports advance reservations and co-allocation," in *Proceeings of the 7th International Workshop on Quality of Service (IWQoS '99)*, pp. 27–36, 1999.

[80] A. Roy and V. Sander, "Advance reservation api," Draft GFD-E.5, Scheduling Working Group, Global Grid Forum (GGF), 2002.

[81] T. Meinl, "Advance reservation of grid resources via real options," in *Proceedings of the 2008 10th IEEE Conference on E-Commerce Technology and the Fifth IEEE Conference on Enterprise Computing, E-Commerce and E-Services (CECANDEEE '08)*, (Washington, DC, USA), pp. 3–10, IEEE Computer Society, 2008.

[82] R. Buyya and S. Venugopal, "The gridbus toolkit for service oriented grid and utility computing: An overview and status report," in *Proceedings of the 1st IEEE International Workshop on Grid Economics and Business Models (GECON '04)*, 2004.

[83] S. Venugopal, X. Chu, and R. Buyya, "A negotiation mechanism for advance resource reservation using the alternate offers protocol," in *Proceedings of the 16th International Workshop on Quality of Service (IWQoS 2008)*, (University of Twente, Enschede,The Netherlands), pp. 40–49, 2008.

[84] R. Buyya and M. Murshed, "Gridsim: A toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing," *Concurrency and Computation: Practice and Experience*, vol. 14, no. 13–15, pp. 1175–1220, 2002.

[85] A. Sulistio and R. Buyya, "A grid simulation infrastructure supporting advance reservation," in *Proceedings of the 16th International Conference on Parallel and Distributed Computing and Systems (PDCS '04)*, (MIT Cambridge, Boston, USA), pp. 1–7, 2004.

[86] L. Pearlman, C. Kesselman, V. Welch, I. Foster, and S. Tuecke, "The community authorization service: Status and future," in *Proceedings of the Conference for Computing in High Energy and Nuclear Physics (CHEP '03)*, 2003.

[87] M. Lorch, D. B. Adams, D. Kafura, M. S. R. Koneni, A. Rathi, and S. Shah, "The prima system for privilege management, authorization and enforcement in grid environments," in *Proceedings of the 4th International Workshop on Grid Computing (GRID '03)*, (Washington, DC, USA), p. 109, IEEE Computer Society, 2003.

[88] R. Alfieri, R. Cecchini, V. Ciaschini, L. dell'Agnello, A. Frohner, K. Lőrentey, and F. Spataro, "From gridmap-file to voms: managing authorization in a grid environment," *Future Generation Computer Systems*, vol. 21, no. 4, pp. 549 – 558, 2005.

[89] R. Alfieri, R. Cecchini, V. Ciaschini, L. dellAgnello, A. Gianoli, F. Spataro, F. Bonnassieux, P. Broadfoot, G. Lowe, L. Cornwall, J. Jensen, D. Kelsey, A. Frohner, D. Groep, W. S. de Cerff, M. Steenbakkers, G. Venekamp, D. Kouril, A. McNab, O. Mulmo, M. Silander, J. Hahkala, and K. Lőrentey, "Managing dynamic user communities in a grid of autonomous resources," in *Proceedings of Computing and High Energy Physics (CHEP 2003)*, 2003.

[90] S. Langella, S. Oster, S. Hastings, F. Siebenlist, T. Kurc, and J. Saltz, "Dorian: Grid service infrastructure for identity management and federation," in *Proceedings of the 19th IEEE Symposium on Computer-Based Medical Systems (CBMS '06)*, (Washington, DC, USA), pp. 756–761, IEEE Computer Society, 2006.

[91] E. Damiani, D. C. di Vimercati, S. Paraboschi, P. Samarati, and F. Violante, "A reputation-based approach for choosing reliable resources in peer-to-peer networks," in *Proceedings of the 9th ACM conference on Computer and communications security (CCS '02)*, (New York, NY, USA), pp. 207–216, ACM, 2002.

[92] M. Yokoo and K. Suzuki, "Secure multi-agent dynamic programming based on homomorphic encryption and its application to combinatorial auctions," in *Proceedings of the first international joint conference on Autonomous agents and multiagent systems (AAMAS '02)*, (New York, NY, USA), pp. 112–119, ACM, 2002.

[93] K. Suzuki and M. Yokoo, "Secure generalized vickery auction using homomorphic encryption," in *Proceedings of the 7th International Conference on Financial Cryptography (FC '03)*, no. 2742 in Lecture Notes in Computer Science, pp. 239–249, 2003.

[94] A. Luther, R. Buyya, R. Ranjan, and S. Venugopal, "Peer-to-peer grid computing and a .net-based alchemi framework," in *High Performance Computing: Paradigm and Infrastructure* (L. Yang and M. Guo, eds.), Wiley Press, 2004.

[95] A. I. T. Rowstron and P. Druschel, "Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems," in *Proceedings of the IFIP/ACM International Confer-*

ence on Distributed Systems Platforms Heidelberg, Middleware '01, (London, UK), pp. 329–350, Springer-Verlag, 2001.

[96] P. Wieder, J. Seidel, O. Wäldrich, W. Ziegler, and R. Yahyapour, "Using sla for resource management and scheduling - a survey," in *Grid Middleware and Services: Challenges and Solutions* (D. Talia, R. Yahyapour, and W. Ziegler, eds.), pp. 335–347, Springer, 2008.

[97] D. Abramson, R. Sosic, J. Giddy, and B. Hall, "Nimrod: a tool for performing parametrised simulations using distributed workstations," in *Proceedings of the 4th IEEE International Symposium on High Performance Distributed Computing (HPDC '95)*, (Washington, DC, USA), p. 112, IEEE Computer Society, 1995.

[98] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the art of virtualization," *SIGOPS Operating Systems Review*, vol. 37, pp. 164–177, 2003.

[99] N. Nisan, "Bidding and allocation in combinatorial auctions," in *Proceedings of the 2nd ACM conference on Electronic commerce (EC '00)*, (New York, NY, USA), pp. 1–12, ACM, 2000.

[100] B. N. Chun, C. Ng, J. Albrecht, D. C. Parkes, and A. Vahdat, "Computational resource exchanges for distributed resource allocation." http://www.eecs.harvard.edu/ chaki/-doc/share04.pdf [Accessed May 2010], 2004.

[101] D. C. Parkes, J. Kalagnanam, and M. Eso, "Achieving budget-balance with vickrey-based payment schemes in exchanges," in *Proceedings of the 17th international joint conference on Artificial intelligence (IJCAI'01)*, (San Francisco, CA, USA), pp. 1161–1168, Morgan Kaufmann Publishers Inc., 2001.

[102] P. A. Dinda, "Online prediction of the running time of tasks," *Cluster Computing*, vol. 5, no. 3, pp. 225–236, 2002.

[103] J. M. Alonso, V. Hernández, and G. Moltó, "Gmarte: Grid middleware to abstract remote task execution: Research articles," *Concurrency and Computation: Practice & Experience*, vol. 18, no. 15, pp. 2021–2036, 2006.

[104] R. Smith, "The contract net protocol: high level communication and control in distributed problem solver," *IEEE Transactions on Computers*, vol. 29, pp. 1104–113, 1980.

[105] S. Basu, S. Banerjee, P. Sharma, and S.-J. Lee, "Nodewiz: peer-to-peer resource discovery for grids," in *Proceedings of the 5th IEEE International Symposium on Cluster Computing and the Grid (CCGRID '05)*, (Washington, DC, USA), pp. 213–220, IEEE Computer Society, 2005.

[106] G. Haggard, D. Pearce, and G. Royle, "Computing tutte polynomials," *ACM Transactions on Mathematical Software (To appear)*, 2010. Victoria University of Wellington.

[107] E. Deelman, S. Callaghan, E. Field, H. Francoeur, R. Graves, N. Gupta, V. Gupta, T. H. Jordan, C. Kesselman, P. Maechling, J. Mehringer, G. Mehta, D. Okaya, K. Vahi, and L. Zhao, "Managing large-scale workflow execution from resource provisioning to provenance tracking: The cybershake example," in *Proceedings of the Second IEEE International Conference on e-Science and Grid Computing (E-SCIENCE '06)*, (Washington, DC, USA), p. 14, IEEE Computer Society, 2006.

[108] M. R. Palankar, A. Iamnitchi, M. Ripeanu, and S. Garfinkel, "Amazon s3 for science grids: a viable solution?," in *Proceedings of the 2008 international workshop on Data-aware distributed computing (DADC '08)*, (New York, NY, USA), pp. 55–64, ACM, 2008.

[109] D. Kondo, B. Javadi, P. Malecot, F. Cappello, and D. P. Anderson, "Cost-benefit analysis of cloud computing versus desktop grids," in *Proceedings of the 2009 IEEE International Symposium on Parallel&Distributed Processing (IPDPS '09)*, (Washington, DC, USA), pp. 1–12, IEEE Computer Society, 2009.

[110] M. Dias de Assunção, A. di Costanzo, and R. Buyya, "Evaluating the cost-benefit of using cloud computing to extend the capacity of clusters," in *Proceedings of the 18th ACM international symposium on High performance distributed computing (HPDC '09)*, (New York, NY, USA), pp. 141–150, ACM, 2009.

[111] E. Deelman, G. Singh, M. Livny, B. Berriman, and J. Good, "The cost of doing science on the cloud: the montage example," in *Proceedings of the 2008 ACM/IEEE conference on Supercomputing (SC '08)*, (Piscataway, NJ, USA), pp. 1–12, IEEE Press, 2008.

[112] I. Foster, "Service-oriented science," *Science*, vol. 308, pp. 814–817, 2005.

[113] K. Chard, C. Onyuksel, W. Tan, D. Sulakhe, R. Madduri, and I. Foster, "Build grid enabled scientific workflows using gravi and taverna," in *Proceedings of the 4th International Conference on eScience (eScience '08)*, (Indianapolis, IN, USA), pp. 614–619, IEEE Computer Society, 2008.

[114] L. Evans and R. Meade, *Alternating Currents or Counter-Revolution? Contemporary Electricity Reform in New Zealand*. Victoria University of Wellington Press, 2005.

[115] I. Foster, "Globus toolkit version 4: Software for service-oriented systems," *Journal of Computational Science and Technology*, vol. 21, no. 4, pp. 523–530, 2006.

[116] B. P. Miller, D. L. Presotto, and M. L. Powell, "DEMOS/MP: the development of a distributed operating system," *SoftwarePractice & Experience*, vol. 17, no. 4, pp. 277–290, 1987.

[117] R. Buyya and S. Venugopal, "Market-oriented computing and global grids: An introduction," in *Market Oriented Grid and Utility Computing* (R. Buyya and K. Bubendorfer, eds.), pp. 3–27, New York, USA: Wiley Press, 2009.

[118] T. Sandholm, "Distributed rational decision making," in *Multi-Agent Systems: A Modern Introduction to Distributed Artificial Intelligence* (W. G, ed.), MIT Press, 2000.

[119] P. Dasgupta and E. Maskin, "Efficient auctions," *The Quarterly Journal of Economics*, vol. 115, no. 2, pp. 341–388, 2000.

[120] T. Sandholm, "Limitations of the vickrey auction in computational multiagent systems," in *the 2nd International Conference on Multiagent Systems (ICMAS '96)*, pp. 299–306, 1996.

[121] P. Milgrom, "The economics of competitive bidding: a selective survey," in *In Social goals and social organization: Essays in memory of Elisha Pazner* (L. Hurwicz, D. Schmeidler, and H. Sonnenschein, eds.), ch. 9, pp. 261–292, Cambridge University Press, 1985.

[122] M. S. Robinson, "Collusion and the choice of auction," *Rand Journal of Economics*, vol. 16, no. 1, pp. 141–145, 1985.

[123] M. H.Rothkopf and R. M. Harstad, "Two models of bid-taker cheating in vickrey auctions," *Journal of Business*, vol. 68, no. 2, pp. 257–26., 1995.

[124] H. R. Varian, "Economic mechanism design for computerized agents," in *Proceedings of the 1st conference on USENIX Workshop on Electronic Commerce (WOEC'95)*, (Berkeley, CA, USA), pp. 2–2, USENIX Association, 1995.

[125] T. J. Rothkopf, M. H. Teisberg and E. P. Kahn, "Why are vickrey auctions rare?," *Journal of Political Economy*, vol. 98, no. 1, pp. 94–109, 1990.

[126] S. de Vries and R. V. Vohra, "Combinatorial auctions: A survey," *INFORMS Journal on Computing*, vol. 15, no. 3, pp. 284–309, 2003.

[127] J. K. MacKie-Mason and H. R. Varian, "Generalized vickrey auctions." Working paper, University of Michigan, 1994. http://hdl.handle.net/2027.42/50432 [Accessed May 2010].

[128] K. Bubendorfer, B. Palmer, and W. Thomson, "Trust in grid resource auctions," in *Market Oriented Grid and Utility Computing* (R. Buyya and K. Bubendorfer, eds.), Wiley Series on Parallel and Distributed Computing, pp. 541–568, New York, USA: Wiley Press, 2009.

[129] S. Goldwasser, S. Micali, and C. Rackoff, "The knowledge complexity of interactive proof systems," *SIAM Journal on Computing*, vol. 18, no. 1, pp. 186–208, 1989.

[130] M. Naor, B. Pinkas, and R. Sumner, "Privacy preserving auctions and mechanism design," in *Proceedings of the 1st ACM conference on Electronic commerce (EC '99)*, (New York, NY, USA), pp. 129–139, ACM, 1999.

[131] H. Lipmaa, N. Asokan, and V. Niemi, "Secure vickrey auctions without threshold trust," in *Proceedings of the 6th International Conference on Financial Cryptography (FC '02)*, vol. 2357 of *Lecture Notes in Computer Science*, pp. 87–101, 2002.

[132] K. Bubendorfer, I. Welch, and B. Chard, "Trustworthy auctions for grid-style economies," in *Proceedings of the 6th IEEE International Symposium on Cluster Computing and the Grid (CC-GRID '06)*, (Washington, DC, USA), pp. 386–390, IEEE Computer Society, 2006.

[133] K. Bubendorfer and W. Thomson, "Resource management using untrusted auctioneers in a grid economy," in *Proceedings of the 2nd IEEE International Conference on e-Science and Grid Computing (E-SCIENCE '06)*, (Washington, DC, USA), p. 74, IEEE Computer Society, 2006.

[134] T. El Gamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," in *Proceedings of CRYPTO 84 on Advances in cryptology*, (New York, NY, USA), pp. 10–18, Springer-Verlag New York, Inc., 1985.

[135] K. Bubendorfer, "Improving resource utilisation in market oriented grid management and scheduling," in *Proceedings of the 2006 Australasian workshops on Grid computing and e-research (ACSW Frontiers '06)*, (Darlinghurst, Australia), pp. 25–31, Australian Computer Society, Inc., 2006.

[136] T. Sandholm, P. Gardfjäll, E. Elmroth, L. Johnsson, and O. Mulmo, "An ogsa-based accounting system for allocation enforcement across hpc centers," in *Proceedings of the 2nd international conference on Service oriented computing (ICSOC '04)*, (New York, NY, USA), pp. 279–288, ACM, 2004.

[137] J. Shneidman, C. Ng, D. C. Parkes, A. AuYoung, A. C. Snoeren, A. Vahdat, and B. Chun, "Why markets could (but don't currently) solve resource allocation problems in systems," in *Proceedings of the 10th conference on Hot Topics in Operating Systems (HOTOS'05)*, (Berkeley, CA, USA), pp. 7–7, USENIX Association, 2005.

[138] V. Vishnumurthy, S. Chandrakumar, and E. G. Sirer, "Karma : A secure economic framework for p2p resource sharing," in *Proceedings of the 1st Workshop on Economics of Peer-to-peer systems (P2PECON '03)*, (Berkeley, California), June 2003.

[139] D. Irwin, J. Chase, L. Grit, and A. Yumerefendi, "Self-recharging virtual currency," in *Proceedings of the 2005 ACM SIGCOMM workshop on Economics of peer-to-peer systems (P2PECON '05)*, (New York, NY, USA), pp. 93–98, ACM, 2005.

[140] S. Krishnan, B. Stearn, K. Bhatia, K. K. Baldridge, W. W. Li, and P. Arzberger, "Opal: Simpleweb services wrappers for scientific applications," in *Proceedings of the IEEE International Conference on Web Services (ICWS '06)*, (Washington, DC, USA), pp. 823–832, IEEE Computer Society, 2006.

[141] D. Gannon, R. Ananthakrishnan, S. Krishnan, M. Govindaraju, L. Ramakrishnan, and A. Slominski, "Grid web services and application factories," in *Grid Computing: Making the Global Infrastructure a Reality* (F. Berman, G. Fox, and T. Hey, eds.), Wiley Series in Communications Networking & Distributed Systems, Wiley, 2003.

[142] D. L. McGuinness and F. van Harmelen, "Owl web ontology language overview," w3c recommendation, W3C, 2004.

[143] E. Friedman-Hill, *Jess in Action: Rule Based Systems in Java*. Manning Publications, 2003.

[144] S. Godik, T. Moses, A. Anderson, B. Parducc, C. Adams, D. Flinn, G. Brose, H. Lockhart, K. Beznosov, M. Kudo, P. Humenn, S. Godik, S. Andersen, S. Crocker, and T. Moses, "extensible access control markup language (xacml)," oasis standard, OASIS, 2005. http://www.oasis-open.org/committees/download.php/2406/oasis-xacml-1.0.pdf [Accessed May 2010].

[145] P. Ashley, S. Hada, G. Karjoth, C. Powers, and M. Schunter, "Enterprise privacy authorization language (EPAL 1.2)," IBM Research Report RZ 3485 (93951), IBM, 2003. http://www.zurich.ibm.com/security/enterprise-privacy/epal/Specification/index.html.

[146] N. Damianou, N. Dulay, E. Lupu, and M. Sloman, "The ponder policy specification language," in *Proceedings of the International Workshop on Policies for Distributed Systems and Networks (POLICY '01)*, (London, UK), pp. 18–38, Springer-Verlag, 2001.

[147] A. Uszok, J. Bradshaw, R. Jeffers, N. Suri, P. Hayes, M. Breedy, L. Bunch, M. Johnson, S. Kulkarni, and J. Lott, "Kaos policy and domain services: Toward a description-logic approach to policy representation, deconfliction, and enforcement," in *Proceedings of the 4th IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY '03)*, (Washington, DC, USA), p. 93, IEEE Computer Society, 2003.

[148] D. Hirtle, H. Boley, B. Grosof, M. Kifer, M. Sintek, S. Tabet, and G. Wagner, "Ruleml specification (0.91)," standard, RuleML Initiative, 2006. www.ruleml.org/spec [Accessed May 2010].

[149] L. Kagal, M. Paolucci, N. Srinivasan, G. Denker, T. Finin, and K. Sycara, "Authorization and privacy for semantic web services," *IEEE Intelligent Systems*, vol. 19, no. 4, pp. 50–56, 2004.

[150] A. Anderson, "A comparison of two privacy policy languages: EPAL and XACML," Technical Report TR-2005-147, Oracle, 2005.

[151] K. Djemame, J. Padgett, I. Gourlay, K. Voss, and O. Kao, "Risk management in grids," in *Market Oriented Grid and Utility Computing* (R. Buyya and K. Bubendorfer, eds.), New York, USA: Wiley Press, 2009. Risk Management In Grids.

[152] K. Voß, "Enhance self-managing grids by risk management," in *Proceedings of the 3rd International Conference on Networking and Services (ICNS '07)*, (Washington, DC, USA), p. 27, IEEE Computer Society, 2007.

[153] L. A. Gordon, M. P. Loeb, and T. Sohail, "A framework for using insurance for cyber-risk management," *Communications of the ACM*, vol. 46, no. 3, pp. 81–85, 2003.

[154] R. M. Piro, A. Guarise, and A. Werbrouck, "An economy-based accounting infrastructure for the datagrid," in *Proceedings of the 4th International Workshop on Grid Computing (GRID '03)*, (Washington, DC, USA), p. 202, IEEE Computer Society, 2003.

[155] H. Bhargava and A. Bagh, "Tariff structures for pricing grid computing resources," in *Proceedings of the 3rd International Workshop on Grid Economics and Business Models (GECON 06)*, (Singapore), 2006.

[156] D. Neumann, S. Lamparter, and B. Schnizler, "Automated bidding for trading grid services," in *Proceedings of the 14th European conference on Information Systems (ECIS '06)*, (Gothenburg, Sweden), pp. 1307–1315, 2006.

[157] D. Allenotor and R. K. Thulasiram, "Grid resources pricing: A novel financial option based quality of service-profit quasi-static equilibrium model," in *Proceedings of the 9th IEEE/ACM International Conference on Grid Computing (GRID '08)*, (Washington, DC, USA), pp. 75–84, IEEE Computer Society, 2008.

[158] A. Caracas and J. Altmann, "A pricing information service for grid computing," in *Proceedings of the 5th international workshop on Middleware for grid computing (MGC '07)*, (New York, NY, USA), pp. 1–6, ACM, 2007.

[159] M. Satterthwaite and S. Williams, "The bayesian theory of the k-double auction," in *The Double Auction Market: Institutions Theories and Evidence* (D. Friedman and J. Rust, eds.), pp. 99–124, Addison-Wesley, 1993.

[160] B. Schnizlera, D. Neumanna, D. Veitb, and C. Weinhardt, "Trading grid services a multi-attribute combinatorial approach," *European Journal of Operational Research*, vol. 187, no. 3, pp. 943–961, 2008.

[161] J. Stößer, D. Neumann, and C. Weinhardt, "Market-based pricing in grids: On strategic manipulation and computational cost," *European Journal of Operational Research*, vol. 203, no. 2, pp. 464–475, 2010.

[162] M. Becker, N. Borrisov, V. Deora, O. F. Rana, and D. Neumann, "Using k-pricing for penalty calculation in grid market," in *Proceedings of the 41st Annual Hawaii International Conference on System Sciences (HICSS '08)*, (Washington, DC, USA), p. 97, IEEE Computer Society, 2008.

[163] A. Anandasivam, S. Buschek, and R. Buyya, "A heuristic approach for capacity control in clouds," in *Proceedings of the IEEE Conference on Commerce and Enterprise Computing (CEC '09)*, (Washington, DC, USA), pp. 90–97, IEEE Computer Society, 2009.

[164] R. Das, J. E. Hanson, J. O. Kephart, and G. Tesauro, "Agent-human interactions in the continuous double auction," in *Proceedings of the 17th international joint conference on Artificial intelligence (IJCAI '01)*, (San Francisco, CA, USA), pp. 1169–1176, Morgan Kaufmann Publishers Inc., 2001.

[165] A. Bagnall and I. Toft, "Autonomous adaptive agents for single seller sealed bid auctions," *Autonomous Agents and Multi-Agent Systems*, vol. 12, no. 3, pp. 259–292, 2006.

[166] D. Cliff, "Minimal-intelligence agents for bargaining behaviors in market-based environments," Technical Report HPL-97-91, Hewlett Packard Labs, 1997.

[167] N. Borissov and N. Wirström, "Q-strategy: A bidding strategy for market-based allocation of grid services," in *Proceedings of Confederated International Conferences, On the Move to Meaningful Internet Systems (OTM 08)*, vol. 5331 of *Lecture Notes in Computer Science*, pp. 744–761, Springer, 2008.

[168] N. Borissov, "Q-strategy: Automated bidding and convergence in computational markets," in *Proceedings of the 21st Innovative Applications of Artificial Intelligence Conference (IAAI)*, (Pasadena, California), pp. 54–59, July 2009.

[169] C. Watkins and P. Dayan, "Q-learning," *Machine Learning*, vol. 8, no. 3–4, pp. 279–292, 1992.

[170] A. E. Roth and A. Ockenfels, "Last-minute bidding and the rules for ending second-price auctions: Evidence from ebay and amazon auctions on the internet," *American Economic Review*, vol. 92, no. 4, pp. 1093–1103, 2002.

[171] A. Ockenfels and A. E. Roth, "Late and multiple bidding in second price internet auctions: Theory and evidence concerning different rules for ending an auction," *Games and Economic Behaviour*, vol. 55, no. 2, pp. 297–320, 2006.

[172] J. Hou, "Late bidding and the auction price: evidence from ebay," *Journal of Product & Brand Management*, vol. 16, no. 6, pp. 422–428, 2007.

[173] P. Bajari and A. Hortacsu, "Economic insights from internet auctions," *Journal of Economic Literature*, vol. 42, pp. 457–486, 2004.

[174] J.-M. Bramsen, "Bid early and get it cheap - timing effects in internet auctions," MPRA Paper 14811, University Library of Munich, Germany, 2008.

[175] C. Castillo, G. N. Rouskas, and K. Harfoush, "Efficient resource management using advance reservations for heterogeneous grids," in *Proceedings of the 22nd IEEE International Symposium on Parallel and Distributed Processing (IPDPS '08)*, (Miami, Florida USA), pp. 1–12, April 2008.

[176] M. Siddiqui, A. Villazón, and T. Fahringer, "Grid capacity planning with negotiation-based advance reservation for optimized qos," in *Proceedings of the 2006 ACM/IEEE conference on Supercomputing (SC '06)*, (New York, NY, USA), p. 103, ACM, 2006.

[177] A. S. Mcgough, A. Afzal, J. Darlington, N. Furmento, A. Mayer, and L. Young, "Making the grid predictable through reservations and performance modelling," *The Computer Journal*, vol. 48, no. 3, pp. 358–368, 2005.

[178] M. A. Netto, K. Bubendorfer, and R. Buyya, "Sla-based advance reservations with flexible and adaptive time qos parameters," in *Proceedings of the 5th international conference on Service-Oriented Computing (ICSOC '07)*, vol. 4749 of *Lecture Notes In Computer Science*, (Berlin, Heidelberg), pp. 119–131, Springer-Verlag, 2007.

[179] W. Smith, I. Foster, and V. Taylor, "Scheduling with advanced reservations," in *Proceedings of the 14th International Symposium on Parallel and Distributed Processing (IPDPS '00)*, (Washington, DC, USA), p. 127, IEEE Computer Society, 2000.

[180] I. Foster, A. Roy, and V. Sander, "A quality of service architecture that combines resource reservation and application adaptation," in *Proceedings of 8th International Workshop on Quality of Service (IWQoS '00)*, pp. 181–188, 2000.

[181] S. Srinivasan, R. Kettimuthu, V. Subramani, and P. Sadayappan, "Characterization of backfilling strategies for parallel job scheduling," in *Proceedings of the 2002 International Conference on Parallel Processing Workshops (ICPPW '02)*, (Washington, DC, USA), p. 514, IEEE Computer Society, 2002.

[182] O. Arndt, B. Freisleben, T. Kielmann, and F. Thilo, "A comparative study of online scheduling algorithms for networks of workstations," *Cluster Computing*, vol. 3, no. 2, pp. 95–112, 2000.

[183] H. Nakada, A. Takefusa, K. Ookubo, M. Kishimoto, T. Kudoh, Y. Tanaka, and S. Sekiguchi, "Design and implementation of a local scheduling system with advance reservation for co-allocation on the grid," in *Proceedings of the Sixth IEEE International Conference on Computer and Information Technology (CIT '06)*, (Washington, DC, USA), p. 65, IEEE Computer Society, 2006.

[184] M. Netto and R. Buyya, "Rescheduling co-allocation requests based on flexible advance reservations and processor remapping," in *Proceedings of the 2008 9th IEEE/ACM International Conference on Grid Computing (GRID '08)*, (Washington, DC, USA), pp. 144–151, IEEE Computer Society, 2008.

[185] B. Li and D. Zhao, "Performance impact of advance reservations from the grid on backfill algorithms," in *Proceedings of the 6th International Conference on Grid and Cooperative Computing (GCC '07)*, (Washington, DC, USA), pp. 456–461, IEEE Computer Society, 2007.

[186] H. R. Moaddeli, G. Dastghaibyfard, and M. R. Moosavi, "Flexible advance reservation impact on backfilling scheduling strategies," in *Proceedings of the 2008 Seventh International Conference on Grid and Cooperative Computing (GCC '08)*, (Washington, DC, USA), pp. 151–159, IEEE Computer Society, 2008.

[187] A. W. Mu'alem and D. G. Feitelson, "Utilization, predictability, workloads, and user runtime estimates in scheduling the ibm sp2 with backfilling," *IEEE Transactions on Parallel and Distributed Systems*, vol. 12, no. 6, pp. 529–543, 2001.

[188] D. Zotkin and P. J. Keleher, "Job-length estimation and performance in backfilling schedulers," in *Proceedings of the 8th IEEE International Symposium on High Performance Distributed Computing (HPDC'09)*, (Washington, DC, USA), p. 39, IEEE Computer Society, 1999.

[189] C. Castillo, G. N. Rouskas, and K. Harfoush, "On the design of online scheduling algorithms for advance reservations and qos in grids," in *Proceedings of the 21st IEEE International Symposium on Parallel and Distributed Processing (IPDPS '07)*, pp. 1–10, 2007.

[190] L. Liu, Y. Yang, W. Shi, W. Lin, and L. Li, "A dynamic clustering heuristic for jobs scheduling on grid computing systems," in *Proceedings of the First International Conference on Semantics, Knowledge and Grid (SKG '05)*, (Washington, DC, USA), p. 4, IEEE Computer Society, 2005.

[191] Z. Jinquan, N. Lina, and J. Changjun, "A heuristic scheduling strategy for independent tasks on grid," in *Proceedings of the Eighth International Conference on High-Performance Computing in Asia-Pacific Region (HPCASIA '05)*, (Washington, DC, USA), p. 588, IEEE Computer Society, 2005.

[192] S. Paurobally, V. Tamma, and M. Wooldrdige, "A framework for web service negotiation," *ACM Transactions on Autonomous and Adaptive Systems*, vol. 2, no. 4, p. 14, 2007.

[193] H. Ludwig, A. Keller, A. Dan, R. King, and R. Franck, "Web service level agreement (wsla) language specification, version 1.0," specification, IBM Corporation, 2003. http://www.research.ibm.com/wsla/WSLASpecV1-20030128.pdf [Accessed May 2010].

[194] D. D. Lamanna, J. Skene, and W. Emmerich, "Slang: A language for defining service level agreements," in *Proceedings of the 9th IEEE Workshop on Future Trends of Distributed Computing Systems (FTDCS '03)*, (Washington, DC, USA), p. 100, IEEE Computer Society, 2003.

[195] A. Paschke, J. Dietrich, and K. Kuhla, "A logic based sla management framework," in *Proceedings of the Semantic Web and Policy Workshop (SWPW) in Conjunction with the 4th International Semantic Web Conference (ISWC '05)*, pp. 68–84, 2005.

[196] H. Ludwig, A. Dan, and R. Kearney, "Cremona: an architecture and library for creation and monitoring of ws-agreents," in *Proceedings of the 2nd international conference on Service oriented computing (ICSOC '04)*, (New York, NY, USA), pp. 65–74, ACM, 2004.

[197] N. Oldham, K. Verma, A. Sheth, and F. Hakimpour, "Semantic ws-agreement partner selection," in *Proceedings of the 15th International World Wide Web Conference (WWW '06)*, pp. 697–706, 2006.

[198] P. Karänke and S. Kirn, "Service level agreements: An evaluation from a business application perspective," in *Proceedings of eChallenges (e-2007)*, 2007.

[199] K. Bubendorfer, P. Komisarczuk, K. Chard, and A. Desai, "Fine grained resource reservation and management in grid economies," in *Proceedings of the 2005 International Conference*

on Grid Computing and Applications (GCA '05) (H. R. Arabnia and J. Ni, eds.), (Las Vegas, Nevada, USA.), pp. 31–38, June 2005.

[200] A. Sulistio, K. H. Kim, and R. Buyya, "Managing cancellations and no-shows of reservations with overbooking to increase resource revenue," in *Proceedings of the 2008 Eighth IEEE International Symposium on Cluster Computing and the Grid (CCGRID '08)*, (Washington, DC, USA), pp. 267–276, IEEE Computer Society, 2008.

[201] K. Czajkowski, I. Foster, and C. Kesselman, "Resource co-allocation in computational grids," in *Proceedings of the 8th IEEE International Symposium on High Performance Distributed Computing (HPDC '99)*, (Washington, DC, USA), p. 37, IEEE Computer Society, 1999.

[202] I. Foster, C. Kesselman, G. Tsudik, and S. Tuecke, "A security architecture for computational grids," in *Proceedings of the 5th ACM conference on Computer and communications security (CCS '98)*, (New York, NY, USA), pp. 83–92, ACM, 1998.

[203] A. Iamnitchi and I. Foster, "A peer-to-peer approach to resource location in grid environments," in *Grid resource management: state of the art and future trends*, pp. 413–429, Norwell, MA, USA: Kluwer Academic Publishers, 2004.

[204] D. Talia and P. Trunfio, "Peer-to-peer protocols and grid services for resource discovery on grids," in *Grid Computing: The New Frontier of High Performance Computing, in Advances in Parallel Computing* (L. Grandinetti, ed.), vol. 14 of *Advances in Parallel Computing*, Elsevier Science, 2005.

[205] P. Trunfio, D. Talia, H. Papadakis, P. Fragopoulou, M. Mordacchini, M. Pennanen, K. Popov, V. Vlassov, and S. Haridi, "Peer-to-peer resource discovery in grids: Models and systems," *Future Generation Computer Systems*, vol. 23, no. 7, pp. 864–878, 2007.

[206] C. Reich, K. Bubendorfer, and R. Buyya, "An autonomic peer-to-peer architecture for hosting stateful web services," in *Proceedings of the 8th IEEE International Symposium on Cluster Computing and the Grid (CCGRID '08)*, (Washington, DC, USA), pp. 250–257, IEEE Computer Society, 2008.

[207] A. Juels and M. Szydlo, "A two-server, sealed-bid auction protocol," in *Proceedings of the 6th International Conference on Financial Cryptography (FC '03)*, vol. 2357 of *Lecture Notes in Computer Science*, pp. 72–86, Springer, 2003.

[208] R. Raman, M. Livny, and M. Solomon, "Matchmaking: Distributed resource management for high throughput computing," in *Proceedings of the 7th IEEE International Symposium on High Performance Distributed Computing (HPDC '98)*, (Washington, DC, USA), p. 140, IEEE Computer Society, 1998.

[209] "GT 4.0 WS GRAM: job description schema document," specification, Globus Alliance.

[210] P. Andreetto, S. Borgia, A. Dorigo, A. Gianelle, M. Mordacchini, M. Sgaravatto, L. Zangr, S. Andreozzi, V. Ciaschini, C. D. Giusto, F. Giacomini, V. Medici, E. Ronchieri, and V. Venturi, "Practical approaches to grid workload and resource management in the egee project," in *Proceedings of the Conference on Computing in High Energy and Nuclear Physics (CHEP 04)*, pp. 899–902, 2004.

[211] A. Streit, D. Erwin, D. Mallmann, R. Menday, M. Rambadt, M. Riedel, M. Romberg, B. Schuller, and P. Wieder, "Unicore - from project results to production grids," in *Grid Computing: The New Frontiers of High Performance Processing, Advances in Parallel Computing.* (L. Grandinetti, ed.), vol. 14, pp. 357–376, Elsevier, 2005.

[212] S. McGough, "A common job description markup language written in xml." http://www.lesc.ic.ac.uk/projects/jdml.pdf [Accessed May 2010], 2003.

[213] W. Lee, A. McGough, S. Newhouse, and J. Darlington, "A standard based approach to job submission through web services," in *Proceedings of the UK e-Science All Hands Meeting*, (Nottingham, UK), pp. 901–905, 2004.

[214] M. Riedel, B. Schuller, D. Mallmann, R. Menday, A. Streit, B. Tweddell, M. S. Memon, A. S. Memon, B. Demuth, and T. Lippert, "Web services interfaces and open standards integration into the european unicore 6 grid middleware," in *Proceedings of the 11th International IEEE Enterprise Computing Conference Workshop (EDOCW '07)*, (Washington, DC, USA), pp. 57–60, IEEE Computer Society, 2007.

[215] D. Roberts and R. Johnson, "Evolving frameworks: A pattern language for developing object-oriented frameworks," in *Pattern Languages of Program Design 3*, Addison Wesley, 1997.

[216] R. Krishnan, "Grid economics: A selective discussion of two research problems," *Journal of Grid Computing*, vol. 6, no. 3, pp. 219–224, 2008.

[217] E. Yom-Tov and Y. Aridor, "Improving resource matching through estimation of actual job requirements," in *Proceedings of the 15th IEEE International Symposium on High Performance Distributed Computing (HPDC '06)*, pp. 367–368, 2006.

[218] M. Dobber, R. van der Mei, and G. Koole, "A prediction method for job runtimes on shared processors: Survey, statistical analysis and new avenues," *Performance Evaluation*, vol. 64, no. 7-8, pp. 755–781, 2007.

[219] W. Smith, I. T. Foster, and V. E. Taylor, "Predicting application run times using historical information," in *Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing in the 12th International Parallel Processing Symposium / 9th Symposium on Parallel and Distributed Processing (IPPS/SPDP '98)*, vol. 1459 of *Lecture Notes in Computer Science*, (London, UK), pp. 122–142, Springer-Verlag, 1998.

[220] N. H. Kapadia, J. A. B. Fortes, and C. E. Brodley, “Predictive application-performance modeling in a computational grid environment,” in *Proceedings of the 8th IEEE International Symposium on High Performance Distributed Computing (HPDC ’99)*, (Washington, DC, USA), p. 6, IEEE Computer Society, 1999.

[221] O. Sonmez, N. Yigitbasi, A. Iosup, and D. Epema, “Trace-based evaluation of job runtime and queue wait time predictions in grids,” in *Proceedings of the 18th ACM international symposium on High performance distributed computing (HPDC ’09)*, (New York, NY, USA), pp. 111–120, ACM, 2009.

[222] R. Wolski, N. T. Spring, and J. Hayes, “The network weather service: a distributed resource performance forecasting service for metacomputing,” *Future Generation Computer Systems*, vol. 15, no. 5-6, pp. 757–768, 1999.

[223] D. Pease, A. Ghafoor, I. Ahmad, D. L. Andrews, K. Foudil-Bey, T. E. Karpinski, M. A. Mikki, and M. Zerrouki, “Paws: A performance evaluation tool for parallel computing systems,” *Computer*, vol. 24, no. 1, pp. 18–29, 1991.

[224] V. Yarmolenko and R. Sakellariou, “Towards increased expressiveness in service level agreements: Research articles,” *Concurr. Comput. : Pract. Exper.*, vol. 19, no. 14, pp. 1975–1990, 2007.

[225] G. Birkenheuer, A. Brinkmann, and H. Karl, “The gain of overbooking,” in *Proceedings of the 14th International Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP)*, vol. 5798 of *Lecture Notes in Computer Science*, pp. 80–100, 2009.

[226] B. C. Smith, J. F. Leimkuhler, and R. M. Darrow, “Yield management at american airlines,” *INTERFACES*, vol. 22, no. 1, pp. 8–31, 1992.

[227] Y. Suzuki, “An empirical analysis of the optimal overbooking policies for us major airlines,” *Transportation Research Part E: Logistics and Transportation Review*, vol. 38, no. 2, pp. 135–149, 2002.

[228] R. Ball, M. Clement, F. Huang, Q. Snell, and C. Deccio, “Aggressive telecommunications overbooking ratios,” in *Proceedings of the 23rd IEEE International Conference on Performance, Computing, and Communications (IPCCC)*, (Phoenix, Arizona), pp. 31–38, 2004.

[229] M. Campbell, “Resource overbooking in a market-oriented grid resource allocation architecture,” honours thesis, School of Mathematicas, Statistics and Computer Science, Victoria University of Wellington, 2008.

[230] C. Chiu and C. Tsao, “The optimal airline overbooking strategy under uncertainties,” in *Proceedings of the 8th International Conference on Knowledge-Based Intelligent Information and Engineering Systems (KES ’04)* (M. G. Negoita, R. J. Howlett, and L. C. Jain, eds.), vol. 3213 of *Lecture Notes in Computer Science*, pp. 937—945, Springer-Verlag, 2004.

[231] J. Subramanian, S. Stidham, Jr., and C. J. Lautenbacher, "Airline yield management with overbooking, cancellations, and no-shows," *Transportation Science*, vol. 33, no. 2, pp. 147–167, 1999.

[232] M. Hovestad, O. Kao, A. Keller, and A. Streit, "Scheduling in hpc resource management systems: Queuing vs. planning," in *Proceedings of the 9th International Workshop Job Scheduling Strategies for Parallel Processing* (D. Feitelson, L. Rudolph, and W. Schwiegelshohn, eds.), vol. 2862 of *Lecture Notes in Computer Science*, pp. 1–20, Springer-Verlag, 2003.

[233] A. Nissimov and D. G. Feitelson, "Probabilistic backfilling," in *Proceedings of the 13th International Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP '07)* (E. Frachtenberg and U. Schwiegelshohn, eds.), vol. 4942 of *Lecture Notes in Computer Science*, pp. 102–115, Springer-Verlag, 2008.

[234] K. T. Talluri and G. V. Ryzin, *The theory and practice of revenue management*, vol. 68 of *International Series in Operations Research & Management Science*. Springer, 2004.

[235] A. Holland and B. O'Sullivan, "Robust solutions for combinatorial auctions," in *Proceedings of the 6th ACM conference on Electronic commerce (EC '05)*, (New York, NY, USA), pp. 183–192, ACM, 2005.

[236] D. P. Porter, "The effect of bid withdrawal in a multi-object auction," *The Review of Economic Design*, vol. 4, no. 1, pp. 73–97, 1999.

[237] J. M. Schopf, I. Raicu, L. Pearlman, N. Miller, C. Kesselman, I. Foster, and M. DArcy., "Monitoring and discovery in a web services framework: Functionality and performance of globus toolkit mds4," Technical Report ANL/MCS-P1315-0106, Argonne National Laboratory, 2006.

[238] W. Thomson, "A framework for secure auctions," msc thesis, School of Mathematics, Statistics and Computer Science, Victoria University of Wellington, 2008.

[239] A. Iosup, H. Li, M. Jan, S. Anoep, C. Dumitrescu, L. Wolters, and D. H. J. Epema, "The grid workloads archive," *Future Generation Computer Systems*, vol. 24, no. 7, pp. 672–686, 2008.

[240] H. Li, "Long range dependent job arrival process and its implications in grid environments," in *Proceedings of the first international conference on Networks for grid applications (GridNets '07)*, (ICST, Brussels, Belgium), pp. 1–8, ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2007.

[241] H. Li and L. Wolters, "Towards a better understanding of workload dynamics on data-intensive clusters and grids," in *Proceedings of the 21st IEEE International Parallel and Distributed Processing Symposium (IPDPS '07)*, p. 60, 2007.

[242] H. Li and R. Buyya, "Model-based simulation and performance evaluation of grid scheduling strategies," *Future Generation Computer Systems*, vol. 25, no. 4, pp. 460–465, 2009.

[243] P. Abry, D. Veitch, and P. Flandrin, "Long-range dependence: Revisiting aggregation with wavelets," *Journal of Time Series Analysis*, vol. 19, no. 3, pp. 253–266, 1998.

[244] H. Li, "Workload dynamics on clusters and grids," *The Journal of Supercomputing*, vol. 47, no. 1, pp. 1–20, 2006.

[245] S. Kavulya, J. Tan, R. Gandhi, and P. Narasimhan, "An analysis of traces from a production mapreduce cluster," in *Proceedings of the 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing (CCGrid)*, pp. 94–103, 2010.

[246] L. Srinivasan and T. Banks, "Web services resource lifetime 1.2," oasis standard, OASIS, 2006. http://docs.oasis-open.org/wsrf/wsrf-ws_resource_lifetime-1.2-spec-os.pdf.

[247] T. Vázquez, E. Huedo, R. S. Montero, and I. M. Llorente, "Gridway internal report: Scalability," tech. rep., Universidad Complutense de Madrid, 2007.

[248] K. Chard, W. Tan, J. Boverhof, R. Madduri, and I. Foster, "Wrap scientific applications as wsrf grid services using gravi," in *IEEE 7th International Conference on Web Services (ICWS '09)*, (Los Angeles, CA, USA), pp. 83–90, IEEE Computer Society, 2009.

[249] W. Tan, K. Chard, D. Sulakhe, R. Madduri, I. Foster, S. Soiland-Reyes, and C. Goble, "Scientific workflows as services in cagrid: A taverna and gravi approach," in *Proceedings of the 7th IEEE International Conference on Web Services (ICWS '09)*, (Washington, DC, USA), pp. 413–420, IEEE Computer Society, 2009.

[250] T. Oinn, P. Li, D. B. Kell, C. Goble, A. Goderis, M. Greenwood, D. Hull, R. Stevens, D. Turi, and J. Zhao, "Taverna/mygrid: aligning a workflow system with the life sciences community," in *Workflows for E-science: Scientific Workflows for Grids* (I. J. Taylor, E. Deelman, D. B. Gannon, and M. Shields, eds.), pp. 300–319, Springer, 2007.

[251] M. Senger, P. Rice, and T. Oinn, "Soaplab – a unified sesame door to analysis tools," in *UK e-Science All Hands Meeting*, pp. 509–513, 2003.

[252] T. Delaittre, T. Kiss, A. Goyeneche, G. Terstyanszky, S.Winter, and P. Kacsuk., "GEMLCA: running legacy code applications as grid services," *Journal of Grid Computing*, vol. 3, no. 1–2, pp. 75–90, 2005.

[253] P. V. Coveney, R. S. Saksena, S. J. Zasada, M. McKeown, and S. Pickles, "The application hosting environment: Lightweight middleware for grid-based computational science," *Computer Physics Communications*, vol. 176, no. 6, pp. 406–418, 2007.

[254] S. J. Zasada, R. Saksena, P. V. Coveney, M. M. Keown, and S. Pickles, "Facilitating user access to the grid: A lightweight application hosting environment for grid enabled computational science," in *Proceedings of the 2nd IEEE International Conference on e-Science and Grid Computing (E-SCIENCE '06)*, (Washington, DC, USA), p. 50, IEEE Computer Society, 2006.

[255] S. Zasada and P. Coveney, "Virtualizing access to scientific applications with the application hosting environment," *Computer Physics Communications*, vol. 180, no. 12, pp. 2513–2525, 2009.

[256] S. Krishnan, K. K. Baldridge, J. P. Greenberg, B. Stearn, and K. Bhatia, "An end-to-end web services-based infrastructure for biomedical applications," in *Proceedings of the 6th IEEE/ACM International Workshop on Grid Computing (GRID '05)*, (Washington, DC, USA), pp. 77–84, IEEE Computer Society, 2005.

[257] S. Krishnan, L. Clementi, J. Ren, P. Papadopoulos, and W. Li, "Design and evaluation of opal2: A toolkit for scientific software as a service," in *Proceedings of the 2009 Congress on Services - I (SERVICES '09)*, (Washington, DC, USA), pp. 709–716, IEEE Computer Society, 2009.

[258] G. Kandaswamy, L. Fang, Y. Huang, S. Shirasuna, S. Marru, and D. Gannon, "Building web services for scientific grid applications," *IBM Journal of Research and Development*, vol. 50, no. 2/3, pp. 249–260, 2006.

[259] S. Hastings, S. Oster, S. Langella, D. Ervin, T. M. Kurc, and J. H. Saltz, "Introduce: An open source toolkit for rapid development of strongly typed grid services," *Journal of Grid Computing*, vol. 5, no. 4, pp. 407–427, 2007.

[260] "Web services business process execution language version 2.0," standard, OASIS, 2007. http://docs.oasis-open.org/wsbpel/2.0/CS01/wsbpel-v2.0-CS01.html [Accessed May 2010].

[261] B. Tieman, "Experiences with gravi," in *NSF Expedition Workshop, The Role of Cyberinfrastructure in Scientific Knowledge: Emergence, Validation, and Peer Review*, 2008.

[262] K. Chard, S. Caton, O. Rana, and K. Bubendorfer, "Social cloud: Cloud computing in social networks," in *Proceedings of the 3rd IEEE International Conference on Cloud Computing (CLOUD 2010)*, (Miami, Florida, USA), 2010.

[263] N. Andrade, F. Brasileiro, M. Mowbray, and W. Cirne, "A reciprocation-based economy for multiple services in a computational grid," in *Market Oriented Grid and Utility Computing* (R. Buyya and K. Bubendorfer, eds.), Wiley Series on Parallel and Distributed Computing, pp. 357–370, New York, USA: Wiley Press, 2009.

[264] Amazon, "Building facebook applications on AWS." http://aws.amazon.com/solutions/global-solution-providers/facebook/ [Accessed May 2010].

[265] R. Curry, C. Kiddle, N. Markatchev, R. Simmonds, T. Tan, M. Arlitt, and B. Walker, "Facebook meets the virtualized enterprise," in *Proceedings of the 12th International IEEE Enterprise Distributed Object Computing Conference (EDOC '08)*, (Washington, DC, USA), pp. 286–292, IEEE Computer Society, 2008.

[266] Z. Guo, R. Singh, and M. Pierce, "Building the polargrid portal using web 2.0 and opensocial," in *Proceedings of the 5th Grid Computing Environments Workshop (GCE '09)*, (New York, NY, USA), pp. 1–8, ACM, 2009.

[267] "Opensocial specification v0.9," specification, OpenSocial and Gadgets Specification Group, April 2009. http://www.opensocial.org/Technical-Resources/opensocial-spec-v09/OpenSocial-Specification.html [Accessed May 2010].

[268] D. Recordon and D. Reed, "Openid 2.0: a platform for user-centric identity management," in *Proceedings of the 2nd ACM workshop on Digital identity management (DIM '06)*, (New York, NY, USA), pp. 11–16, ACM, 2006.

[269] D. Werthimer, J. Cobb, M. Lebofsky, D. Anderson, and E. Korpela, "Seti@home—massively distributed computing for seti," *Computing in Science and Engineering*, vol. 3, no. 1, pp. 78–83, 2001.

[270] M. R. Shirts and V. S. Pande, "Screensavers of the world unite!," *Science*, vol. 290, pp. 1903–1904, 2000.

[271] A. L. Beberg and V. S. Pande, "Storage@home: Petascale distributed storage," in *Procedings of the 21st IEEE International Parallel & Distributed Processing Symposium (IPDPS)*, p. 482, 2007.

[272] L. Peterson and T. Roscoe, "The design principles of planetlab," *ACM SIGOPS Operating Systems Review*, vol. 40, no. 1, pp. 11–16, 2006.