Improving the Performance of Cloud-based Scientific Services

by

Ryan 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
2016

Abstract

Cloud computing provides access to a large scale set of readily available computing resources at the click of a button. The cloud paradigm has commoditised computing capacity and is often touted as a low-cost model for executing and scaling applications. However, there are significant technical challenges associated with selecting, acquiring, configuring, and managing cloud resources which can restrict the efficient utilisation of cloud capabilities.

Scientific computing is increasingly hosted on cloud infrastructure—in which scientific capabilities are delivered to the broad scientific community via Internet-accessible services. This migration from on-premise to on-demand cloud infrastructure is motivated by the sporadic usage patterns of scientific workloads and the associated potential cost savings without the need to purchase, operate, and manage compute infrastructure—a task that few scientific users are trained to perform. However, cloud platforms are not an automatic solution. Their flexibility is derived from an enormous number of services and configuration options, which in turn result in significant complexity for the user. In fact, naïve cloud usage can result in poor performance and excessive costs, which are then directly passed on to researchers.

This thesis presents methods for developing efficient cloud-based scientific services. Three real-world scientific services are analysed and a set of common requirements are derived. To address these requirements, this thesis explores automated and scalable methods for inferring network performance, considers various trade-offs (e.g., cost and performance) when provisioning instances, and profiles application performance, all in heterogeneous and dynamic cloud environments. Specifically, network tomography provides the mechanisms to infer network performance in dynamic and opaque cloud networks; cost-aware automated provisioning approaches enable services to consider, in real-time, various trade-offs such as cost, performance, and reliability; and automated application profiling allows a huge search space of applications, instance types, and configurations to be analysed to determine resource requirements and application performance. Finally, these contributions are integrated into an extensible and modu-

lar cloud provisioning and resource management service called SCRIMP. Cloud-based scientific applications and services can subscribe to SCRIMP to outsource their provisioning, usage, and management of cloud infrastructures. Collectively, the approaches presented in this thesis are shown to provide order of magnitude cost savings and significant performance improvement when employed by production scientific services.

Acknowledgements

First of all, I wish to express my sincere gratitude to my advisors, Kris Bubendorfer and Bryan Ng. Not only have your contributions and guidance been invaluable to this work, but your hilarious combination has helped make my entire PhD a thoroughly enjoyable experience.

I am deeply indebted to both Ravi Madduri and Ian Foster. Ravi, your guidance and mentoring inspired the direction of this work and has led me into a research area I am truly passionate about. Ian, I am very grateful for all of your contributions and support over the last few years.

I would like to thank my examination committee: Ian Warren, Carl Kesselman, and Ian Welch, for making my defence such an enjoyable experience. Your insightful comments and suggestions have strengthened my thesis. I owe a special thanks to Ian Welch, who has also been a fantastic mentor and colleague over the years.

I have been fortunate to work with many outstanding academics throughout my PhD. In particular, I am thankful for the opportunities I have had to collaborate with Rich Wolski, whose dedication and excitement is infectious, Nick Karonis, who never fails to make work fun, and Salman Habib and Katrin Heitmann, for being so inviting and willing to teach cosmology to a random computer science student.

I have also been involved with many great research groups throughout my studies. Thank you to all of the past and present MCS/ECS graduate students and staff that have been a part of my time at VUW. In particular, I am very appreciative to the DSRG and NERG groups for their discussions which have contributed to this work. I am deeply thankful to everyone in the Globus Genomics team. I am also thankful to the wider Computation Institute and Globus teams who made me feel very welcome during my time in Chicago.

I would not have been able to undertake this research without the support of Victoria University of Wellington. Thank you for the scholarships and assistance

that made this work possible. I am also grateful to The University of Chicago and Argonne National Laboratory for supporting my research with funding and networking opportunities. I look forward to a long and prosperous working relationship with these institutions.

I am greatly appreciative for all of the support from my friends and family. Words cannot express how thankful I am to my parents, who have fostered a value in education and always supported my endeavours. To my brother, Kyle, you have been my unofficial advisor and mentor throughout almost all of my studies. I will be forever grateful for all of your patience, assistance, and guidance. Finally, Janelle. None of this would have been possible without your unconditional love and support.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Scientific discovery is currently experiencing a disruptive transition from traditional experimental practices to computational and data-intensive approaches [2]. Such changes to the basic pattern of research have already proved transformative in many domains, from astronomy [3] to zoology [4]. Data is now being created at unprecedented rates [5] with scientific datasets increasing in size exponentially [6]. This results in increased analytical complexity [7], with researchers' computation demands rapidly outgrowing the capabilities of their personal computers [8].

Researchers have historically relied on dedicated High Performance Computing (HPC) infrastructures to satisfy large-scale analytical requirements [9]. However, for many researchers—especially those with sporadic usage—the overhead of purchasing, supporting, and using dedicated infrastructure is cost-prohibitive. It is these same challenges that have underpinned the success of cloud computing in the enterprise world [10]. Cloud computing has commoditised computing capacity and represents a shift in paradigm towards an elastic utility computing model. As the capabilities of cloud providers have increased, cloud platforms have become a viable alternative to HPC platforms for hosting large-scale analyses [11, 12, 13, 14]. The cloud is accessible and affordable [15], with low barriers of entry [16], and reduced operational costs [17] when compared to dedicated HPC platforms. It presents an opportunity to accelerate scientific discovery, providing researchers access to enormous on-demand and elastically scalable compute resources at the click of a button [18].

Analysis of scientific workloads deployed on clouds has highlighted several key challenges, including: limited network performance [19], opaque infrastruc-

ture [20], and economic complexity [21]. While the opaque nature of cloud infrastructure is often considered advantageous, it can restrict the ability to exploit data-locality when performing data-intensive workloads. There are also inherent technical challenges in acquiring, configuring, and managing the virtual machines (or *instances*) leased from cloud providers, for example: 1) Users require an understanding of cloud mechanics, such as the available instance types and pricing models, in order to efficiently utilise cloud platforms. 2) Managing large pools of compute resources is challenging on dedicated infrastructure and doing so on elastic (and often preemptable) cloud infrastructure further complicates the process. Thus, methods are required to automate the provisioning, configuration, management, and termination processes. 3) The cloud is not a panacea for "cheap" and efficient resource usage. In fact, naïve provisioning approaches can result in increased cost and poor execution performance [21, 22]. One way of mitigating these challenges is by employing dedicated services to manage the deployment of analyses over cloud infrastructures.

Science as a service [18]—an increasingly common model for delivering scientific capabilities to users—offers a separation between discipline-specific content and the underlying computational infrastructure. Scientific services [23] provide an opportunity to abstract the technical challenges of using the cloud and expose the benefits of on-demand, elastic, analyses to researchers. Similarly, Scientific gateways [24]—a common type of scientific service which abstract domain-specific HPC computations via easy to use web interfaces—are increasingly being deployed on cloud platforms. Scientific gateways, henceforth called *gateways*, typically provide domain-specific end-to-end solutions by exposing the datasets, analytical tools, and compute infrastructure required to perform analyses. Generally, gateways are tightly coupled with software stacks and the underlying infrastructure supporting the analyses of scientific data, making them difficult to generalise and apply to multiple research domains. Cloud computing provides the ideal platform to facilitate the operation of gateways with readily available resources that can be horizontally scaled on-demand to meet dynamic workload requirements.

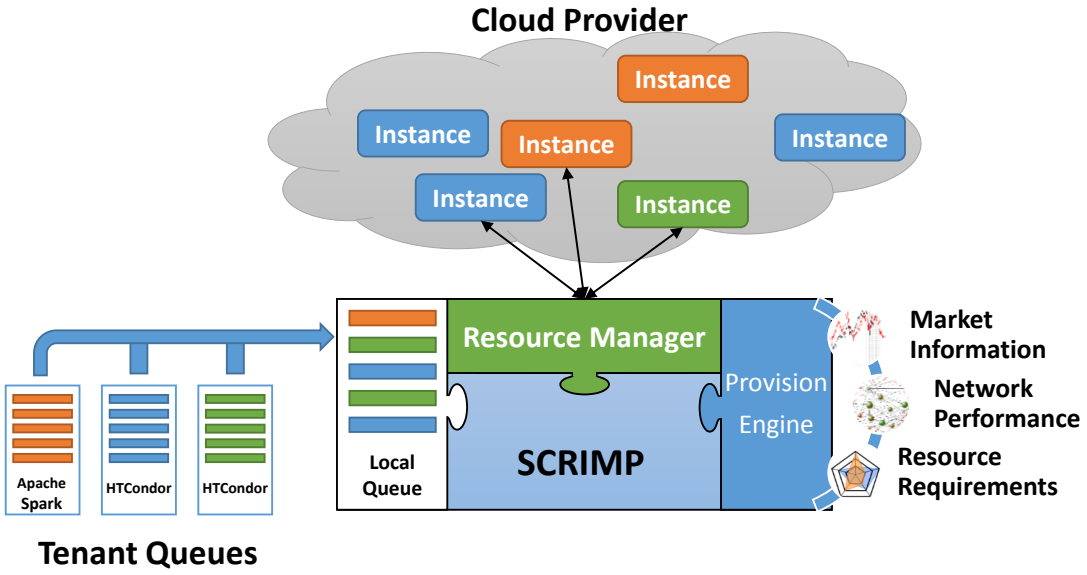


Figure 1.1: An overview of SCRIMP.

1.1 Thesis Synopsis

The research presented in this thesis is motivated by the study of three scientific services spanning genomics, medical imaging, and cosmology use cases. Chapter 3 analyses these three services and poses a set of research questions which are explored throughout the thesis. The research questions pertain to four key challenges faced by cloud-based services: network limitations, economically aware cloud usage, cloud provisioning models, and infrastructure management. The findings and contributions of this thesis are collectively integrated to develop a new Scalable Cloud Resource Management and Provisioning (SCRIMP) service. A conceptual overview of SCRIMP is depicted in Figure 1.1. The figure shows SCRIMP's modular architecture in which different tenant queues, cloud platforms, and provisioning strategies can be integrated. It also shows the service's ability to incorporate economic, network, and application information into the provisioning process. This section briefly describes each of the identified challenges and outlines their investigation.

1.1.1 Cloud Network Limitations

Limited network performance is often identified as the major performance bottleneck when executing scientific applications on the cloud [25]. In addition, com-

mercial cloud infrastructures are typically opaque and do not publish their internal properties. This restricts the ability for developers to minimise the network overhead through the exploitation of data-locality. A key research goal of this thesis is to improve the execution performance of cloud-based scientific applications by enabling network-aware deployments. Chapter 4 adopts network tomography techniques to infer the properties and performance of cloud networks. A network tomography framework is developed and deployed over testbeds of cloud instances to analyse tomographic probes as they traverse the network. The resulting information is used to formulate a set of network performance measurements and metrics, to compute the relative performance and proximity of cloud instances. The framework, called a Network Health Diagnostic System (NHDS), monitors clusters of instances to identify characteristics of the underlying networks, such as high degrees of performance variability, and the sustained nature of performance fluctuations. The NHDS gathers and computes network information over a set of instances by periodically transmitting tomographic probes between instances and aggregating the data into usable network *health scores*. These health scores are used to nominate instances, based on their network performance, to execute workloads. Using the NHDS is shown to substantially improve the execution performance of scientific workloads.

1.1.2 Cloud Economics

Commercial cloud providers offer highly flexible platforms with many different services, instance types, and pricing models. Almost all on-demand cloud-hosted services share a common requirement of having to cost-effectively acquire instances. However, this is a surprisingly difficult task to perform in practice. The Amazon Web Services' (AWS) [26] Elastic Compute Cloud (EC2) [27] uses three different pricing models to lease instances to users: Spot, On-demand, and Reserved. The Spot tier allows users to bid on spare EC2 capacity for a fraction of the instances' On-demand price. However, Spot instance reliability is dependent on users' bid values, cloud market prices, and EC2 capacity. A research goal of this thesis is to automate the selection of instances based on real-time Spot market information in order to minimise the monetary cost of executing workloads. Chapter 5 presents the exploration of cloud pricing models to establish automated and cost-aware resource provisioning techniques. Employing cost-aware techniques, such as broadening search scopes and analysing real-time market information, is

shown to achieve 92.1% cost savings when compared with existing approaches used by a production Globus Galaxies [28] gateways. Further, this thesis explores the benefits of using a predictive approach for computing bid prices with the aim of improving instance reliability. This approach leverages the Durability Agreements From Time Series (DrAFTS) [29] system which provides a list of instance types and minimum bid price that would offer a probabilistic guarantee of instance lifetime. I present these techniques as an open-source, cost-aware, provisioning system which acquires instances for reduced costs.

1.1.3 Provisioning Models

For a provisioning solution to optimally select instances for workloads it must consider both the instance capabilities and the workload's resource requirements. Understanding the tools and applications used by scientists is crucial to ensuring successful execution and improving execution performance [30], as well as reducing the monetary cost of cloud-based analyses. Chapter 6 explores this problem by investigating approaches to quantify a tool's execution requirements, such that they can be suitably matched to cloud instance types. I present a fine-grained profiling service to automate the analysis of scientific tools. The service can deploy, monitor, and analyse tools (executables and docker containers) dynamically over various cloud configurations to construct resource utilisation (e.g., CPU, memory, disk, and network) profiles. I demonstrate that employing tool profiles during the provisioning process can decrease execution time by 15.7% and reduce cost by up to 86.6%.

1.1.4 Infrastructure Management

In addition to resource provisioning, it is crucial for on-demand services to manage acquired cloud infrastructure. Cloud infrastructure is dynamic and ephemeral in nature. For example, the reliability of a Spot instance depends on many factors and may be terminated at any time with just two minutes warning. Persistent monitoring of an instance's state is necessary to identify and react to involuntary terminations. In this thesis, I describe the resource management and analytics platform developed for SCRIMP to monitor and manage clusters of cloud instances. The platform records information regarding each request and fulfilled instance, enabling detailed analysis of a service's cloud utilisation and perfor-

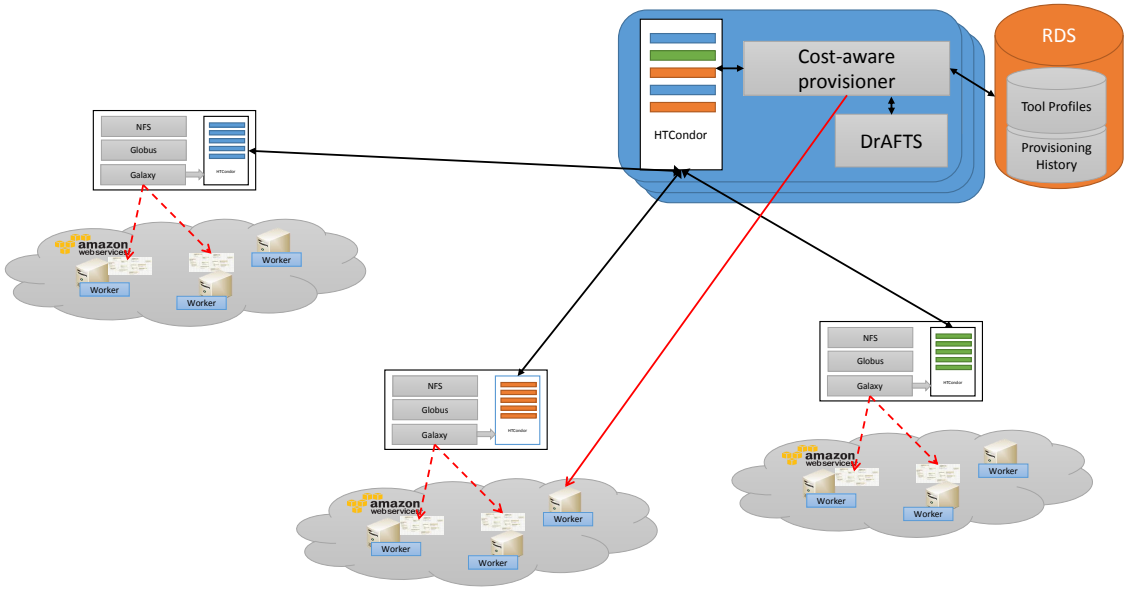


Figure 1.2: SCRIMP serving three Globus Galaxies gateways (a multi-tenant deployment). SCRIMP monitors the queues of each tenant. A cost-aware provisioner consults DrAFTS bid predictions and application profiles to determine which instance type(s) to provision for a waiting job. SCRIMP configures acquired instances, deploys waiting workloads, and monitors the instance for the duration of the tool execution.

mance. The platform also provides global-awareness and facilitates the migration of instances between collaborative services in a multi-tenant deployment. Migration improves global efficiency by increasing instance utilisation and replacing the approximately ten minute start up cost of launching a new instance [31] with a one minute reconfiguration cost.

1.1.5 Provisioning as a Service

Finally, Chapter 7 presents the SCRIMP service. Scientific services can use SCRIMP, as depicted in Figure 1.2, to outsource the creation and management of on-demand cloud infrastructure. SCRIMP uses the cost-aware provisioning system to minimise costs and employs DrAFTS bid prices to improve Spot instance reliability. Tool resource utilisation profiles are incorporated into the provisioning process to guide instance selection, while online profiling dynamically adjusts execution forecasts. Network tomography is used to capture network information between

provisioned instances to enable network-aware decision making. SCRIMP's resource management platform provides detailed analysis of each tenant's cloud utilisation and facilitates the dynamic migration of instances between tenants to improve global efficiency. SCRIMP is currently used by numerous production Globus Galaxies gateways and is publicly available to the community as an open-source project.

1.2 Contributions

This thesis explores and makes contributions in multiple research areas, including: scientific gateways, analysing cloud networks, leveraging cloud economics, tool profiling, and managing cloud infrastructures. Specifically, the major contributions of this thesis are:

1. Development of three scientific services and gateways in collaboration with other researchers. The requirements of these use cases motivate the research presented in this thesis. In particular I have made the following contributions to these projects:

   (a) Design and development of a cloud-based service for reconstructing proton computed tomography (pCT) images on-demand [17]. I implemented the service which utilises more than 100 GPU-enabled cloud instances to reconstruct medical images on-demand. I also created a simulation framework to demonstrate the scalability of the service. This research was conducted in collaboration with researchers from Northern Illinois University (NIU) and Argonne National Laboratory (ANL).

   (b) Design and development of the Portal for Data Analysis Services for Cosmological Simulations (PDACS) [32, 33]. This work has allowed researchers to perform cosmological analyses using Galaxy [34] workflows and seamlessly deploy large-scale analyses over HPC resources. PDACS was developed in a collaboration with researchers from ANL, Fermi National Accelerator Laboratory (FNAL), and the National Energy Research Scientific Computing Center (NERSC).

   (c) I participate in the on-going development of the Globus Galaxies platform [28]. The Globus Galaxies platform combines a number of data

management and analysis services, such as Globus [35] and Galaxy, to simplify the creation of scientific services. It leverages cloud resources to dynamically fulfil scientific workflow requirements. The Globus Galaxies platform has been applied to a number of different scientific domains including genomics [36], climate and policy [37], traumatic brain injury [31], and cardiovascular [38] research. My key contributions have been to design and develop provisioning solutions, enable fine-grained analysis of gateways, and optimise resource selection by profiling tools. The Globus Galaxies project is a collaborative effort between researchers at the University of Chicago and ANL.

2. Development of a network tomography framework to analyse cloud networks and evaluate network performance. The primary contributions of this work are:

   (a) Development of a network tomography framework. The framework enables users to express tomographic probing sequences and efficiently deploy them over large cloud infrastructures to evaluate and infer network properties.

   (b) Analysis of tomographic information to determine properties of commercial clouds. I show that cloud instances experience substantial variations in network performance which persist for prolonged periods of time.

   (c) Formulation of health metrics to derive the relative network performance of cloud instances. These metrics are used to guide instance selection and are demonstrated to improve the execution performance of the pCT codes.

   (d) Development of a Network Health Diagnostic System (NHDS). The system can be deployed over a set of cloud instances and will compute relative health scores between the hosts in order to enable network-aware deployment decisions.

3. Analysis of cloud economics to establish cost-effective techniques to fulfil scientific workloads. I explore various approaches to achieve cost-aware provisioning and demonstrate substantial improvements in both cost and execution time over production Globus Galaxies gateways.

4. Development of an automatic tool profiling service for the cloud. The service enables users to profile executable applications and docker containers over a wide range of cloud instance types. Resource (CPU, memory, disk, and network) utilisation is monitored and used to automatically construct fine-grained tool profiles. In addition, I explore online profiling techniques to capture execution traces and dynamically adjust execution forecasts.

5. Development of a model for migrating cloud instances between scientific services. In collaboration with researchers from University of Wisconsin-Madison, the University of Chicago, and ANL, I have developed a framework for dynamic migration of cloud instances between HTCondor pools. This work has been shown to reduce the number of instances launched and improve the throughput of several Globus Galaxies services.

6. Design and development of SCRIMP—a multi-faceted service for acquiring and managing cloud infrastructure. The service combines the above contributions to improve the performance of scientific services. SCRIMP is unique in the following ways:

   (a) Incorporates real-time market information to dynamically adjust the provisioning approach and is capable of selecting resources across availability zones and acquirement models (On-demand or Spot requests).

   (b) Uses detailed tool profiles to guide provisioning. Profiles are combined with real-time market information to exploit the trade-off in cost and performance.

   (c) Captures network performance information between provisioned instances to enable network-aware decision making.

   (d) Includes a resource management and analytics platform to capture cloud interactions and monitor provisioned instances. The platform enables the analysis of different provisioning approaches and provides cloud utilisation metrics.

   (e) Integrates a predictive bidding strategy, DrAFTS [29, 39] to provide reliability guarantees for provisioned Spot instances. This work was conducted in collaboration with researchers at the University of California at Santa Barbra.

(f) Includes a custom simulation plug-in to emulate the AWS cloud and enable rapid experimentation with little resource costs.

(g) Is capable of dynamically constructing on-demand clusters (HTCondor or Apache Spark) to fulfil workloads.

(h) Enables resource migration to improve global efficiency and reduces instance acquisition time by enabling instances to be shared between tenants.

1.3 Publications

The papers published throughout my PhD candidature are enumerated in this section. The following publications relate to the scientific services discussed in Chapter 3:

- **R. Chard**, R. K. Madduri, N. T. Karonis, K. Chard, K. L. Duffin, C. E. Ordoñez, T. D. Uram, J. Fleischauer, I. T. Foster, M. E. Papka, and J. Winans, *"Scalable pCT image reconstruction delivered as a cloud service,"* IEEE Transactions on Cloud Computing. 2015.

- **R. Chard**, S. Sehrish, A. Rodriguez, R. Madduri, T. D. Uram, M. Paterno, K. Heitmann, S. Cholia, J. Kowalkowski, S. Habib, *"PDACS: a portal for data analysis services for cosmological simulations,"* 9th Gateway Computing Environments Workshop, pp 30-33, 2014.

The following publications are partially derived from the work presented in Chapter 4.

- **R. Chard**, K. Bubendorfer, B. Ng, *"Network health and e-science in commercial clouds,"* Future Generation Computing Systems, 2015.

- **R. Chard**, K. Bubendorfer, B. Ng, *"Network health and e-science in public clouds,"* 10th International Conference on e-Science, pp 309 - 316, 2014.

The following publications are partially derived from the work presented in Chapter 5.

- **R. Chard**, K. Chard, K. Bubendorfer, L. Lacinski, R. Madduri, I. Foster, *"Cost-Aware Cloud Provisioning,"* 11th International Conference on eScience, pp 136-144, 2015.

- **R. Chard**, K. Chard, K. Bubendorfer, L. Lacinski, R. Madduri, I. Foster, *"Cost-Aware Elastic Cloud Provisioning for Scientific Workloads,"* 8th International Conference on Cloud Computing, pp 971-974, 2015.

The following publication is partially derived from the work presented in Chapter 6.

- **R. Chard**, K. Chard, B. Ng, K. Bubendorfer, A. Rodriguez, R. Madduri, I. Foster, *"An Automated Tool Profiling Service for the Cloud,"* Accepted to 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, 2016.

Chapter 2

Related Work

This thesis investigates methods to improve cloud-based scientific services. In particular it focuses on the use of elastic computing infrastructures and methods by which they can be dynamically provisioned to facilitate on-demand analyses. Thus, this thesis overlaps several distinct areas of related literature including: cloud computing, scientific computing, network tomography, cloud economics and cost-aware computing, application profiling, and cloud resource management. The remainder of this chapter discusses these topics and their relevance to the thesis.

2.1 Cloud Computing

The term *cloud computing* encapsulates a general model in which computing capacity (e.g., storage and compute) is accessed over the Internet rather than on a local personal computer. Cloud computing is characterised by a number of differences from traditional computing models. In particular, cloud capabilities are offered via service interfaces through the Internet [40]; cloud models are based on virtualisation layers in which the logical service with which a user interacts is separated from the underlying physical hardware; computing needs can be scaled up or down elastically; and users pay only for the resources they use. Cloud computing models are offered as private, public, or hybrid models. Private clouds are delivered from within an on-premise business' data centre to internal users. Public clouds are offered by third-party providers and resources are sold to public users. Hybrid clouds represent a mixture of public and private models allowing workloads to migrate between the two depending on requirements. This thesis

focuses primarily on public cloud models.

Broadly, cloud services can be categorised as: Infrastructure-, Platform-, or Software-as-a-Service providers; or IaaS, PaaS, and SaaS, respectively [41].

- IaaS encompass services that provide low-level computing infrastructure with the fewest abstractions, such as virtual machines and raw disk storage. Prominent IaaS clouds include Amazon Web Services' [26] (AWS) Elastic Compute Cloud [27] (EC2) and Microsoft Azure [42].
- PaaS are offered at a higher abstraction level and provide complete application stacks for users to develop and deploy applications. Google App Engine [43] and Heroku [44] are two examples of leading PaaS clouds.
- SaaS encompass the idea of well-defined applications being offered via service interfaces. For example, Dropbox [45] and Google Docs [46], offer specific applications to users over the Internet.

This thesis focuses on public IaaS clouds (primarily AWS) where virtual machines of various size and capability are leased to users on-demand. The remainder of this thesis will refer to IaaS clouds simply as clouds.

2.1.1 Scientific Cloud Computing

Many studies have explored the feasibility of hosting scientific applications on the cloud [47, 48, 49, 13, 12]. Researchers have investigated commercial clouds, such as Microsoft Azure [50] and AWS [19], and private clouds [51]. Cloud instances have been shown to possess the necessary compute capabilities to meet the execution requirements of scientific applications [8]. However, network performance is frequently identified as a key restriction when using cloud, rather than traditional High Performance Computing (HPC) infrastructures (Grids, clusters, and supercomputers), for HPC applications [25].

The network performance of standard EC2 instance types has been found to be one to two orders of magnitude worse than InfiniBand [52, 53], a technology commonly used by dedicated HPC platforms. Scientific and HPC applications often have vastly different quality of service requirements than the applications for which cloud computing is commonly used (e.g., e-commerce applications [52]). HPC applications can be extremely sensitive to bandwidth and latency variations, where small overheads can compound to substantially affect an applica-

tion's overall performance. Comparisons between commercial clouds and traditional HPC resources (using NAS Parallel Benchmarks (NPB) [54]) have found the network infrastructure behind clouds to be a limiting factor when performing scientific analyses [25]. Concerns have also been raised regarding the economic models employed by cloud providers, especially when moving and analysing large scientific data [47]. However, the majority of this research was conducted prior to the inclusion of AWS cluster compute and network specialised instance types [55], which are specifically designed to meet the demands of HPC workloads.

Despite these limitations, scientific cloud computing is rapidly being adopted across almost all scientific domains. A 2013 survey [56] on the use of clouds in research and education reported that cloud resources have been successfully adopted in projects spanning over 25 scientific domains, from science and engineering, as well as humanities, arts, and social sciences. On-demand access to burst compute through public clouds and support for high throughput scientific workflows were found to be two of the main causes of adoption. The growth of cloud computing as a viable platform for science has also been proven via scientometric analysis [57]. With an emphasis on current trends toward Big Data and data analytics scientifically purposed clouds, such as Magellan [58], are also finding success exploring how the cloud paradigm can be used to specifically serve future data-intensive workloads.

The cloud is becoming an increasingly pervasive component of scientific discovery. In turn, researchers are now faced with a need for the skills and capabilities to utilise cloud platforms effectively. However, these skills are not yet common and efficient usage of cloud infrastructure is technically challenging: naïve use can result in unnecessary performance and cost overheads [21]. Scientific services present a promising solution to this problem by arbitrating the execution of analytical tools over elastic cloud infrastructures.

2.1.2 Cloud-based Scientific Services

Science as a Service [23], the idea of publishing scientific data and applications through the Internet, is now a common model for delivering scientific capabilities (e.g., via science gateways [18]). Science gateways expose tools, applications, and data through a portal or suite of applications [24] while abstracting the computational infrastructure underlying the gateway. Gateways increase access to

reliable and supported analytical tools without the need for researchers to install, maintain, or even have an in-depth understanding of the software or underlying infrastructure.

Science gateways are typically comprised of four layers: the user layer, application layer, job and data management layer, and the underlying computing infrastructures [59]. Gateway development tools, such as Apache Airavata [60], Agave [61], and Globus [62], facilitate the construction of science gateways by providing the necessary APIs and libraries. Similarly, the Globus Galaxies [28] platform expedites the creation of gateways by enabling the rapid deployment of cloud based installations of Galaxy [34] for domain-specific research objectives. Galaxy, Kepler [63], and Taverna [64], are workbenches that provide workflow management capabilities. These enable the creation, editing, submission, and sharing of workflows as well as support the inclusion of custom tools and datasets that can be shared with the community.

Many science gateways offer domain-specific end-to-end research solutions, where data, analytical tools, and a compute infrastructure are delivered as a standalone service. Gateways are often tightly coupled with software stacks and the underlying compute infrastructure, thus increasing the usability of analytical applications and complex infrastructures [59]. Cloud computing provides an ideal platform for hosting scientific gateways [65, 66, 67]. Clouds provide affordable access to resources that can be horizontally scaled on-demand in order to fulfil sporadic workload requirements. However, there are many challenges that restrict cloud-based scientific services, such as cost-effectiveness, network overheads, and resource management, that must still be addressed.

2.2 Commercial Cloud Platforms

Commercial cloud computing platforms expose computing infrastructure as an on-demand utility [40]. The cloud provides users access to an enormous source of elastic and affordable computing resources [15], with low barriers to entry [16]. Large commercial clouds are built upon geographically distributed data centres. For example, Amazon Web Services' EC2 [27], a leading cloud provider and a key focus of this thesis, is organised into nine geographical *regions*. These are essentially distinct instantiations of the EC2 service, with each region typically offering a subset of virtual machine types and AWS services. Regions are di-

vided into a small set of *availability zones*, where one availability zone can fail independently of other availability zones within the same region. A virtual machine (or *instance*) is provisioned within a single region and availability zone. EC2 availability zones impose data transfer restrictions (both monetary and performance) between distinct zones, thus is it optimal to group communication-intensive workloads within a single availability zone. More fine-grained topological information regarding how instances are connected is not exposed to the end user.

Today's data centres can contain hundreds of thousands, if not millions, of computers [6] and require specialised network topologies to support the full aggregate bandwidth of a large scale cluster. Understanding the architecture and topology of a cloud network is essential when trying to derive meaning from network measurements. Many commercial clouds employ multi-level tree topologies of switches and routers to interconnect data centre resources [68]. A common tree design is divides resources into three tiers: core, aggregation, and edge. These topologies often rely on a *fat-tree* design, where *fatter* network connections are used as you move up the tree [69].

2.2.1 Virtualisation

Cloud providers use virtualisation technologies to offer a wide range of virtual machines with different configurations and capabilities. Xen [70] (used by AWS) and Hyper-V [71] (used by Microsoft Azure) are two prominent hypervisors used in cloud data centres. Hypervisors create and manage virtual machines, enabling multiple operating systems to execute concurrently on shared hardware. Virtualisation allows applications to be dynamically configured, deployed, migrated, and suspended. Virtualisation also provides security and manageability through isolation, while emphasising high degrees of availability and recovery [6]. Many open-source virtualisation projects, such as Nimbus [48], OpenNebula [72], and OpenStack [73], build upon hypervisor technology and can transform local clusters into IaaS clouds.

Although virtualisation has traditionally been blamed for performance overheads [74], modern hardware support is said to reduce these restrictions [6]. Research has demonstrated no statistically significant overhead caused by Xen for many HPC applications [75]. In fact, a range of particle physics applications have been shown to execute equally well on virtualised and native systems [76].

A comparison between HPC benchmarks (HPCC [77] and NPB [54]) illustrate that virtualisation results in only minimal overhead [78]. However, the authors explain that virtualisation effects each application differently. Other work has demonstrated that virtualisation can negatively affect application performance, particularly those that are sensitive to latencies, as overhead increases with the number of virtual machines deployed to shared hardware [79].

2.2.2 Cloud Usage

IaaS clouds offer resources in predefined *instance types* – virtual machines defined with set resource capabilities in various units (e.g., 1 vCPU, 8GB of memory, etc). A variety of instance types are typically offered to provide flexibility for users to select instances and accurately meet the demands of custom use cases [80]. AWS alone offers 48 distinct instance types (as of April 2016), each of which are offered with set disk, memory, and CPU capabilities [27]. Instance types are categorised into groups, known as *families*, which are optimised for storage, network, compute, memory, or general purposes. Other families include specialised hardware (e.g., cluster and GPU-enhanced instances). Each family is comprised of multiple instance types which vary by size and capability. This diversity allows users to lease resources which best match the requirements of custom workloads.

AWS charges users hourly for the occupancy of instances. Users are billed for full hours of usage, regardless of whether the instance is terminated during a billable hour. AWS provides three pricing models to lease instances: *Reserved*, *On-demand*, and *Spot*. On-demand instances incur an advertised instance/hour price, allowing a user to request an instance at their convenience, pay the advertised hourly rate, and release the instance at their discretion. Reserved instances allow users to make a one-time, upfront payment, for a fixed duration lease (typically one or more years). In this case the price is based on forecast (light, medium, or heavy) usage. The hourly rate of a reserved instance is typically discounted from the short term On-demand price, making it a cost-effective method of acquiring instances for long periods of time. Spot instances are offered via a dynamic Spot market model in which users bid to reserve instances. Instances are available to users while their bid price exceeds the fluctuating market price. If the market price exceeds the bid price the instance may be terminated involuntarily. The market price is computed to fill the excess supply of instances, where the highest n bids are used to exhaust the pool. Thus, the market price reflects demand

and capacity, if a bid falls below the market price the instance may be terminated. Spot instances are typically offered at a fraction of the On-demand price and potentially offer the most affordable option to acquire instances [81].

Instances acquired through the Spot market are charged the current market price at the beginning of each billable hour [39]. When the instance is terminated by the user they are charged for the full billable hour. However, Spot instances may be terminated with as little as two minutes notice due to fluctuating market prices and AWS silently resizing Spot pools. If an instance is involuntarily terminated, the user is not charged for the occupancy of the partial billable hour.

2.3 Cloud Provisioning

The elastic nature of cloud platforms present opportunities to dynamically acquire infrastructure on which to host services or workloads. Cloud instances can be provisioned on-demand and provide the flexibility to optimally match heterogeneous workloads to instance type capabilities. Importantly, cloud provisioning does not consider the process by which applications are executed, data is staged, execution is monitored, or data is returned. Instead, these tasks are performed by existing frameworks such as the high throughput computing, MapReduce, and general purpose data-processing frameworks (e.g., HTCondor [82], Hadoop [83], and Apache Spark [84]) used to submit and manage executions.

Programmatically selecting and provisioning instances is fundamental to the operation of almost every on-demand cloud service. AWS facilitates programmatic provisioning and management of cloud instances via an Application Program Interface (API) and Software Development Kits (SDK), such as Python-boto [85]. However, automating the provisioning process requires comprehensive consideration of instance types, services, resource requirements, provisioning models, bidding strategies, and market price information. This thesis investigates techniques to improve automated provisioning approaches for scientific services.

AWS provides degrees of auto-scaling through various services, such as Spot Fleets [86] (dynamic Spot instance management) and Elastic MapReduce [87] (on-demand MapReduce). Although these services are capable of dynamically acquiring instances to provide reliability and meet demand, they are designed to fulfil a single goal which restricts customisation. Therefore, it is common for

cloud-based services to develop custom provisioning systems.

On-demand provisioning of cloud resources is a challenge faced in both academic and commercial projects. A system to provision and manage cloud resources is necessary to utilise the cloud at scale. Fenzo [88] is an example of a commercial solution to cloud provisioning and is used to auto-scale Netflix's cloud infrastructure. Fenzo constructs and manages Apache Mesos [89] frameworks on AWS in order to serve Netflix's content globally. Cloud Scheduler [90] is another cloud resource manager which enables the provisioning of resources to meet the demands of HTC applications [91]. StarCluster [92] is an open-source toolkit for automating the process of building and managing virtual clusters on AWS. Users specify the type and count of nodes to be provisioned. The system supports a plug-in framework to enable custom usage policies to be defined, one such plug-in enables elastic scaling of clusters.

Dynamically provisioning cloud infrastructure can also be used to supplement local infrastructures [93], or facilitate the bursting of local applications to the cloud when necessary. For example, the Eucalyptus cloud [94] is a hybrid cloud that acquires commercial cloud (AWS) instances to supplement local infrastructure.

Cloud provisioning decisions are generally made by monitoring Quality of Service (QoS) requirements or workload intensity to guide when additional resources should be provisioned [95]. Alternative approaches can focus on cost-minimisation [96], deadline constraints [97], or a combination of both [98]. Cost- and deadline-constraints provide Service Level Agreement (SLA) guarantees for when and how expensively a workload will be completed [99, 100].

2.3.1 Cost-Aware Provisioning

The utility computing models employed by cloud computing providers have driven efforts to develop cost-minimising provisioning approaches [96]. Often the motivation for such provisioning approaches is due to cost-constrained workloads [101]. Increasingly, however, researchers are focused on minimising the cost of executing workflows on the cloud [102, 103]. Cost-aware provisioning plays a vital role in almost every provisioning system and is a key focus of this thesis.

The AWS Spot market presents a potentially cost-effective method of acquiring instances. However, even when using the Spot market, ineffective provisioning strategies can result in increased execution costs [21, 22]. To successfully

utilise the Spot market real-time market information must be incorporated into the provisioning process. Cost-aware provisioning has been shown to reduce operational costs [99] and improve the reliability of clusters composed of Spot instances [104]. It can also improve utilisation, performance, and reduce failures [101]. The effect of cost-aware provisioning on instance reliability appears to be at least partially due to the least expensive instances being those that experience low demand.

The Spot market provides a trade-off between cost and reliability [104, 105]. Resources provisioned through the Spot market are fulfilled as long as the specified bid price exceeds the Spot market price. Therefore, instance reliability is dependent on nominating a suitable bid price when requesting Spot instances. Numerous studies have investigated techniques for selecting optimal bid prices when using the Spot market [106, 107, 108]. One such study proposes using a probabilistic decision model to select bid prices for a specific instance type [109], where configurable parameters are used to tune the balance between reliability and monetary savings. This produces levels of confidence in the ability for an instance with a specific bid price to meet QoS objectives and deadlines.

Wolski and Brevik [29] extend previous work [110] on the prediction of bounds for queuing delays and apply their prediction techniques to the Spot market. In this work, Wolski and Brevik present the Durability Agreements From Time Series (DrAFTS) system. DrAFTS is capable of analysing historic Spot market information to predict bid prices for an instance type with 95% confidence that the bid price will not being exceeded over a specific time frame. In this thesis I leverage DrAFTS to employ reliability-aware bid prices as well as guide instance selection in order to minimise monetary costs and improve reliability.

2.4 Managing Cloud Infrastructures

Provisioning cloud infrastructure is only one aspect of using the cloud to execute workloads on-demand. Cloud instances—particularly Spot instances—are unreliable and therefore must be monitored and managed throughout their life cycle to ensure reliable and efficient executions. This thesis presents several resource management techniques to improve the utilisation and reliability of elastic infrastructures after they are provisioned.

The resource management capabilities of a provisioning service are typically

used to track existing infrastructure and guide subsequent auto-scaling decisions. Provisioning and auto-scaling systems react to various QoS requirements and demand [95] to determine what and when additional resources are required. Conversely, this information can be used to downscale elastic infrastructure as demand declines. This is only possible by monitoring the utilisation of an instance once it has been provisioned for a service. The location of where instances are launched can also be used to load-balance services. For example, Netflix's Fenzo [88] system can be directed to load balance workloads by assigning related tasks to the same availability zone.

Cloud instances generally persist until they are specifically terminated by the user, the bid is exceeded, provider capacity is exceeded, or there is a failure. Therefore, it is important to diligently monitor and detect instance state changes. Furthermore, a core requirement of a cloud management system is the need to ensure that unused or unnecessary instances are terminated prior to incurring additional charges. The Globus Galaxies platform employs a self-termination script on each instance to automatically downscale resources [28]. The self-termination script is designed to shutdown the instance immediately prior to the next billable hour if it is not currently being used. Resource management services also become increasingly important as the pool of maintained resources increases in size. Cloud Scheduler [90] employs a technique to manage large pools of resources by logically separating them into clusters for each user.

2.5 Network Tomography

The network that connects cloud instances is often identified as the key performance limitation when executing scientific analyses and the cloud [52, 53]. Network performance is vital to scientific applications due to the size of scientific datasets [6], the cost to move large datasets [111], and the bottlenecks caused during processing [112]. Simply adding compute capacity does not necessarily improve the performance of data-intensive workloads [113]. Thus, several researchers have proposed techniques to minimise the movement of datasets as a means to improve performance [114, 115].

Commercial cloud providers do not advertise their underlying infrastructure to customers. However, network topology information can be crucial to the efficiency of communication-intensive applications [116]. In addition, data-locality is

an increasingly common consideration when optimising the performance of distributed computing applications [112, 117]. In order to exploit data-locality and minimise data transfers, the proximity of instances to one another (and therefore datasets) combined with properties and the topology of the network connecting them are required.

One of the key research topics investigated in this thesis focuses on minimising network-overheads when executing workloads in the cloud. Specifically, network tomography [118] is used to derive network information from provisioned instances to enable network-aware deployments. Network tomography is a tool that uses end-to-end probes to infer and study network properties (e.g. architecture and load). The field of network tomography is relatively young, with Vardi [119] first presenting work toward estimating traffic intensity between hosts in 1996.

Tomographic methods can be broadly classified as either *loss-based* or *delay-based*. Loss-based methods focus on identifying congested links within a network by observing packet loss. Loss-based tomography was initially a popular research area [120, 121]. However, loss-based mechanisms have now become less effective due to the reliability of modern connections, especially under light utilisation. Thousands of probes must be measured before a one percent loss rate can significantly effect the end-to-end performance of a link [122]. Delay-based tomographic methods are more suited to today's networks as they focus on measuring the delay incurred by messages as they traverse a network.

Tomographic inference can be applied to a wide range of areas where knowledge of the network can be exploited. For example, Karonis et al. [123] investigated exploitation of hierarchical network topologies in order to optimise the collective communication algorithms for MPICH-G, an implementation of MPI for Grids. The topology is discovered by Globus, and is automatically detected by MPICH-G [124], to enable applications to infer and then exploit hierarchical topologies. This work was later extended in MPICH-G2 [125] in order to enable communication-intensive workloads to be co-allocated to improve execution performance. This work demonstrates the potential benefits of inferring network properties. When applied to opaque cloud networks, tomography can enable data-locality and reduce the cost of data transfers required by communication-intensive jobs.

2.5.1 Tomographic Techniques

The earliest approaches for determining link delay within a network relied on multicast tomographic techniques [126, 127]. These techniques use multicast end-to-end measurement mechanisms to infer internal network delay in a multicast tree. Network Radar [128], a technique based on Round Trip Time (RTT), improved upon multicast approaches as it does not require the cooperation of receivers, multicast routing capabilities, synchronised clocks, or the capability to capture measurements at both the sending and receiving hosts. Network Radar captures the TCP SYN and SYN-ACK segments of a message to determine the delay in a link.

Novel probing schemes that extend unicast end-to-end measurements have also been explored. Sandwich probing [122] is one such technique that operates by sending packets of different size between two hosts. The scheme transmits two smaller packets separated by an individual large packet. With a priori knowledge of the difference in time between the two smaller packets, the delay induced by the large packet can be measured.

Many established applications and services benefit from recognising network congestion and estimating link capacity. For example, Globus [35] uses UDT, a reliable implementation of UDP, to improve performance of GridFTP data transfers. The mechanism involves sending every 16^{th} data packet and its successor packet back-to-back, creating a packet pair. A median filter on the delay between the arrival of each packet pair can be used to measure the link capacity and amend the transfer accordingly [129].

Network Weather Service (NWS) [130] is a distributed system designed to capture performance measurements of networked resources and produce short-term performance forecasts. In addition to CPU metrics, the NWS captures TCP connection time, end-to-end TCP latency, and bandwidth measurements between sensor pairs. These metrics are then computed to estimate the future that can be expected at the application-level.

Tomographic systems are necessary to deploy and scale tomographic probing schemes over large infrastructures. Typically a collector is placed on a specific host within the infrastructure and will periodically probe other hosts of interest. However, collecting data for each pair of hosts in a large network can be costly and time consuming. An alternative to actively sending probes between each pair of hosts is to identify *subgraphs*, where direct measurements can be replaced

with inference [131].

2.5.2 Tomography in the Cloud

Many network diagnostic tools, such as traceroute, rely on the cooperation of link-layer components [132]. However, most commercial cloud providers do not disclose their network infrastructure, rendering such tools ineffective. The public nature of commercial clouds also presents challenges to the deployment and verification of tomographic systems. Generally, tomography tests are conducted with the specific intention of avoiding interference and are run multiple times on idle networks to suppress the influence of background noise. However, commercial clouds are often noisy and present a number of new challenges, such as providers performing load balancing and multiple virtual machines sharing physical resources.

Battre et al. [133] employed a testbed of 64 Xen VMs to evaluate the ability of various tomographic techniques to determine the network topology of opaque cloud infrastructures. The authors evaluated both loss- and delay-based tomography techniques, showing loss-based approaches to be less effective and finding that RTT measurements outperformed Sandwich probing when using the Robinson-Foulds metric [134].

Cloud MPI (CMPI) [135] is a compelling use case for cloud-based network tomography. CMPI is a network-aware implementation of MPI designed for cloud environments. Here the MPI collective communication algorithms are optimised to utilise network performance information. This approach relies on basic latency and bandwidth matrices to evaluate network performance and optimise the Broadcast and Reduce, and Gather and Scatter operations. Evaluation showed improvements of between 13% and 38%, compared to that of MPICH2 [136]. CMPI exemplifies the potential opportunities that are available within the cloud when applying inferred network information in the deployment of distributed computations.

Researchers have also investigated the potential of network probes to expose cloud instances being migrated during load balancing [137]. The migration detection system collects network measurements between the instance and a set of static *landmarks*. The information is collected and analysed periodically to detect significant performance changes and determine whether an instance is migrated.

This thesis employs network tomography to determine network performance

between provisioned cloud instances. Network information is analysed and used to guide the selection of instances to execute a communication-intensive scientific application.

2.6 Application Profiling

Accurately forecasting the execution requirements of an application is crucial to supporting efficient execution at scale [138], maximising the yield of resources [139], and minimising monetary costs [22]. This thesis explores application profiling techniques to optimise the matching of applications to cloud resources.

The HPC community has long recognised the importance of understanding and forecasting resource requirements for the efficient management and utilisation of resources [30]. Similarly, understanding how a tool (any application or script) executes is critical to deploying them efficiently and cost-effectively on cloud infrastructures [22]. The flexibility of cloud providers can make it especially challenging to select a resource which best meets the requirements of a given tool.

An application profile is the concise description of the execution performance, behaviour, and resource (CPU, memory, network, disk) requirements of an application as it is executed. Many profiling systems have been developed and used to predict job run times [140, 141, 142] as this is an important input to many scheduling algorithms. Although most profiling research has focused on making predictions in homogeneous environments, profiling can be used to predict performance in heterogeneous platforms by analysing executions on different configurations [143].

2.6.1 Profiling Approaches

There are many techniques that can be used to record and analyse application execution and then predict the performance of subsequent executions. Workload modelling can be conducted either *online* or *offline* [144, 145]. Online modelling [146] can be used to construct application profiles of tools by analysing the utilisation and behaviour of a workload in real-time during execution. Whereas offline modelling is achieved by analysing traces and logs from historic executions [144].

Statistical analysis is a common method used to forecast execution performance and has been shown to significantly improve job run time estimates based on large supercomputing logs [139]. Other statistical approaches, such as ARIA (Automatic Resource Inference and Allocation), use mathematical models to estimate job completion times [147]. Most often these approaches are based on historical analysis of execution traces, however small scale benchmarks in test environments can also be used to produce run time estimates [148].

Performance information can be derived by analysing the structured patterns in workload execution logs [149] or by using machine learning algorithms to classify applications [150]. Another approach is to gather execution information via microbenchmarks [151]. Microbenchmarks provide an estimate of a tool's execution signature and can be used to forecast resource requirements. One limitation of this approach is that microbenchmarks do not monitor an application for the duration of its execution, therefore it is difficult to accurately capture the behaviour of applications with varying degrees of resource utilisation. Canonical correlation analysis provides an alternative method of prediction by identifying the relationship between resource utilisation and performance [30].

Virtualisation technologies enable fine-grained resource consumption information to be captured at the virtualisation layer. Virtualisation-level modelling can support the construction of real-time (online) resource consumption profiles [152]. Many of these efforts focus on profiling benchmark tests, in which workloads are analysed, and profiles are established, prior to production deployment. However, similar techniques have been applied as a way of responding to resource utilisation and adapting long running workflows [153].

2.6.2 Profiling in the Cloud

The potential for inefficient execution and poor utilisation is more likely in cloud deployments due to the broad range of services and instance types that are available. Different instance types can vary substantially in terms of capabilities, thus applications can perform quite differently on different cloud configurations [80]. Optimal use of cloud infrastructure requires that knowledge of resource requirements is known a priori and can be used when selecting instance types.

A common use-case for profiling applications is to determine the performance of the underlying infrastructure [154]. The performance of AWS instance types has been measured extensively via benchmark tools (such as HPCC and NPB)

and comparing execution traces between different instance configurations [14]. The performance of AWS instance types has also been compared to traditional HPC infrastructures using a similar method [25] and have been found to have near native computational performance [76]. Workload modelling and analysis of resource utilisation can also aid in the identification of performance degradation and faults. El-Khamra et al. [51] demonstrate this possibility by monitoring scientific simulations executed on the Eucalyptus-based FutureGrid cloud and AWS to identify the cause of overheads.

Many cloud applications (e.g., services) are long running and depend on external factors (such as demand), making them difficult to extensively profile a priori. Thus, cloud services require a dynamic, online, monitoring approach to ensure that QoS requirements are met. AWS offers a built-in cloud monitoring service, known as CloudWatch [155], which captures and publishes utilisation data for provisioned instances. Combining CloudWatch with AWS's Auto Scaling and Elastic Load Balancing features, users can define rules to enable automatic scaling for long running services [156]. However, CloudWatch is primarily focused on instance-based QoS measurements, such as overall CPU utilisation and responsiveness, rather than capturing fine-grained metrics regarding application execution. Customisable event-based monitoring has been explored to facilitate highly expressive monitoring rules [157], however, these approaches focus on primitive instance metrics and are therefore suited to long running cloud services. This thesis investigates a combination of online and offline profiling techniques. In particular, an automatic cloud profiling service is presented as a low overhead method of constructing fine-grain execution profiles.

2.7 Summary

Cloud computing is changing the landscape of scientific discovery. Researchers are rapidly adapting their applications to leverage cloud resources both as an alternative to on-premise infrastructure and also as a means of dynamically scaling computing infrastructure to perform on-demand analyses. However, the sheer number of cloud services and the complexity of provisioning and configuring infrastructure leads to significant technical challenges. Scientific services, such as science gateways, abstract these details and therefore enable cloud infrastructure to be more readily used by scientists. Scientific services can manage cloud

interactions for users and facilitate on-demand analyses in a scalable fashion.

This thesis explores challenges in efficiently delivering cloud-based scientific services. The ultimate goal of this work is to improve performance, increase adoption, decrease costs, and accelerate scientific discovery. Given the breadth of scope of this work this thesis touches upon a multitude of research areas as discussed in this chapter. These areas include: cloud computing, scientific computing, cloud economics and cost-aware computing, cloud infrastructure management, network tomography, and application profiling.

Chapter 3

Science as a Service – Use Cases

Scientific gateways [24] aim to abstract low level technical infrastructure while delivering reliable and usable scientific tools, datasets, and technologies as a service via the Internet. The on-demand and elastic cloud computing model is ideally suited to hosting science gateways due to the sporadic computation requirements of today's researchers. However, as science gateways become yet more pervasive, user expectations and requirements increase. A gateway's efficiencies (or inefficiencies), whether performance-based or monetary, are cumulatively passed on to those that use the service. Therefore, optimisation at the gateway-level can directly enhance the ability of users to conduct research.

This chapter presents the analysis of three services as use cases. I employ a direct observation methodology [158] to draw generalisable conclusions from their requirements. The generalised requirements are then translated into a set of research questions which underpin the research presented in this thesis. The first service (*cf.* Section 3.1) is a scalable Proton Computed Tomography (pCT) image reconstruction service [17]. The pCT service provides on-demand medical image reconstructions by employing more than 100 GPU-enabled cluster instances to reconstruct 100GB datasets. The second service (*cf.* Section 3.2) is the the Portal for Data Analysis Services for Cosmological Simulations (PDACS) [32, 33]. PDACS enables processing of enormous simulation datasets. Thus, it exhibits unique challenges as simulation datasets cannot be readily recomputed, or transferred. The PDACS portal also leverages dedicated infrastructures from both NERSC (the Carver HPC cluster) and ANL (the Magellan Scientific Cloud [58]). Finally, the third service, or more accurately, services (*cf.* Section 3.3) considered are Globus Genomics gateways [36] deployed using the Globus Galaxies plat-

form [28]. The Globus Galaxies platform offers a set of capabilities that can be used to create domain-specific gateways which leverage on-demand cloud infrastructures to execute user-defined workloads.

3.1 Proton Computed Tomography

Proton Computed Tomography (pCT) [1] is a medical imaging modality based on tracking the change in trajectory and energy loss of protons as they pass through an object. pCT imaging was initially developed as a method for acquiring high accuracy images for proton cancer therapy applications. pCT systems provide a number of advantages over traditional X-ray Computed Tomography (xCT) scanners, such as higher accuracy of electron density reconstruction, and lower dose for the same density resolution [159]. A typical configuration for capturing pCT images is shown in Figure 3.1 [1]. This configuration enables the path and energy of individual protons, known as a proton's history, to be recorded as a broad beam is directed at the target.

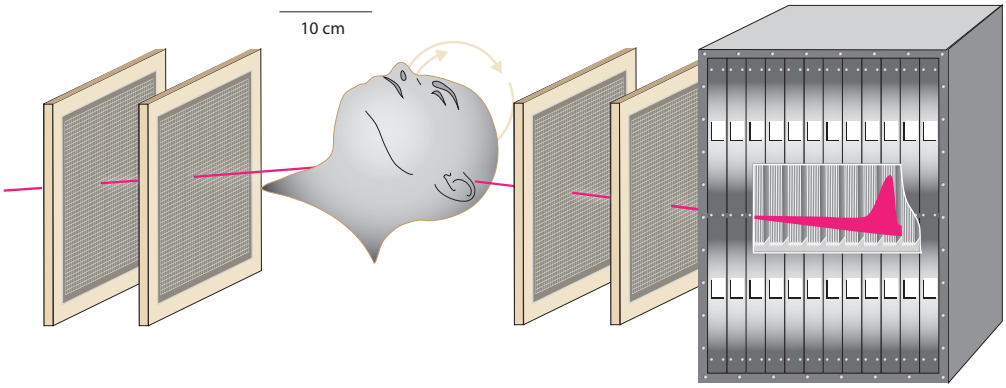


Figure 3.1: The configuration of the NICADD/NIU pCT detector. Protons pass left-to-right through sensor planes and traverse the target before stopping in the detector at the far right [1].

The enhanced accuracy in reconstructed electron density allows physicians to develop more accurate treatment plans, thus sparing healthy tissue during treatment, and allows health care providers to more accurately position patients during treatment sessions. Presently, a patient's position is verified just prior to

receiving each treatment through the use of two-dimensional orthogonal projections but could be significantly improved by instead using the three-dimensional image produced by pCT.

In order to use pCT imaging for position verification, images must be reconstructed in near real-time – initial studies suggest within ten to fifteen minutes [1]. Due to the non-linear path of protons through an imaged material, data reduction techniques cannot be applied to pCT datasets in the same way that they can to other modalities such as positron emission tomography (PET) and xCT. Thus, extremely large datasets must be processed in order to reconstruct an image. It is estimated that the ratio of total protons to total number of 1mm^3 voxels in a target should be greater than 100 to 1 in order to image the target [160]. This gives a conservative upper limit of $\sim 2 \times 10^9$ proton histories required to image a human head. Each history can be represented in 50 bytes, producing a dataset of $\sim$100GB. Considerable compute resources are needed to reconstruct a dataset of this size in a timely manner. For example, Penfold [161] reports that a reconstruction on a small 6GB dataset of 131 million proton histories, using a single CPU and GPU, took almost seventy minutes. At this rate, two billion histories (100GB) would require almost nine hours to process.

3.1.1 pCT Reconstruction Service

To address the needs of providing near real-time reconstructions I created an on-demand and elastic pCT reconstruction service that leverages cloud resources to support time-varying workloads which may involve many concurrent reconstructions of different sizes. The service processes pCT image reconstruction requests by provisioning cloud resources in an on-demand fashion. The service is hosted on AWS and offers a persistent representational state transfer (REST) [162] API. Clients worldwide can connect to this service to request reconstructions. A single instance is responsible for hosting the REST interface, managing data transfers, delivering the shared file system, provisioning clusters, and scheduling reconstructions across these clusters. A high performance, parallel Message Passing Interface (MPI)-based code developed by Karonis et al. [163] is used to perform the pCT image reconstruction. The service uses Globus [35] for asynchronous upload and download of input datasets and reconstructed images.

Figure 3.2 shows the core components of the pCT reconstruction service. At the bottom of the figure, clients, representing hospitals or proton imaging centres,

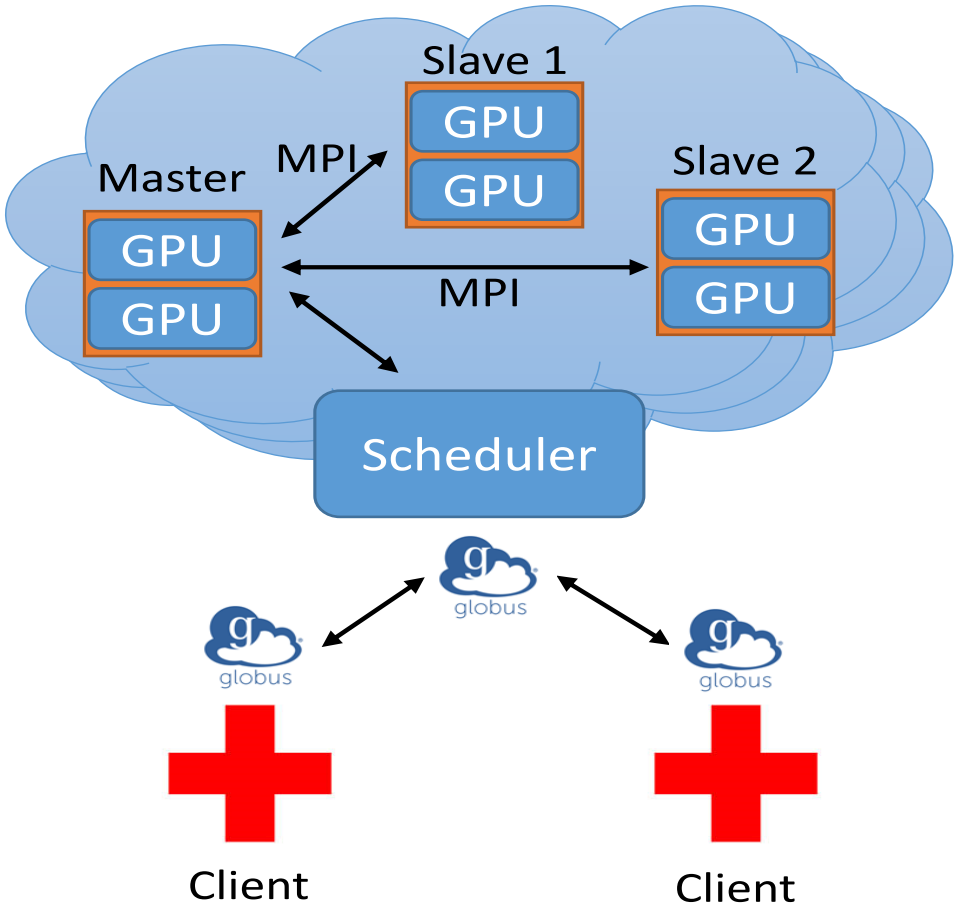


Figure 3.2: An outline of the on-demand pCT reconstruction service.

request image reconstructions from the pCT reconstruction service. The service includes a scheduler that both schedules reconstructions and dynamically provisions and manages clusters as required. Requests submitted to the scheduler specify a priority that represents the type of reconstruction (e.g., treatment plan or position verification). The priority determines whether the work must be immediately processed (which may require starting a new cluster), or whether the job can be optimally scheduled over existing infrastructure. The resulting reconstructions are transferred back to the requesting client.

The pCT service uses GPU-enhanced cluster compute instances (CG1) to perform reconstructions. Each instance has an On-demand price of $2.10 per hour and therefore an entire 120 instance cluster costs $252 an hour. The largest envisioned images, consisting of two billion histories, require approximately nine minutes to reconstruct. Thus six of these reconstructions can be completed within an hour, making the price per reconstruction $42, assuming sufficient demand.

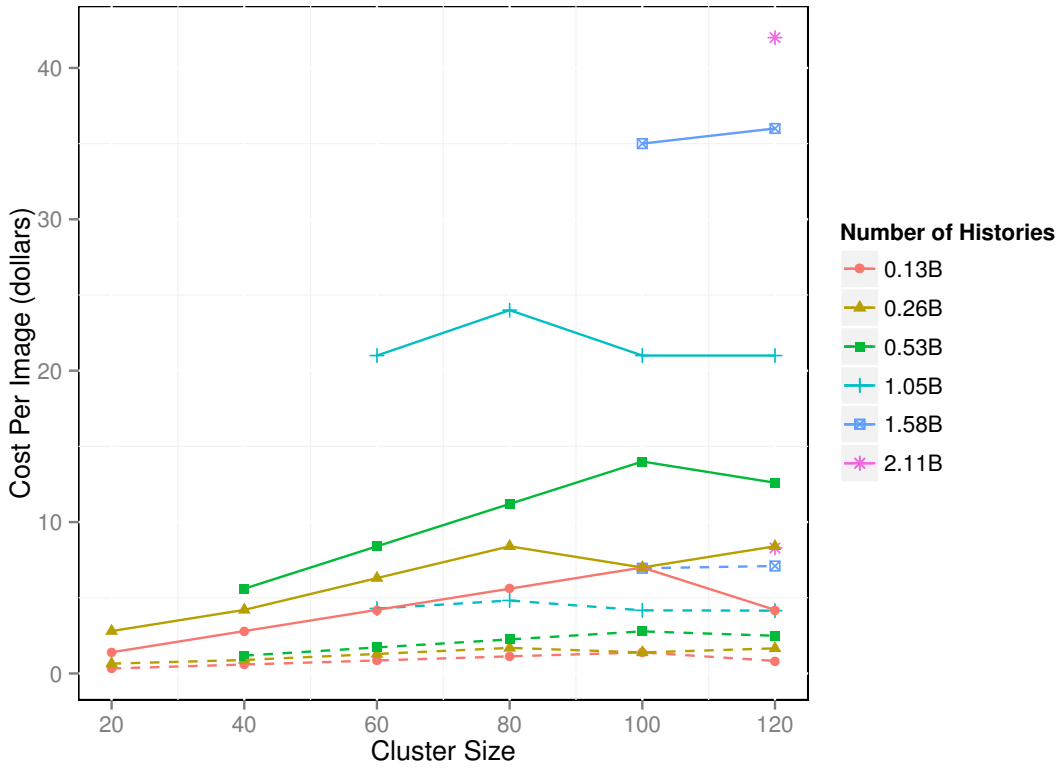


Figure 3.3: Per-image PPN=2 reconstruction costs for various datasets, when using clusters of different sizes that are made up of either entirely On-demand (solid) or entirely Spot (dashed) instances.

As datasets vary significantly in size, smaller and cheaper clusters can be used to service smaller reconstruction requests. Spot instances can provide significantly lower prices than that of On-demand instances. During the initial evaluation in 2013 the average Spot price for CG1 instances was $0.34. Based on analysis of market fluctuations it was determined that a bid price of $0.40 provided high reliability when provisioning large clusters. Using Spot instances a 120 instance cluster could be provisioned for less than $50 an hour. Two billion history reconstructions could therefore be performed for less than $10 each. The projected cost for reconstructing various size datasets under different cluster configurations with both On-demand (solid line) and low Spot price (dashed line) is shown in Figure 3.3. Whereas, a dedicated cluster, such as Gaea, has an initial cost of approximately $800,000, and an annual cost of over $150,000 in staffing and operational expenses [17].

3.1.2 Requirements

On-demand pCT reconstruction presents a number of challenges and requirements due to the size of input data, large and specialised clusters, and the real-time need for image reconstructions. When compared to execution on traditional HPC clusters, the cloud-based approach was found to be able to reconstruct images in a similar amount of time when using clusters with approximately the same amount of collective memory. However, due to the HPC cluster nodes being significantly larger than their cloud instance counterparts (in terms of number of cores, memory, and GPU memory) far more cloud instances were required to reconstruct images comprised of a similar number of proton histories. Given the strict reconstruction time requirements it is paramount that reconstructions are conducted reliably and efficiently. However, when using a cluster composed of Spot instances the probability of errors is significantly larger than using a dedicated HPC cluster. For example, instances occasionally fail to launch and are frequently terminated by price. The real-time requirement of pCT reconstructions emphasises the requirement for reliability as failures during the provisioning, configuration, or management of a cluster can have real-world implications. Therefore the service requires techniques to automatically configure, correct, and detect the state changes (such as failures) of instances through persistent monitoring.

Figure 3.4 depicts the total time required to reconstruct images for various sized input datasets. The total time is calculated by combining the time needed to transfer the input data to the service, reconstruct the image, and transfer the resulting image back to the client. The high transfer times are caused due to the size of the datasets. As is further discussed in [17], the time required to upload input datasets to the service is currently a limitation to the service's ability to facilitate real-time reconstructions. In turn, it is a requirement of the pCT service to facilitate high performance and reliable data transfers.

The distribution phase of the pCT reconstruction code was identified as a key overhead for cloud-based reconstructions. The distribution phase takes almost ten times as long to distributed data over cloud instances when compared to a dedicated cluster. This is due to two reasons: cloud instance memory limitations require that more processes are required to reconstruct an image of the same size; and the cloud network has much lower performance than the InfiniBand [164] supported cluster. As described previously, the cloud network overhead is a key

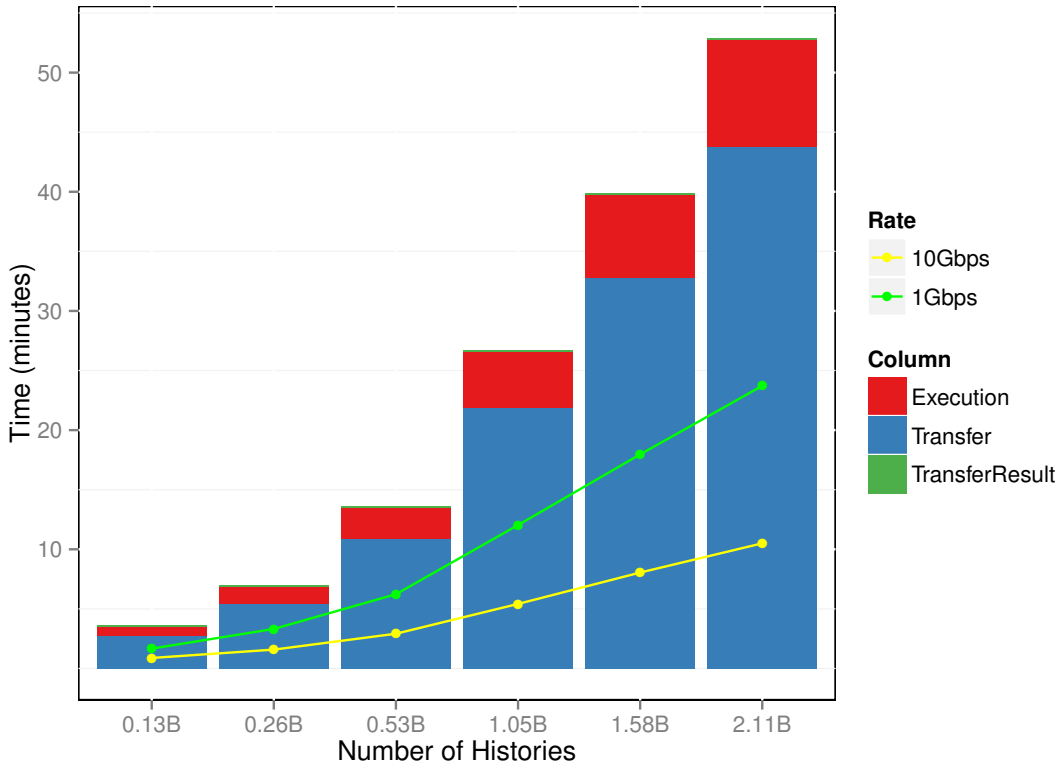


Figure 3.4: The total time required to transfer and reconstruct a pCT image. Each bar shows the time taken to transfer datasets to and from the cloud service, as well as perform the reconstruction. The forecast time required for transfer and reconstruction when supported by a 1-Gigabit and 10-Gigabit network with 100% utilisation are also shown.

challenge to providing an effective cloud-based solution. Thus, the pCT reconstruction service requires methods to minimise the overhead caused by cloud networks and leverage data-locality to improve performance.

Due to the scale of the clusters used to perform reconstructions, the cost for reconstructing a single image can be substantial. Cost-effectively selecting resources is crucial to providing an affordable reconstruction solution. Interestingly, given the specialised nature of the cloud instance type used (and the fact that these instance types are offered in only two AWS regions) the pCT reconstruction service was found to influence the Spot market price when reconstructing large images. This resulted in instances being terminated due to the raising market price and an increased cost for reconstructions. This behaviour has important implications. Notably, such services require more sophisticated provisioning

approaches to understand the impact that they are having on the market. To alleviate risks there is potential to compose clusters of mixed Spot and On-demand instances as well as to distribute computation across availability zones.

The requirements of the pCT reconstruction service use case are:

P1 Using pCT reconstruction as a near real-time position verification tool places strict requirements on the time to deliver results (less than 10 minutes). To meet these goals it is imperative to accelerate the data upload and distribution phase of the reconstruction. Techniques to minimise network overheads must be used to enable the effective deployment on cloud infrastructure.

P2 Resource management techniques are required to reliably configure and scale cloud infrastructures.

P3 Persistent monitoring is necessary to identify terminated Spot instances and react by acquiring additional resources to fulfil workloads.

P4 Economic principles must be leveraged to minimise the cost of image reconstructions. In addition, cost-aware provisioning techniques enable the resources to be distributed across availability zones, improving the services reliability

3.2 PDACS: A Cosmology Portal

Cosmology has made rapid strides in the last two decades based on the remarkable results from a number of large-scale sky surveys carried out across multiple wavebands. Investigations of the mysteries of dark energy and dark matter increasingly rely on large, high-resolution simulations of cosmological structure formation to provide the theoretical underpinnings of cosmological research. These advanced simulations require enormous amounts of compute resources and are generated on leadership computing facilities [165]. The datasets generated are characteristically large and can be hundreds of terabytes in size, extending into the petabytes. Due to the size of these datasets and the complexity of many cosmology simulations, on-demand regeneration of datasets is not feasible. When combined with the complexity of the associated analysis, the number of analyses that can be performed, and therefore science that can potentially be achieved from the simulations, is limited.

The Portal for Data Analysis Services for Cosmological Simulations (PDACS) [32, 33] is designed to increase the value of cosmological simulations by improving the accessibility of the resulting datasets. PDACS aims to enrich the entire scientific community by delivering a service for researchers to contribute and share repositories for tools, workflows, and datasets. PDACS is a workflow engine and scientific gateway for cosmology research. The platform exposes advanced cosmological models as well as the tools and computational resources required to analyse them. PDACS also provides the necessary storage services and manages the output of analyses for researchers.

PDACS is a collaboration between Argonne National Laboratory (ANL), Fermi National Accelerator Laboratory (FNAL), and the National Energy Research Scientific Computing Center (NERSC). PDACS is built upon Galaxy [34], a workflow engine for biomedical research, to create a research environment for cosmology. PDACS deployments are currently in operation at NERSC and ANL, providing a preliminary set of test users access to flexible, large-scale computational resources and storage services through NERSC's HPC infrastructure and ANL's Magellan cloud [58].

3.2.1 PDACS Platform

PDACS enables users to execute and share existing cosmology analysis tools and execute disparate tools in orchestrated workflows with coherent metadata. While basic workflow capabilities are already provided by Galaxy, significant enhancements have been made to make Galaxy suitable for the cosmology community. Galaxy works by allowing researchers to execute *tools*, where tools are wrappers around user defined applications or scripts that invoke a user application written in any language. I repurposed Galaxy to construct the PDACS platform by removing biomedical tools, data types, and formats and replacing them with cosmology equivalents.

Figure 3.5 depicts the process by which PDACS allows researchers to analyse cosmological models. Specifically, this figure shows the PDACS deployment that uses NERSC compute resources. In this case, users are authenticated to access PDACS by using NERSC's NEWT [166] API. PDACS provides a range of tools and datasets which allow researchers to experiment with analyses or construct pipelines. Individual tools and pipelines can be executed by users on supported compute infrastructure. The resulting output files can then be down-

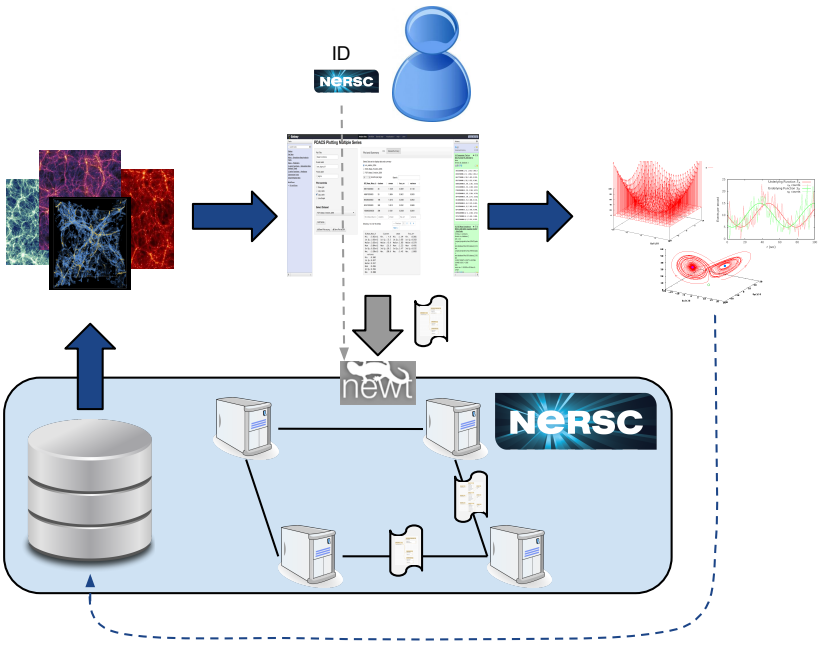


Figure 3.5: The process of using NERSC's PDACS platform to analyse cosmological models.

loaded, shared, or further processed by other analysis or visualisation tools that are supported in PDACS.

PDACS currently exposes a set of 37 cosmological models known as the Coyote Universe [167]. The metadata regarding the generation of each model and the parameters used during the simulation are captured in a novel SQLite [168] data type, developed by researchers at FNAL.

PDACS provides a set of frequently used cosmology tools, allowing users to perform tasks such as finding clusters of particles, known as halos, within simulation models. The provided tools include: Friends-of-Friends (FOF) and Spherical Overdensity (SO) halo finders, halo profile measurements, concentration measurements, binning routines to calculate, e.g., mass functions from the halo finder outputs, a concentration-mass relation emulator, a power spectrum emulator, and measurement tools for two-point statistics, such as the fluctuation power spectrum and correlation function. PDACS has been designed to execute tools in different environments depending on their requirements. For example, computationally intensive tools, such as the halo finder, are automatically submitted as a batch job to be run on cluster and supercomputing resources. Less computationally intensive tools, such as the emulators, are executed on the same node that

manages the Galaxy instance. To enable user-oriented analysis of large datasets PDACS integrates Globus [35] data management services. Globus provides reliable, secure and efficient data-transfer for large datasets in an asynchronous manner. Transfers can be started from within the PDACS instance and uploaded to the supported storage services.

The PDACS service operates as a standalone science gateway and can be deployed on different compute infrastructures. Currently, two distinct PDACS platforms have been deployed at NERSC and ANL. To support these different execution environments PDACS includes new job runners that transparently dispatch jobs to these HPC resources. In order to deploy jobs to NERSC's computing resources the job runner uses the NEWT execution API. NEWT exposes a range of NERSC functionality and resources, providing the ability to authenticate users, monitor system status, and schedule workloads over the HPC resources maintained at NERSC. The NEWT runner allows custom tools to be contributed to the service without users needing to understand the underlying compute infrastructure and submission process. The PDACS platform operating on ANL's Magellan cloud authenticates users with using ANL credentials via Shibboleth [169]. Tool execution on Magellan are associated with a user's Galaxy account to support accountability. In contrast to the NERSC model, the Magellan runner is responsible for provisioning resources before the job is executed, it also releases cloud instances after a job completes.

3.2.2 Requirements

The data-intensive nature and size of the cosmological simulation datasets presents a key challenge to implementing and operating PDACS. The data cannot realistically be downloaded to local user storage, nor can it be reconstructed on-demand. In order to perform analyses special consideration must be given to the location of both the compute resources and the storage location of the datasets. Thus, PDACS requires sophisticated data management techniques. PDACS also presents a requirement for an extensible execution interface, as the platform is designed to be deployed over multiple leadership computing facilities with different execution environments.

The resources requested (e.g., number of NERSC cluster nodes) to execute a tool must be specified prior to job submission. PDACS enables expert users to customise the resource requirements of tools while providing hints as to ap-

propriate configurations for the most frequently run tools. However, once the PDACS service is made publicly available, less experienced users may inadvertently acquire unnecessary resources and incur substantial costs. PDACS requires an automated approach to selecting resources based on tool resource requirements. Similarly, as users contribute additional tools it becomes increasingly complicated for PDACS administrators to manually provide estimates of a tool's resource requirements.

Another requirement of PDACS related to auditing the financial cost associated with HPC resource usage (typically determined via a user allocation). User-level accounting and support for assigning workloads to existing allocations must therefore be supported by PDACS. PDACS accomplishes this by linking authenticated users to existing allocations. Before executing workloads PDACS will first determine that the user has sufficient allocation remaining.

The primary requirements of the PDACS use case are:

C1 Due to the size of cosmological datasets consideration must be given to data location and network performance when performing analyses with PDACS.

C2 Resource provisioning requires knowledge of tool requirements due to the diverse range of computing infrastructure supported and the custom tools provided by PDACS. The hint-based model used by PDACS provides value for users, however automated approaches would facilitate use by less experienced users.

C3 Managing resource utilisation is essential to providing accountability and enforcing usage constraints. PDACs must leverage and record user-level accounting information to restrict the usage of HPC platforms to authenticated users, with existing allocations, while also tracking and exposing usage statistics.

3.3 Globus Galaxies

The Globus Galaxies platform [28] enables the creation of custom SaaS-based science gateways. The platform serves as a rich use case for the work presented in this thesis as it has been used to deploy over 30 gateways across a range of domains, including genomics [36], climate and economic policy [37], medical imaging [31], and medical research [38]. Each gateway is hosted on the cloud

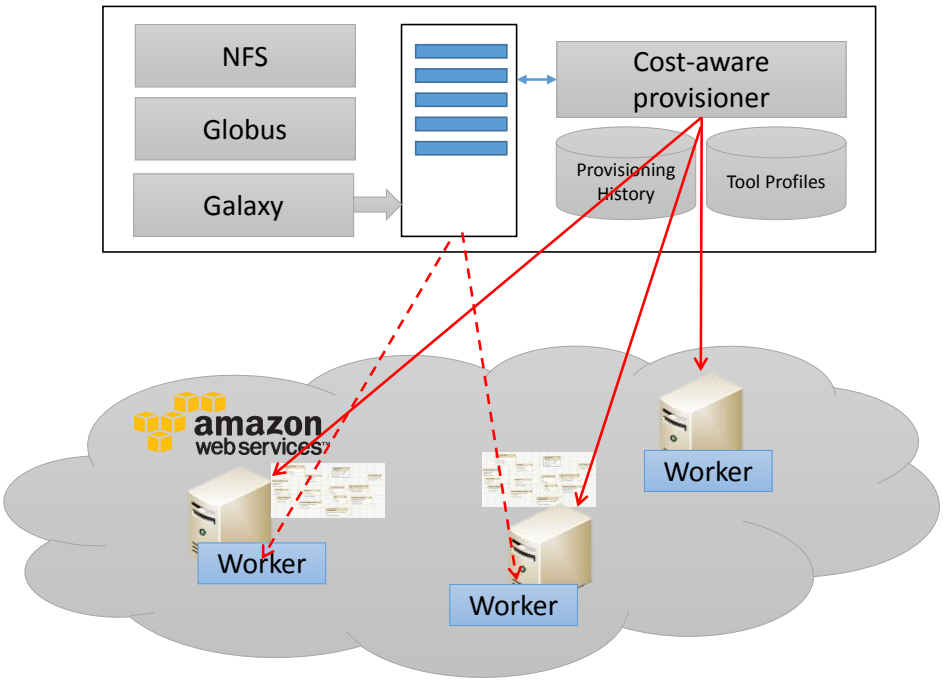


Figure 3.6: An outline of a Globus Galaxies gateway's architecture.

and uses elastic compute resources to dynamically execute the jobs submitted by users. The gateways leverage the Galaxy workflow engine [34], for the construction, execution, and management of parallel end-to-end workflows; Globus [35] to provide reliable and secure data transfers between users and the gateways; Globus Nexus [170] for identity management, authentication, and access control; Swift [171] for parallelising individual workflow components; and HTCondor [82] for job management. Globus Galaxies facilitates rapid development and deployment of scientific gateways which abstract away the complexities of managing advanced software and large scale infrastructures. An outline of the Globus Galaxies architecture is depicted in Figure 3.6.

Globus Genomics [36], the largest instantiation of the Globus Galaxies platform, provides state-of-the-art genomics analysis capabilities and collectively serves over 300 researchers working on various cancers, neurodegenerative disorders, and other disorders. While these gateways provide similar capabilities, and in most cases the same tools, usage can vary significantly between gateways depending on the requirements and research focus of users. Globus Genomics is employed as the primary use case for the majority of the research presented in this thesis. The usage of production Globus Genomics gateways and the ana-

lytical tools provided by the gateways enable evaluation of various techniques to improve the performance and lower the monetary cost of operating scientific gateways. My contributions to the Globus Galaxies platform primarily involve the design and development of cloud provisioning and management solutions. I have also instrumented the gateways for analytics purposes to explore and optimise usage techniques.

3.3.1 Globus Galaxies Platform

Globus Galaxies employs the Galaxy workflow engine to compose workflows by dragging and dropping arbitrarily applications and scripts, then linking their inputs and outputs through a web interface. As described above, Galaxy uses an extensible architecture which allows users to define new data types and tools. Tools are *wrapped* with descriptive XML files which express their input and output data types, the location of the executable, and the arguments and parameters used when invoked. Data types are used to enforce type checking and provide a visual representation for users when combining tools into workflows.

Galaxy is designed to operate over local compute infrastructures, however, it offers an extensible job runner interface for supporting different execution infrastructure. Globus Galaxies gateways each operate an HTCondor master to manage and schedule workloads. The HTCondor runner enables jobs to be submitted directly to the HTCondor queue which then dispatches the jobs over the pool of workers. A shared Network File System (NFS) is used to host and share the tools and datasets required to perform analyses between the service and the workers.

Globus Galaxies gateways relies on an elastic cloud provisioner to acquire and configure cloud instances. The provisioner monitors the HTCondor queue and uses the wait time of idle jobs to determine whether additional instances should be requested. If the queue length, or a job's wait time, exceeds a predefined threshold, the provisioner will request instances to meet the demand of the gateway. Resources are configured to then join the HTCondor pool and begin servicing jobs.

In addition to being able to parallelise the execution of workflows within Galaxy, Globus Galaxies gateways enable the parallelisation of individual workflow components. The parallel scripting language, Swift, is used to compose applications into workflows which can be executed on multicore processors, clusters, grids, and supercomputers. Users can wrap their tools in a Swift wrapper,

allowing large-scale parallel execution of the application transparently to users. Globus Galaxies gateways are tightly integrated with Globus Nexus, allowing the gateways to outsource all identity and group management tasks. Globus Galaxies gateways do not manage users or the identity management workflows (such as resetting forgotten passwords), instead opting to use Globus identities across the platform.

3.3.2 Requirements

The usage of production Globus Galaxies gateways is characteristically sporadic. Figure 3.7 shows the usage (compute hours) of seven production Globus Genomics gateways over a three month period. The usage varies considerably between gateways. The inconsistent usage patterns is exemplified by Gateway 4, which shows considerable usage between weeks 8 and 11, but very little usage over the other periods. In comparison, Gateway 7 shows relatively consistent usage over the period. These usage patterns demonstrate the potential benefits of using on-demand computing infrastructures to fulfil dynamic requirements. Across all Globus Galaxies gateways, there can be thousands of provisioned instances concurrently running. The management and configuration of these instances must be automated to ensure they are capable of performing the required analyses. Analysing the utilisation of the provisioned resources can identify optimisation opportunities and is necessary to provide accounting information to end users. Thus, Globus Galaxies gateways must capture resource utilisation in order to facilitate future analysis and relate workload executions to acquired resources.

Globus Galaxies gateways collectively offer over one thousand custom tools to their users. Table 3.1 presents the 10 most frequently executed tools across seven Globus Genomics gateways. The 10 most frequently executed tools account for 49.9% of all executions and 52.7% of the total execution time. The 20 most frequently executed tools account for 64.7% of executions and 77.3% of the total execution time. This information has been used to prioritise the manual creation of rudimentary resource requirement profiles for the most frequently executed tools. However, this task is labour intensive and error prone. It is not practical to manually construct tool profiles for all of the tools offered through Globus Galaxies gateways. Instead, an automated solution is required to analyse resource requirements of various tools. An individual tool's execution time can drastically differ depending on the size of the input dataset, input parameters,

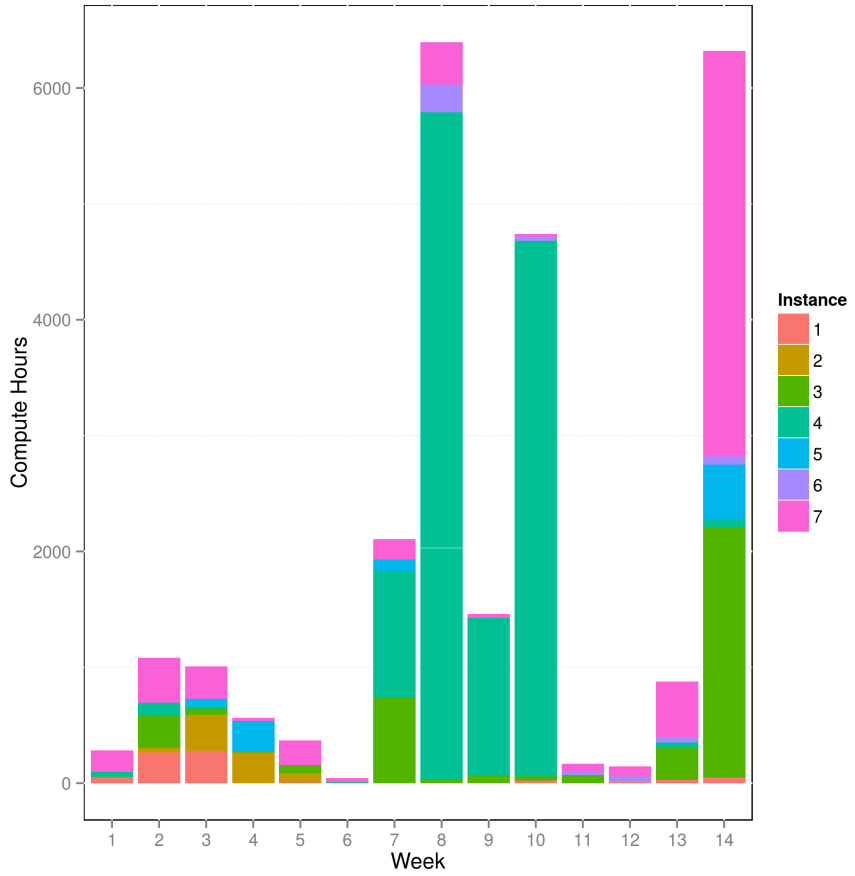


Figure 3.7: Total compute hours used by seven production Globus Genomics gateways over a 14 week period.

and cloud configuration. Therefore, it is necessary for the execution forecasts to employ adaptive techniques to adjust resource requirements based on analysis of production usage.

Minimising operational costs is a primary requirement for Globus Galaxies gateways. One technique that can lower costs is to enable workloads to be co-allocated with resources. To achieve this, the provisioning process must consider tools' resource requirements and use cost-aware techniques to determine when a large instance type should be provisioned to concurrently execute multiple jobs, rather than provisioning many small instance types. This can be achieved by dynamically configuring an instance to divide the resources into smaller units (e.g., HTCondor slots). Therefore, in this case the provisioner must be able to dynamically configure instances as they are provisioned. Similarly, gateways can be configured to prioritise different criteria, such as deadlines and data location.

Table 3.1: Globus Genomics tool usage.

Tool	Avg Exec Time	Freq	Freq (%)	Time (%)
FastQ Parallel	5:48:25	1047	8.40	23.55
BWA MEM	4:34:49	994	7.98	17.64
FastQC	0:34:53	820	6.58	1.85
MarkDups	0:43:53	659	5.29	1.87
ARRG	0:27:13	512	4.11	0.90
Sickle	1:56:27	511	4.10	3.84
Flagstat	0:21:20	489	3.92	0.67
SortSam	0:15:26	421	3.38	0.42
BuildBamIndex	0:23:12	394	3.16	0.59
Bowtie2	0:15:11	273	2.19	0.27

Therefore, there is a requirement for extensible and configurable provisioning algorithms to meet the demands of individual use cases.

The requirements of the Globus Galaxies platform use case are summarised as:

GG1 Resource management techniques are necessary to enable each gateway to reliably scale and manage large clusters of cloud resources.

GG2 Analysing real-time commercial cloud markets is crucial to acquiring instances cost-effectively and providing a cost-effective research platform for users.

GG3 Provisioning resources effectively requires understanding of the fine-grain needs of applications. Reliably profiling tools is necessary to ensure that tools are matched with appropriate cloud instances.

GG4 Expensive data transfers should be minimised by using data-aware deployment techniques.

3.4 Discussion

The scientific services described in this chapter exhibit many similarities and differences, across three scientific domains. Table 3.2 summarises the key characteristic differences between the use cases and the requirements that have been

Table 3.2: Use case characteristics and requirements.

	pCT	PDACS	Globus Galaxies
Characteristic			
Job Type	Single	Workflow	Workflow
Datasets	Uploaded	Hosted	Both
Execution Model	Cloud	Dedicated	Cloud
Multi-tenant	No	No	Yes
Data Sizes (GB)	Hundreds	Thousands	Tens
Job Coupling	Tight	Loose	Loose
Requirement			
Cost Optimisation	High	Low	High
Deadline Optimisation	High	Medium	Medium
Network Optimisation	High	High	Medium
Reliability	High	Medium	Medium
Data Management	High	High	Medium
Auditing	Medium	High	High
Security	High	Low	Medium

observed. The identified characteristics describe the behaviour and usage of each of the use cases. The meaning of these characteristics are as follows:

Job Type describes whether a service executes an individual, heterogeneous, workload (single) or whether it performs a wide range of custom workloads and workflows (workflow).

Datasets indicates how input datasets are contributed to service, e.g. whether they are uploaded by users (uploaded), provided by the service itself (hosted), or a combination of both (both).

Execution Model pertains to the compute infrastructure employed by the service to execute workloads, be it dedicated HPC resources (dedicated) or commercial cloud platforms (cloud).

Multi-tenant describes whether the service has multiple instantiations (yes) or if a single service is operated (no).

Data Sizes refers to the typical size (GB) of input datasets.

Job Coupling indicates whether the service's workloads are deployed across multiple compute nodes (tight) or if workloads are executed on independent resources (loose).

In some ways, these services are representative of a new model of scientific computing, in which elastic computing infrastructures are abstracted behind easy to use web-based interfaces. These services enable elicitation of the requirements and challenges associated with designing, developing, and operating such services. Here the specific requirements derived from each service are generalised into four common themes: network limitations, economic optimisation, resource provisioning, and resource management.

3.4.1 Network Limitations

All three of the scientific services presented operate on large-scale datasets and face challenges related to network performance. Transfers of large data must be reduced to minimise costs and execution time. This requirement is most obvious in the pCT scenario as users require near real-time results. However, the pCT reconstruction service requires that large datasets are uploaded by clients before being divided and delivered to each of the working nodes. The distribution phase takes substantially longer (order of magnitude) to perform on cloud infrastructures than on a dedicated cluster. Similarly, in Globus Genomics gateways large reference genomes (tens of Gigabytes) must be frequently transferred to worker nodes to perform analyses. Although PDACS does not typically enable input datasets to be transferred (due to their extreme size), the use case highlights the implications on service design caused by networks with limited performance.

The use cases identify an opportunity to improve the performance of scientific analyses by minimising network overheads. This problem is especially evident in low performance cloud networks [52], where many scientific applications perform poorly [25], as data-locality techniques cannot be applied extensively [112, 117].

Thus, the first research question is:

RQ1 How can opaque cloud network performance be accurately measured or inferred and used to minimise the need for, and cost of, data movement?

Network tomography provides the ability to capture network characteristics and determine network structure. In Chapter 4 I address this research question

by proposing approaches for using network tomography to understand cloud network structure.

3.4.2 Economic Optimisation

In most cases the operating costs of on-demand scientific services are directly passed on to researchers. Therefore, it is imperative for scientific services to minimise the cost of performing analyses. Both the pCT and Globus Galaxies services have exemplified the requirement to minimise cloud usage costs. Globus Galaxies gateways historically used a naïve provisioning approach, in which a single large instance type in a single availability zone was selected to fulfil all users' workloads. As demand for this instance type grew, so too did the cost of operating the service (as it was incapable of adapting). In fact, this particular instance type's Spot price would frequently exceed its On-demand price, which would significantly increase the cost of using the service. The pCT use case also identified a unique requirement for economic optimisation techniques. When reconstructing large pCT images over clusters of GPU-enabled instances, the service itself would directly affect the Spot market price. This in turn would increase costs and delay instance creation. These two uses cases demonstrate the need for scientific services to optimise instance selection and usage with respect to cost.

Developing a cost-aware model for provisioning instances is particularly challenging as it requires real-time market information, an ability to predict future prices, understanding of application requirements, and effective provisioning techniques. The combination of the cloud's flexibility and the fact that simple provisioning approaches can increase costs [21, 22] underpins the second research question explored in this thesis, **RQ2**.

RQ2 How can cloud computing market models be exploited to minimise the cost of provisioning infrastructure to execute arbitrary scientific workloads?

There are a wide range of possible solutions for minimising costs associated with executing arbitrary user workloads. For example, employing real-time Spot market pricing data enables services to avoid volatile and expensive instance types. In Chapter 5 I propose, evaluate, and compare several cost-aware techniques, such as distributing work over multiple availability zones to derive best-practice approaches and construct a cost-aware provisioning system. Additionally, I explore bidding approaches to improve reliability and ensure instances do not incur excessive costs.

3.4.3 Resource Provisioning

Resource provisioning is a challenge faced by each of the three scientific services. Optimally selecting an instance type for a specific job requires consideration of both the instance capabilities and the job's resource requirements. However, it can be challenging for researchers to accurately determine and define the resource requirements of custom tools. Conversely, it is labour intensive for service providers to manually evaluate each tool that is contributed to a service (e.g., Globus Genomics has over one thousand tools). The need for intelligent resource provisioning is ubiquitous across the three scientific services. PDACS and Globus Galaxies gateways clearly exhibit a need for understanding and encoding resource requirements in order to effectively provision resources. Similarly, Globus Galaxies and pCT services highlight the need for automated resource provisioning and instance configuration.

Provisioning a sufficiently, but not wastefully, powerful instance is crucial to the effective operation of a service. For this to be achieved the provisioning algorithm must accurately estimate the resource utilisation of a given workload. There are currently 48 different instance types offered by AWS [26]. Many of these instance families are optimised for a specific class of applications, such as compute-, memory-, or IO-intensive applications. In turn, scientific services require an automated approach to select resources based on the resource requirements of the workload. Once an appropriate resource is selected, reliably acquiring and configuring the resource is also essential to the operation of scientific services. These challenges have resulted in the definition of the third and fourth research questions investigated in this thesis:

RQ3 How can the resource requirements of custom applications, with varying input data and configurations, be computed to determine the "best" instance types for execution?

RQ4 How can provisioning decisions be automated to consider trade-offs (e.g., cost, time, data movement) and optimally select instances for a given application?

This thesis presents an automated profiling service to solve the challenges associated with manually deriving resource requirements of arbitrary tools. The service can dynamically acquire instances and deploy user-defined workloads to construct fine-grained profiles of their behaviour and resource requirements. In

Chapter 7 SCRIMP is presented as a system capable of using application profiles to automate the provisioning and configuration processes.

3.4.4 Resource Management

Managing and analysing provisioned cloud instances is an essential component of cloud-based scientific services. Large services can concurrently maintain thousands of instances which must be persistently monitored in order to identify state changes (e.g., instance terminations) and utilisation to enable effective autoscaling. The Globus Galaxies and pCT use cases demonstrate a requirement to manage cloud infrastructures after they have been provisioned. This requirement is most obvious in the pCT reconstruction service. If a single instance involved in a pCT reconstruction fails, or is terminated, an entire cluster (often over 100 instances) will become incapable of performing reconstructions. The Globus Galaxies use case also describes a need for auditing capabilities. Globus Galaxies can concurrently operate hundreds, if not thousands, of instances across several gateways which presents a challenge to associating usage with specific users and jobs. Accountability is also a key requirement for PDACS, as users' resource allocations on dedicated infrastructure must be strictly recorded and enforced.

The diverse challenges relating to resource management has prompted the definition of two research questions:

RQ5 How can services improve the reliability of elastic clusters when employing potentially unreliable infrastructure?

RQ6 How can fine-grain accounting information be maintained for purposes of auditing and cost reclamation?

Chapter 7 describes online monitoring techniques that record every interaction with a service. Based on these records individual instance usage can be calculated and used to optimise allocations, while also providing fine-grained accounting information.

3.5 Summary

Creating and operating elastic cloud-based scientific services is fraught with challenges: instances are unreliable, networks and infrastructure are opaque, there

are an enormous number of deployment configurations, and weighing trade-offs to determine the optimal execution environment is non-trivial. This chapter presented three scientific services that span data-intensive scientific domains, namely cosmology, medical imaging, and genomics. Based on experiences developing and operating these services a core set of requirements have been presented. These requirements provide a broad spectrum which have been used to derive the six research questions that motivate the research presented in this thesis.

Chapter 4

Network Health

Many cloud providers have limited network performance which has been shown to hamper data-intensive scientific applications [52, 25]. Exploiting the proximity of data and compute resources is a proven technique used to minimise the effect of low-performance networks [112, 117]. Thus, to examine the first research question, **RQ1**, I propose investigating network tomography as a means to infer network information and use it to exploit data-locality within opaque clouds.

Network tomography is the process of deriving internal network information by sending and monitoring packets as they travel between two hosts. End-to-end probes can be used to infer the condition of a network at a fine-grain level, identifying bottlenecks, Round-Trip-Time (RTT), and packet loss [172]. By measuring the delay incurred by a probe between two end points, congested links that incur long queuing delays can be detected and avoided [173].

The remainder of this chapter presents the exploration of network tomography techniques to better understand opaque cloud networks and infer both the network load and relative proximity of instances. Throughout this chapter I employ a health metaphor to describe the research. *Network health* is synonymous with the network performance of an instance and tomographic probing sequences are referred to as health indicators. Two testbeds, of up to one hundred AWS instances, are presented to refine the tomographic techniques and aggregation methods. The findings of this work are embodied in a usable Network Health Diagnostic System (NHDS). Finally, the NHDS is deployed over a pCT cluster to demonstrate the potential benefit of considering network performance during the deployment of real world scientific workloads.

4.1 Network Health and the Cloud

The health metaphor used in this chapter is derived from existing literature, where a *health metric* is used to compute an overall health score for a service container and guide deployment decisions [174, 175]. In this chapter I apply the health concept to network tomography as a means to describe the network performance exhibited by provisioned cloud instances. Health metrics (the weighted aggregation of tomographic probes) are computed to provide a mechanism to evaluate and compare the network performance of instances.

A set of health indicators, or collections of tomographic probing sequences (e.g., ICMP echo requests with various payload size), can be used to infer the network health between two instances. In this context a health indicator is a tomographic method of measuring the network performance between two instances. Although AWS provides an integrated health service for EC2 instances, its capabilities are limited to identifying instances becoming unresponsive. In this chapter a range of health indicators are employed to thoroughly observe the network and identify performance properties. Collectively, the health indicators use multiple network protocols (ICMP, UDP, and TCP) and customisable attributes, such as varying payload size and probe frequency. This technique of deriving network performance through network probes can be traced back to existing literature [173, 176] and remains a preferred method to monitor the state of a network [177, 178].

To use the information gathered from health indicators, a set of health markers are formulated. A health marker is a binary, lightweight, and easily computable diagnostic, used to detect a significant change in network performance across probing cycles and trigger an alert. This chapter presents the formulation of a health marker for each of the health indicators that are employed. Markers quickly establish the degree to which the network performance between two instances changes over a period of time and prompt the recalculation of the overall health score for an instance when necessary.

A health metric is a normalised aggregation of health indicator measurements. Health metrics combine a set of health indicator measurements into a single value. One health metric is formulated for each of the network protocols employed. The set of health metrics are weighted and combined to compute a single, overall, health score for a target instance. A health score provides a high-level diagnostic of the network performance of an instance with respect to its peers. A health score

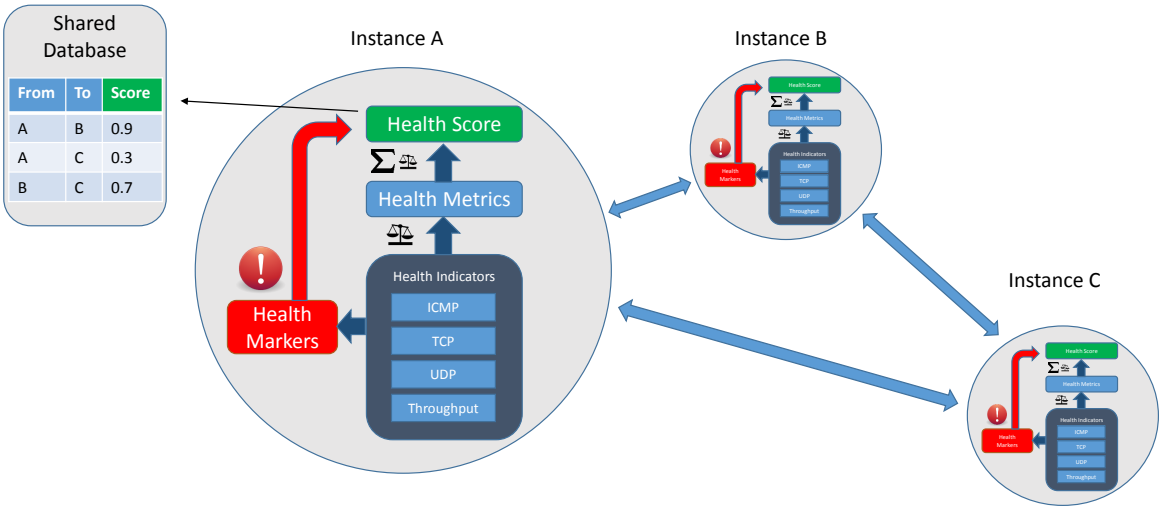


Figure 4.1: An outline of the Network Health Diagnostic System operating over three instances.

is a single value which allows instances to be compared to one another. When computed, a health score gives a perspective of the load the target instance, or the network connecting two instances, is experiencing and can facilitate the selection of healthy clusters.

Collectively, these components create a usable tool, known as the Network Health Diagnostic System (NHDS). Figure 4.1 depicts an outline of the NHDS deployed over a set of cloud instances. The NHDS uses health indicators to collect network performance measurements between provisioned instances. It streams information to the health markers, which can trigger performance deterioration alerts. In addition, an overall health score is periodically computed by aggregating health metric values. An instance's health score can be compared to other instances in order to select clusters with high network performance.

4.2 Testbeds and Cloud Performance Baselines

Two testbeds (*Testbed I* and *Testbed II*) of AWS instances have been created to investigate the properties of commercial cloud networks and develop the components of the NHDS. Testbed I consisted of a small set of six instances and has been used to observe baseline AWS cloud performance characteristics and evaluate various tomographic techniques. Testbed II consisted of one hundred instances to represent a large-scale environment from which additional performance characteristics could be identified. It also allowed the evaluation of the overhead in-

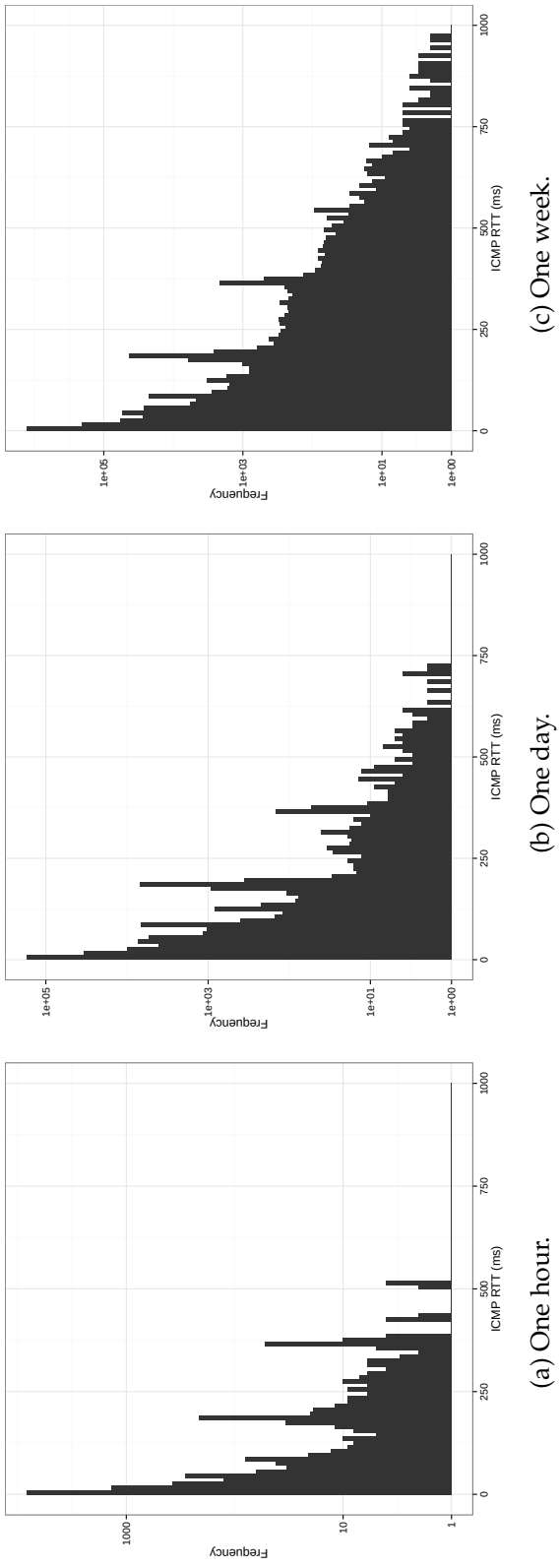
curred by actively performing network measurements over large infrastructures. Monitoring of the testbeds has resulted in the creation of two datasets: *Dataset I* and *Dataset II*. These datasets are analysed to evaluate the effectiveness of the health indications, refine the triggering points of health markers, and determine appropriate weights to aggregate the health indicators into health metrics. These testbeds and analysis of their datasets has guided the development of the NHDS. The following describes the testbeds, data collection methods, and the performance characteristics that have been identified.

4.2.1 Testbed I

Testbed I initially consisted of just three t1.micro and three m3.medium type instances. Testbed I enabled the exploration of various cloud features, such as how recurring events (e.g., time of day) influenced the network health measurements. The testbed's instances were deployed across two availability zones in order to derive the effect of data traversing between zones. The tests were run in a round-robin process from each instance, where every five minutes each instance would probe every other instance in the testbed. The probing schedules were deliberately offset in an effort to reduce the interference caused by multiple instances concurrently performing measurements with a single target instance. Testbed I was later expanded to include 20, and then 50 instances, however the software harness used to collect network information was unable to scale to larger infrastructures. This limitation led to the development of Testbed II, see Section 4.2.2.

ICMP echo requests are a typical method used to measure the latency between instances. The *jitter*, or variance in latency, within a link can be established by collecting RTT probes over a sufficient period of time. Different payload sizes are used to determine how packet size effects network measurements.

Figures 4.2a – 4.2c show the variation in ICMP RTT measurements over different periods of time. These results show the distribution of probes contains a significant number of high RTT values, over each period of time—demonstrating the high degree of network RTT volatility experienced by a regular AWS instance.



(a) One hour. (b) One day. (c) One week.

Figure 4.2: The frequency (log) of ICMP packet RTTs over different periods of time.

Over long measurement intervals, the variation in ICMP RTT occurs over sufficiently long periods of time to have an impact on the performance of an application—or in other words, the performance of a link can deteriorate (or improve) for periods of hours, rather than in short intermittent bursts. One example of this is shown in Figure 4.3, where the average hourly RTT between two instances, for various ICMP packet sizes, over a one day period is depicted. Over this period, the 4096 and 512 byte ICMP packets have higher RTTs at the beginning of the day and then gradually improve. The performance of the 64 byte ICMP packets is reasonably consistent over the same period. While having different shapes, similar trends are observable across both over other periods and between other instance pairs. The fact that these variations occur for meaningful lengths of time supports the theory that monitoring network performance can be used to improve scheduling and deployment decisions.

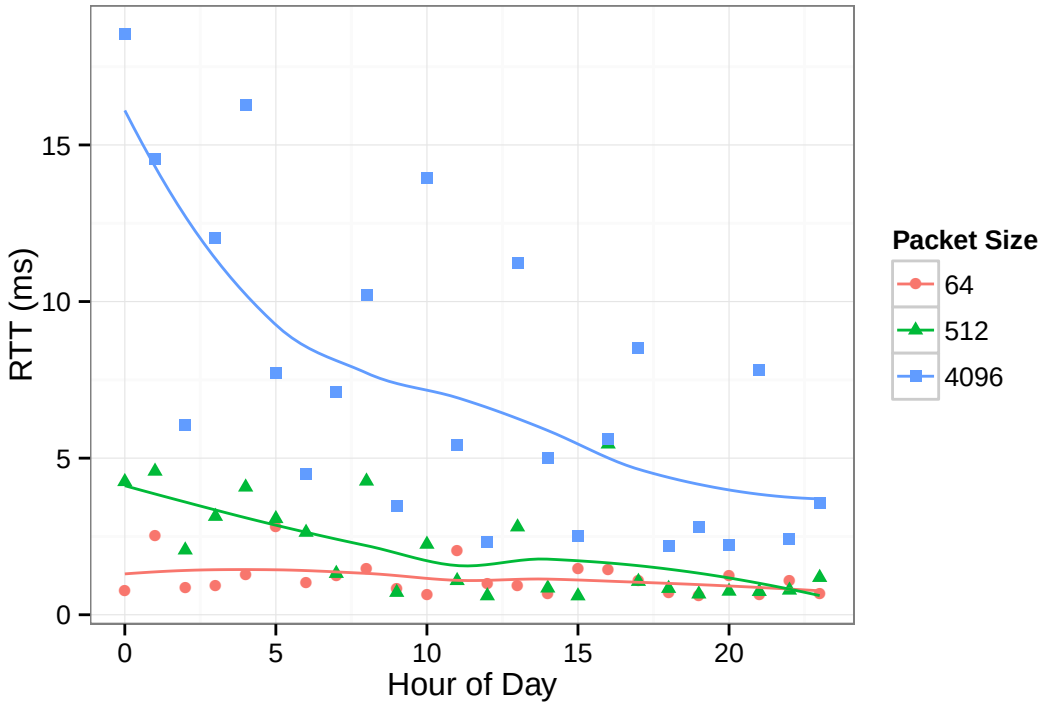


Figure 4.3: The hourly average RTT for different packet sizes between two instances.

A series of bandwidth measurements have been conducted in order to further understand the network performance between various instance types and across availability zones. The measurements involved transferring as much data as pos-

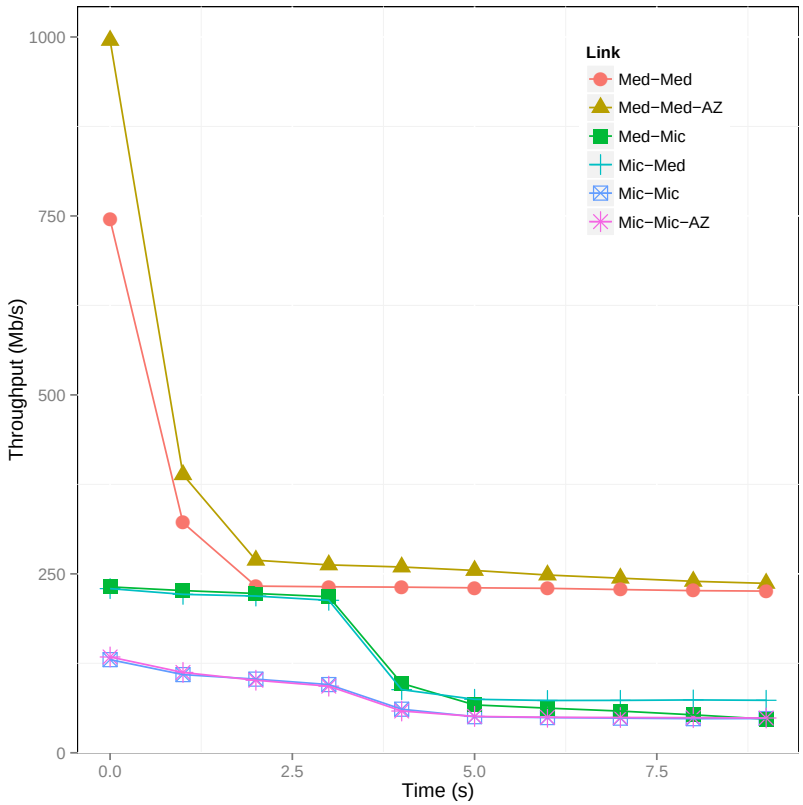


Figure 4.4: The average throughput between medium and micro instances within and across an availability zone.

sible between pairs of instances over a ten second period. The results found significant bursting characteristics for TCP transfers. Figure 4.4 shows the average throughput over the ten second time span between various instance types, where links across availability zones are denoted by *-AZ*. The figure depicts the presence of substantial throttling and probable profiling of data transfer within the AWS network. The throttling differs between instance types, where t1.micro instances achieve a relatively high throughput of approximately 200Mb/s for the first four seconds of a transfer before being throttled to approximately 80Mb/s. Similarly, the m3.medium instances initially achieved a throughput of almost 1000Mb/s for approximately one second before being throttled to slightly over 200Mb/s. From this data it appears that the instances are granted a burst throughput rate for the first 1000Mb of data being transferred. Additional tests with cluster compute, cg1.4xlarge, instances demonstrated a sustained throughput of almost 8000Mb/s within an availability zone, and a sustained 2000Mb/s connection across zones. The cluster compute instance types did not appear to be subjected to the same

throttling techniques and were most likely achieving their maximum available throughput.

These findings demonstrate two interesting properties that support the goal of using network information during the deployment of scientific applications. Firstly, network performance exhibited by instances is significantly volatile and is subject to changes in excess of 50% over a short period of time. The performance fluctuations also persist for sufficiently long periods of time for a scheduler to take action. If the network variance was only observable for short periods of time (for example, on the order of seconds), the volatility of the network would render any deployment optimisations ineffective as the network performance could change many times during execution. However, these results indicate that this is not the case. In a number of examples the degraded performance of an instance is consistently observed from multiple other instances for periods of hours.

4.2.2 Testbed II

A second testbed, referred to as *Testbed II*, was developed to examine the scalability and overhead of conducting network performance tests over large cloud infrastructures while simultaneously generating a rich dataset of network information to refine the NHDS. Testbed II consisted of one hundred m3.medium instances distributed across three availability zones in the US-East-1 region. Due to gradually scaling the testbed size, more instances were acquired in the first availability zone than the others, with 35 instances on us-east-1a, 33 instances on us-east-1b, and 32 instances on us-east-1c.

A simple provisioning system was constructed to improve the reliability of launching and experimenting with large cloud infrastructures. Figure 4.5 represents an overview of the provisioning system used to acquire and configure instances. The provisioning system creates spot requests for the desired instance type and attempts to evenly distribute them across the specified availability zones. AWS presents two methods for establishing provisioned instances: through the use of a predefined Amazon Machine Image (AMI), or by dynamically contextualising the instance. In this case, the provisioning system dynamically contextualises each instance using cloud-init [179] as it provides additional flexibility in the deployment of the health system in subsequent tests. During contextualisation the health monitoring software is downloaded to the instance and deployed. Once the health system begins operating it contacts a shared Relational Database

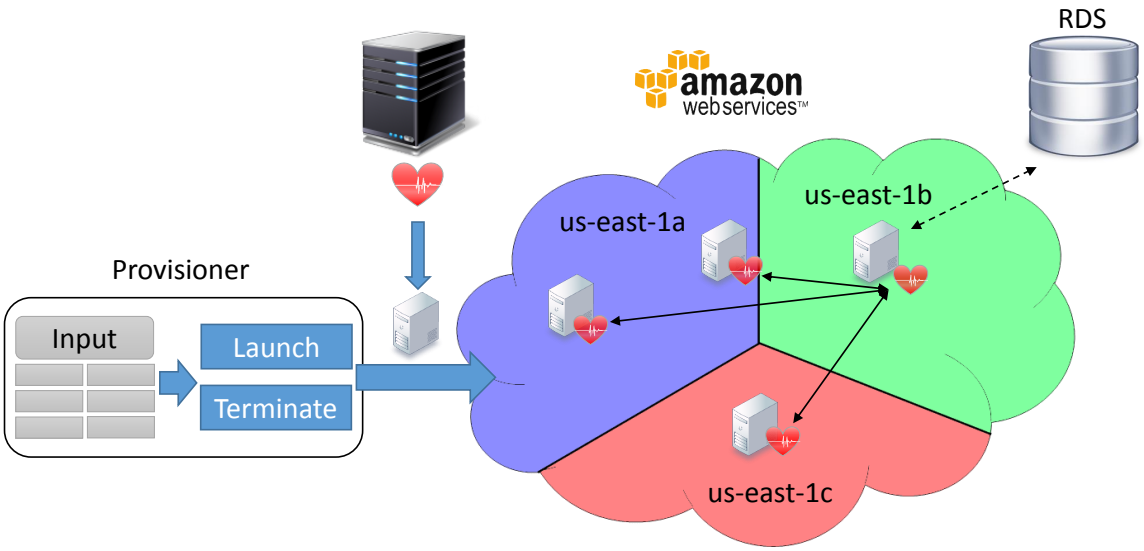


Figure 4.5: The provisioning system used to construct testbeds and evaluate network performance.

Service (RDS) instance to publish its address for others to probe. The system then begins periodically probing other instances in the testbed. The collected results are then reported to the shared RDS for analysis.

It takes approximately twelve seconds to perform all of the health indicator measurements on a target instance. The majority of this time is spent collecting the throughput measurements which are taken at one second intervals over a ten second period. This means to probe each of the 99 other instances in Testbed II requires almost 20 minutes. To reduce the risk of collisions, which could potentially distort results, the rate at which instances are probed has been reduced in Testbed II. To further minimise collisions, the shared database is used to determine the order in which probes are conducted. The health system deployed in Testbed II probes peers once hourly in a round-robin fashion, beginning with instances that have joined the testbed since itself (as recorded in the database), before iterating through the remaining instances.

4.3 Network Health Diagnostic System

Analysis of the two testbeds informed the creation of a Network Health Diagnostic System (NHDS). The testbeds have guided the selection of health indicators to be used in the NHDS as well as determined the trigger points for health mark-

ers and the aggregation weights used to compute a health score. The NHDS is capable of capturing network performance measurements between instances in a large cloud infrastructure. The remainder of this section discusses the components of the NHDS.

4.3.1 Health Indicators

The performance of an instance can vary over time due to the network load both itself and surrounding resources are experiencing. A set of health indicators have been selected and evaluated with respect to their ability to reliably observe performance fluctuations and influence the health score of a target instance. Health indicators are capable of monitoring fine-grained latency and throughput variations as well as capturing timeout occurrences.

The delay-based tomographic indicators utilise ICMP, UDP, and TCP protocols, with the goal of establishing load by observing variations in RTT and measuring jitter in the network. A range of packet sizes and varying intervals between sending packets have been used to identify the effect of queuing in the network.

Spot instances, which are often used to reduce costs, are characteristically unreliable. Involuntary termination results in an instance becoming unresponsive to those that are probing it. For this reason, timeouts have been incorporated as indicators as it is critical to identify unresponsive instances.

Throughput indicators and Sandwich probing, first presented by Coates et al. [122], have also been employed. Sandwich probing measures the delay incurred by a small packet traversing a network when preceded by a large packet. This is accomplished by sending two small packets separated by a time d with a larger packet immediately preceding the second packet. By measuring the time between the arrival of the two packets, d', the difference between d and d' can be obtained to infer the delay caused by the large packet.

Consideration of Sandwich probing packet sizes is required when implementing the probing scheme. The maximum transmission unit (MTU) for t1.micro and m3.medium instances is 1500 bytes. However, AWS supports jumbo frames for cluster compute instances types, allowing packet sizes of up to 9001 bytes to be used when probing cg1.4xlarge instances. I adapt the Sandwich probing large packet size to reflect the MTU, while maintaining a small packet size of 64 bytes regardless of instance type.

4.3.2 Health Markers

Health markers have been defined to provide lightweight diagnostic values and alert the NHDS of degrading network performance. A health marker is a binary indicator (trigger/no-trigger) of the presence of a problematic state in the network (analogous to tumor markers in medicine). Five markers have been selected and implemented in order to alert the NHDS of significant variations in network performance. These markers each represent a quantifiable threshold value related to a specific goal gathered from a protocol between two instances. The following describes the implementation details of each health marker. The decisions regarding the point at which a marker is trigged, and an alert is raised, have been established through experimental analysis in order to identify targets which are only triggered when a significant degree of volatility or degradation is detected in the network.

- **Timeout** A timeout health marker is used to raise a notification when an instance becomes unresponsive. This health marker requires consensus from more than one indicator of unresponsiveness, or lack of response within five seconds, of another instance.

- **ICMP** An ICMP health marker combines each of the three packet sized RTT measurements and compares them with the standard deviation from the previous round of probes. The marker is triggered when 20% of the current round's measurements exceed the standard deviation of the previous round.

- **UDP** The UDP health marker employs UDP RTT and Sandwich probing. The RTT is gathered from 1024 and 64 byte probes and the standard deviation from the previous round is used to infer degradation. Sandwich probing measurements are used to trigger a notification when a 50% increase in delay is observed, implying the effect of queuing in the network is significant.

- **TCP** The TCP health marker monitors the time required to establish a TCP connection between two instances as well as the RTT of the connection. The standard deviation from the previous round is used to establish threshold times, which when exceeded triggers the marker to raise a diagnostic notification.

- **Throughput** The throughout health marker measures the available throughput between two instances over a ten second period. The marker is triggered when the total throughput over the ten second period drops below a longer term threshold.

The next section introduces the concept of health metrics, which are concrete measures used to inform selection of instances for improved performance. Health markers are not used in computing the health metrics but both utilise the same information (health indicators) collected by the tomographic probes, as shown in Figure 4.1.

4.4 Health Metrics

Health metrics provide a normalised mechanism to evaluate instances against one another. A health metric has been formulated for each individual network protocol utilised by the indicators and typically aggregates the information collected from multiple indicators into a single value. Each health metric computed for an instance is normalised with the other instances that are being monitored, providing a relative health for each network protocol.

An overall health score is computed by combining each of the individual health metrics through weighted aggregation. The weights associated with each health metric have been selected through a statistical evaluation in order to give each marker meaningful influence on the overall health score. The overall health score, $H-All$, of an instance gives the host a mechanism to directly compare each instance in the environment and select healthy nodes to perform workloads.

The ICMP health metric (denoted by $H-ICMP_{ij}$) prioritises packet size from largest to smallest, with heavier weights given to the larger payload measurements. The health metric computes the ICMP score by averaging the RTT of each packet size, and normalises it against the average RTT of its respective size for each instance the host has probed during a round. Eq 4.1 shows the calculation of the ICMP health score where $S = \{64, 512, 4096\}$ is the set of packet sizes being used as probes and P_{ij_s} represents a set of probes from host i to j of size s where $s \in S$. A_{i_s} denotes the set of the average ICMP probe measurements sent by host i, of size s, to every other host in the environment. Finally a weight (denoted by ω_s) is associated with each packet size, such that larger packets are given more influential than smaller packets.

$$H - ICMP_{ij} = \sum_{s \in S} \frac{\text{avg}(P_{ij_s}) - \min(A_{i_s})}{\max(A_{i_s}) - \min(A_{i_s})} \times \omega_s \tag{4.1}$$

The UDP health metric (denoted by $H - UDP_{ij}$) normalises the average RTT values and the average delay measured from Sandwich probing to evaluate the link between instances i and j. The two RTT values and the Sandwich probing delay are measured by the UDP health indicator and are combined with weights giving more influence to larger packets. The Sandwich delay is given an equal weighting to the RTT measurements to give influence to the delaying properties of the network.

The TCP health metric (denoted by $H - TCP_{ij}$) is computed in a similar fashion and normalises the average connection time and RTT through the connection during a measurement period. Each value is then combined with equal weighting to provide a relative health score of the connection ij.

The throughput health metric (denoted by $H - TP_{ij}$) incorporates the total amount of data transferred over the ten second measurement period with the variance in throughput during each one second interval. These values are individually normalised and then combined with equal weighting to construct the throughput health score.

Two datasets, *Dataset I* and *Dataset II*, which have been collected from Testbed I and Testbed II, respectively, are used to analyse the role and influence of each health metric. An associated weight is required for each health metric in order to aggregate the metrics to compute an overall health score. For each health metric the variable selection technique was applied to a linear regression model over the complete dataset and the metrics with the strongest influence on the aggregate health score were ranked [180, 181]. Based on trace data collected from AWS from 29 April 2014 (10:45:16) to 6 May 2014 (14:24:27) forward step analysis was applied on Eq. (4.2) to rank the influence of the individual health metrics on the overall health score. Starting from Step 4 (Column Dataset I) in Table 4.1, One health metric is added per step and the corresponding Akaike Information Criteria (AIC) measure and p-values are calculated.

$$\begin{aligned} &H - ALL_{ij} \sim \\ &H - TP_{ij} + H - ICMP_{ij} + H - UDP_{ij} + H - TCP_{ij} \end{aligned} \tag{4.2}$$

Lower values of the AIC signify stronger influence of the health metric in the

regression while the p-values indicate confidence in health metric influence on the health score. In Table 4.1 for example, using the sole metric $H - TCP_{ij}$, the step analysis at Step 1 yields an AIC of 106.842 and corresponding p-value of 0.00609, adding the $H-UDP_{ij}$ metric reduces the AIC to 90.221 but increases the resulting p-value to 0.00752. The reduced confidence in the regression at Step 2 is expected due to collinearities in metrics $H - TCP_{ij}$ and $H - UDP_{ij}$. The predictive power of the throughput metric is captured by both $H - TCP_{ij}$ and $H - UDP_{ij}$ thus weakening the $H - TCP_{ij}$ metric in the forward step analysis. Upon terminating the forward step analysis, all four health metrics are selected because the p-value is greater than 0.05, which indicates that the health indicators are significant and hence all four health metrics influence the network performance prediction.

Another set of trace data, referred to as Dataset II, was collected from AWS from 28 January 2015 (16:59:15) to 28 January 2015 (22:09:31). The forward variable selection analysis on this newly collected data is presented in Table 4.1. The observed AIC trend is consistent with the results from Dataset I and therefore reinforces the earlier findings on the merits of the four selected health metrics on predicting the health score. For both datasets, the reverse step (elimination) analysis yields final AIC values identical to the forward analysis.

Table 4.1: Forward step analysis of health metrics for two sets of trace data.

Step	Health metric	Dataset I April 2014		Dataset II January 2015	
		AIC	p-value	AIC	p-value
4	$H - TP_{ij}$	216.311	0.04126	106.686	0.02454
3	$H - ICMP_{ij}$	98.227	0.00587	99.114	0.01399
2	$H - UDP_{ij}$	90.221	0.00752	103.098	0.00605
1	$H - TCP_{ij}$	106.842	0.00609	118.212	0.00465

The adjusted–R^2 (adjr2) measure for different subsets of network health metrics are shown in Figure 4.6. The adjr2 compares the errors in the regression that is adjusted to the different numbers of health metrics. For example, in the plot of Figure 4.6, using a single health metric $H - TP_{ij}$ yields an adjr2 value of -0.034. This value is calculated by taking into account the fact that only a single health metric is used. For the case of using three health metrics ($H - TP_{ij}$, $H - UDP_{ij}$ and $H - TCP_{ij}$) a smaller adjr2 value of -0.0099 is obtained, this value is a fair

comparison with -0.034 because it has been adjusted for three health metrics. The minimum value of the adjusted adjr2 is -0.0071 when all four health metrics are used and this is marked with red bars denoting the lowest error range. The maximum value of adjr2 is -0.050 and it occurs when a single metric is used ($H - TP_{ij}$). This is marked with beige bars (highest error range) in Figure 4.6. These observations from the heat map suggests that all the selected health metrics have merit in predicting network health, some more than others, and this conclusion agrees with the conclusion drawn from the forward step analysis.

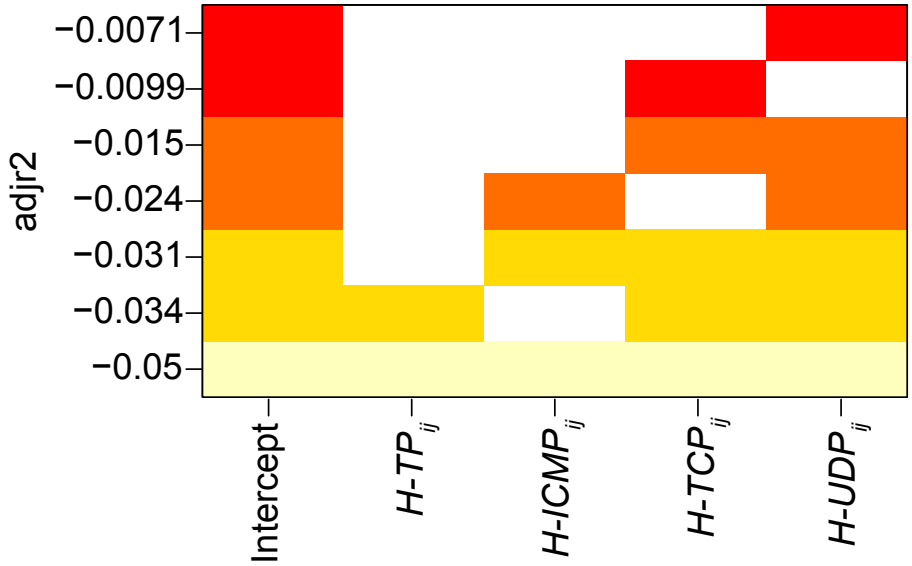


Figure 4.6: The heat map depicting the merit of the health metrics. The red bars indicate the lowest error range (higher merit) while beige bars indicate the highest error range (lower merit).

4.4.1 Health Metric Diagnostics

The health metric selection diagnostics are used after performing variable selection to check if the linear regression and its assumptions are consistent with the observed data. The basic indicator for the diagnostic is the residual. A residual or fitting error, is an observable estimate of the statistical error. If the linear regression does not give a set of residuals that appear to be reasonable, then the choice of the health metrics (one or more) may be called into doubt. Visual inspection is used to analyse the merit of the health metric used to predict network health.

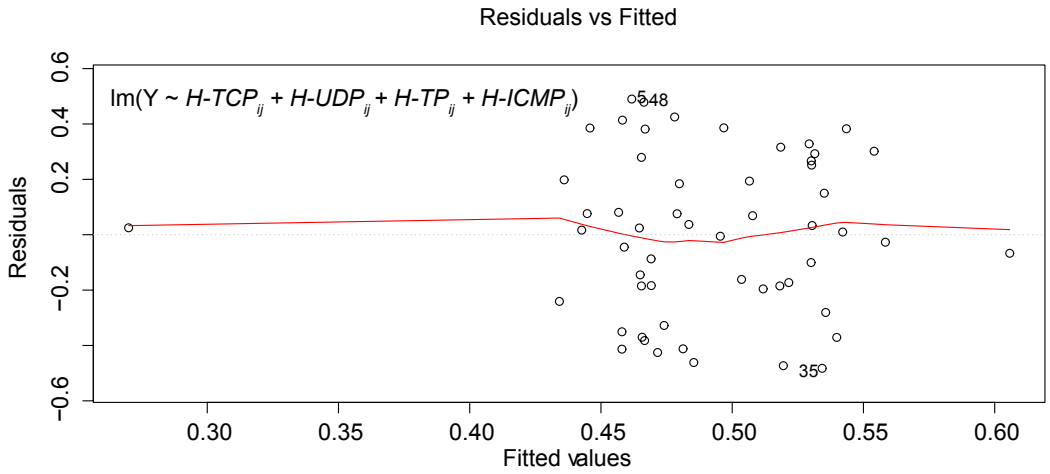


Figure 4.7: The residuals for linear regression.

The evenly distributed residuals (with respect to Residuals=0) in the plot of Figure 4.7 shows that the residuals (errors) and the fitted values of the health metrics are uncorrelated. This validates the choice of health metrics as appropriate measures for predicting network performance.

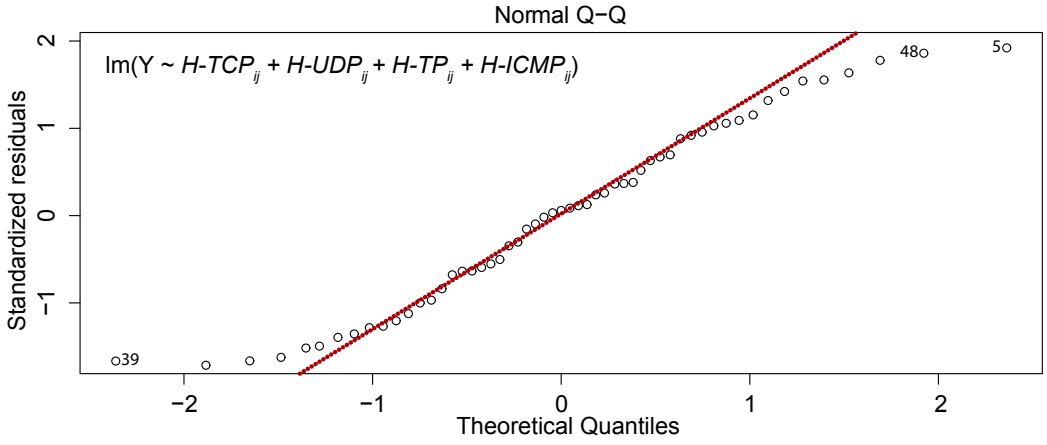


Figure 4.8: The Q–Q plot of the standardised residuals from the linear regression (y-axis) vs. theoretical (Normal) quantiles (x-axis).

The quantile–quantile or Q–Q plot is a graphical diagnostic used to check the validity of a distributional assumption [182] for the linear regression model used in Eq. 4.2. In the linear regression, the residuals are assumed to have a normal distribution and thus the Q–Q plot for the standardised residuals will be close to a straight if this assumption is valid. Figure 4.8 shows that the distribution of the model residuals are balanced with respect to both the upper and lower quantiles.

Moreover, the curve tracks the straight line quite well for theoretical quantiles between -1.5 and $+1.5$ which is what is expected of normally distributed residuals.

This shows the relative significance of each health metric and their respective statistical interpretations, however, this analysis must be framed within a networking perspective. The following section introduces an aggregate measure called the health score, which summarises the four health metrics, and discusses the effect of each health metric on the health score.

4.4.2 Health Score

The timeout of an instance is critical to identify and nullifies additional health metrics. Therefore, the overall health metric incorporates timeout values by computing a health score of zero, or $H - ALL_{ij} = 0$. However, if no timeouts are identified, and an instance is considered operational, the overall health metric is computed as seen in Eq 4.3. From the variable selection analysis, the marker $H - TP_{ij}$ is weaker than the remaining markers. Moreover, throughput is one of the key metrics in service level agreements in AWS specifications. Thus, the weight of 0.4 is chosen for marker $H - TP_{ij}$ to prioritise throughput over latency. Each of the individual metrics are normalised against the other connections in the system and aggregated together with weights.

$$\begin{aligned} H - ALL_{ij} = 0.4 \times H - TP_{ij} \; + \\ 0.2 \times H - ICMP_{ij} \; + \\ 0.2 \times H - UDP_{ij} \; + \\ 0.2 \times H - TCP_{ij} \end{aligned} \tag{4.3}$$

The health score has been computed between each instance in Testbed II and is represented as a heat map in Figure 4.9. The instances are ordered by availability zone, with the first 35 residing in us-east-1a, 33 in us-east-1b, and 32 in us-east-1c. While the heat map demonstrates the similarities between availability zones, it clearly depicts the boundaries of the us-east-1b and us-east-1c availability zones, showing the deprecation in network performance across them. The heat map also clearly demonstrates the volatility in network performance that can be experienced within a single availability zone.

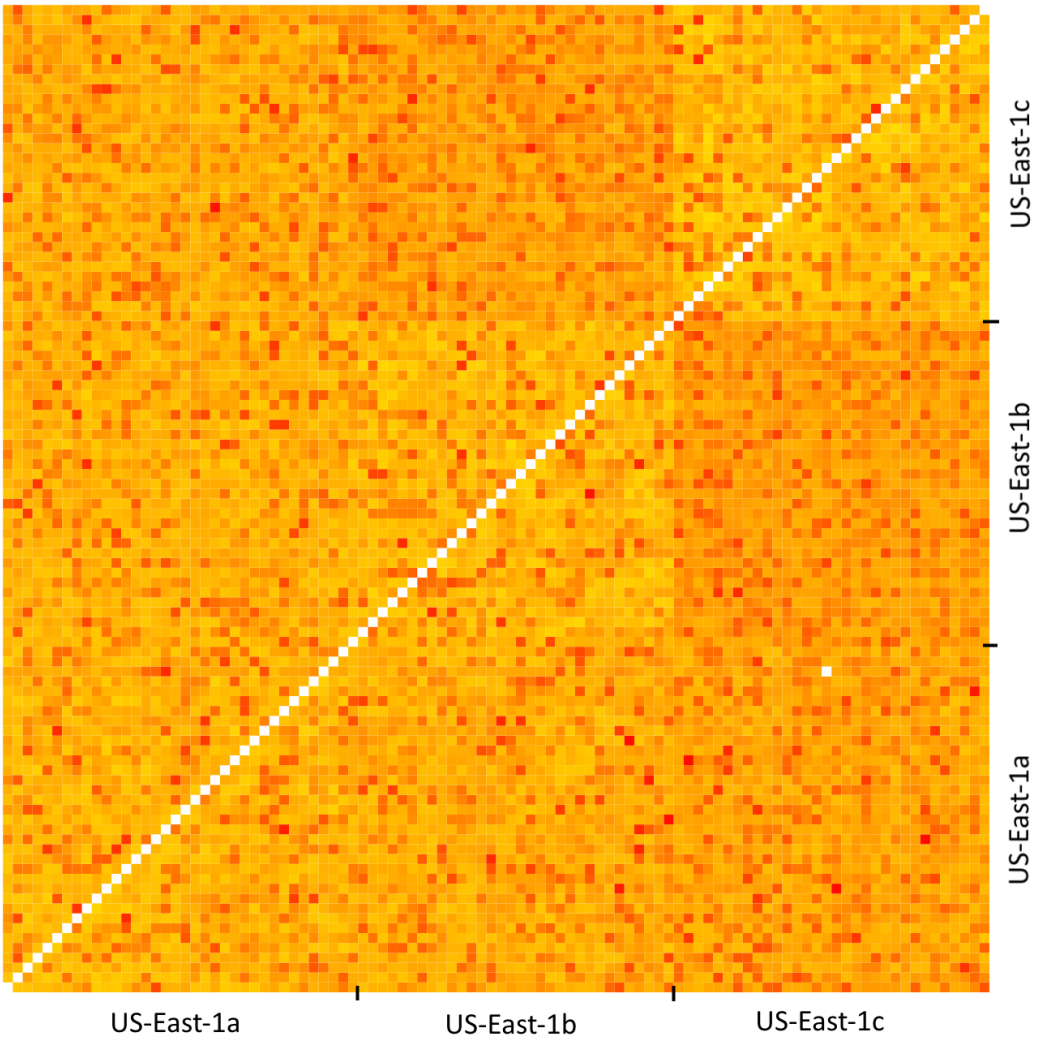


Figure 4.9: A heat map of the health scores computed between the one hundred instances monitored in Dataset II. Each square represents the health score computed between two instances, where red indicates a lower, or less healthy score, and lighter values depict healthier connections.

The collection and computation of health scores between every pair of instances in the environment requires a significant amount of time. When employing the health system for a scientific application, the profile of the application should be considered. For example, the pCT application utilises a centralised file system to distribute workloads, which is pivotal to the performance of the data distribution phase of a reconstruction. Therefore instances can focus their monitoring efforts on the connection between themselves and the shared file system, rather than monitoring all instances in the environment which could negatively influence the execution of the application.

4.5 Proton Computed Tomography

The pCT application, described in Section 3.1, is used here to evaluate the effectiveness of the NHDS. pCT leverages a shared Gluster [183] file system to distribute the input data to each working process. The data distribution phase of pCT reconstructions was found to take substantially longer (approximately an order of magnitude) on the cloud when compared to a dedicated HPC infrastructure. For small 131 million history reconstruction over 20 instances, the data distribution phase accounted for 25.8% of the total execution time. Moreover, when deployed over 120 instances, the data distribution phase accounted for 38% of the total execution time when reconstructing a two billion history image.

The NHDS has been deployed over a pCT reconstruction cluster to explore the potential of network inference and health metrics to improve execution performance. The network-aware pCT reconstruction experiment was deployed over fifteen GPU enhanced cluster compute instances, known as the cg1.4xlarge instance type. The instances were provisioned from two separate availability zones within the US-East-1 region. The pCT codes were used to reconstruct a 131 million proton history image over eight instances, utilising two processes per instance to match each available GPU.

Each test requires eight instances to be selected to perform a reconstruction. Three groups of clusters are used to evaluate the affect of using the NHDS: those with the highest health score, lowest health score, or random. Health scores have been calculated between each instance and the shared file system immediately prior to selection. Figure 4.10 depicts the inferred distance of each instance from the shared file system, using health scores to weight edge lengths. The topological

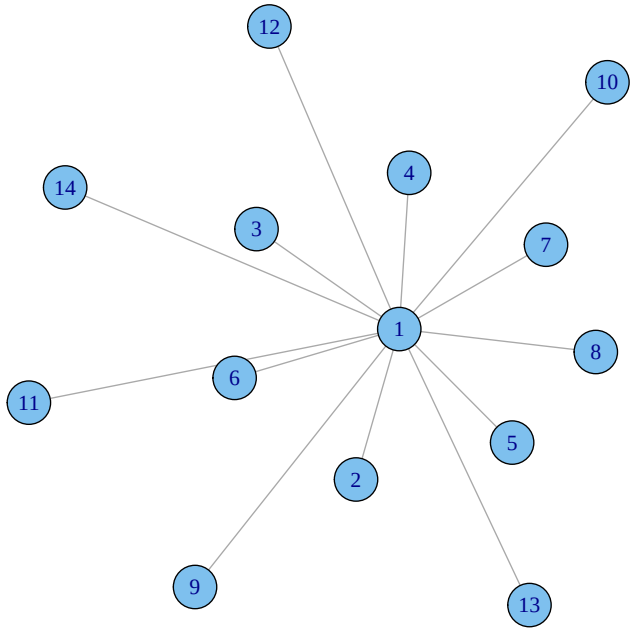


Figure 4.10: The proximity of instances used for pCT reconstruction where edge length is determined by health score.

distribution of each instance is not considered in this figure.

The average result of multiple pCT reconstructions over each group has been computed and is presented in Figure 4.11. The figure demonstrates a distinct advantage to leveraging the proximity of instances when deploying the application. Improvements in performance can be seen during the data distribution phase and during execution of the linear solver. The inclusion of health information resulted in the data distribution phase taking, on average, less than half as long as that of the least healthy group of instances. Similarly, the cuts and margins phase has been reduced as it requires data to be communicated between the instances, whereas the time required to compute a proton's most-likely-path (MLP) is consistent between clusters as little data is transferred. Due to the small number of instances being used to reconstruct the 131 million history dataset, the computationally intensive linear solver accounts for the majority of the reconstruction time. Over eight instances the linear solver accounts for between 80% and 85% of the reconstruction time. Whereas when performing larger reconstructions over 120 node clusters the linear solver accounted for less than 55% of the execution.

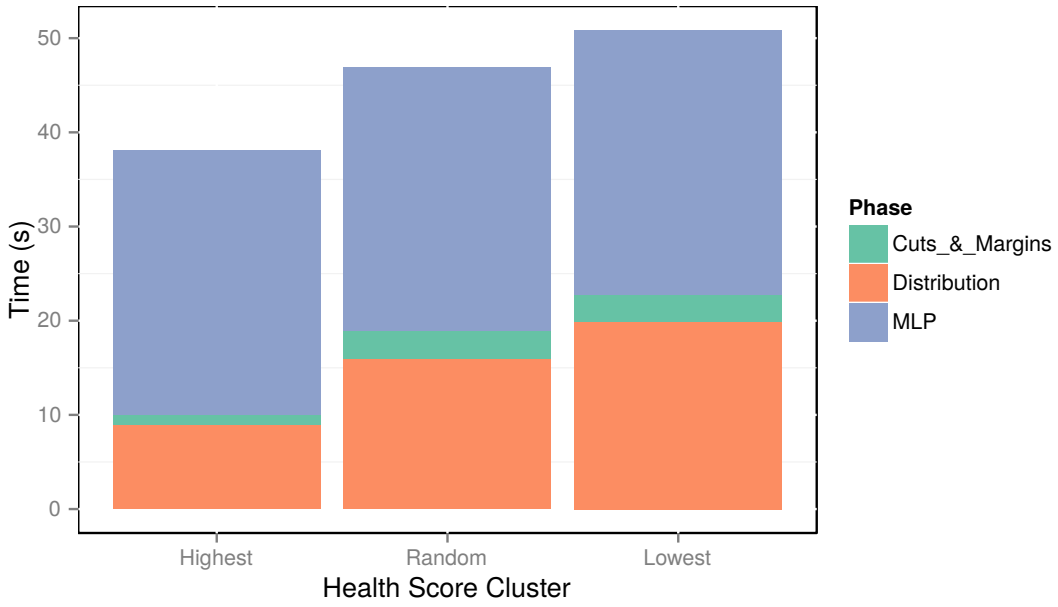


Figure 4.11: The average time required for phases of pCT reconstruction by various health score clusters.

4.6 Discussion

The network information captured in Testbed I and Testbed II has shown that the selected health indicators are capable of identifying network fluctuations. The volatile nature of the cloud network resulted in substantial variations across all of the health indicator measurements. Capturing the network performance between Testbed II's one hundred instances highlighted the volatility in cloud network performance. The testbed showed unexpectedly high levels of variation within availability zones.

An evaluation of the variance observed by the health indicator measurements has been used to rank the influence of health metrics and contributed to the weights used when computing the overall health score. Counterintuitively, the throughput health metric was identified as less influential than other metrics in the forward step regression analysis. To reflect data-intensive scientific applications' reliance on efficient network transfers, the throughput indicator was assigned a larger weight than other indicators when computing the overall health score.

Diagnostic health markers have been established from these variations in order to prompt reassessment of the overall network health. Due to the noisy na-

ture of the regular instance testbed, an initial set of threshold-based markers frequently responded to network fluctuations. However, the cluster compute 10-Gigabit Ethernet connection is far less volatile and resulted in fewer notifications being raised by the health markers. In order to operate more effectively, the health markers need to adapt to the environment in which they are executing. Lower tolerances and the inclusion of more historic data is needed to fine tune the markers over various platforms.

Health metrics provide an effective method to compare and evaluate instances against one another. The metrics operate successfully in all of the monitored environments, and have demonstrated the ability to improve the data distribution performance of pCT reconstructions. Although the difference in cluster compute instance health scores is most apparent between availability zones, the health metrics were accurate enough to consistently identify low performance cluster compute instances within a single zone as well.

The deployment and evaluation of Testbed II has also identified limitations of the health diagnostic system to scaling. The risk of interference and the time required to establish health scores for each instance grew sharply as more instances joined the testbed. This restricts the ability for the NHDS to identify network variations in a timely manner, and also limits the viability of applying it to larger scientific applications.

The throughput indicators account for the majority of the time required to establish network health and are most likely to negatively impact the performance of an instance. However, the throughput metrics have been shown to have little significance on the overall health score computed for an instance, meaning they can be reduced in length, or eliminated from the health score calculation entirely.

4.7 Summary

The research presented in this chapter has been conducted to investigate the first research question, **RQ1**. This question asks how opaque cloud networks can be measured to minimise the effect of low-performance networks. This chapter explored network tomography as a means of revealing cloud network information. A number of properties of commercial clouds have been identified, such as the variability in network performance, and the sustained nature of performance fluctuations that an instance can experience. Two testbeds, comprised of up to

one hundred AWS instances, have been used to investigate and analyse the capabilities of tomographic probes.

A Network Health Diagnostic System (NHDS) has been presented. The NHDS is capable of being deployed over large cloud infrastructures to monitor and establish the network health between instances. Health markers have been defined to trigger alerts when significant changes in network performance are detected and health metrics have been formulated to compute comparable health scores, indicative of an instance's current network performance.

Overall, this chapter addresses **RQ1** by demonstrating the ability for the NHDS to improve the execution performance of a scientific application by enabling network-aware deployment. The system is generalisable and can be employed over almost all networked resources to capture and evaluate network performance.

Chapter 5

Cost-aware Resource Provisioning

The lure of cost-effective and elastic computing capacity has resulted in a move towards hosting scientific applications on commercial clouds [56]. However, using commercial clouds does not necessarily result in cost-effective and efficient computing. Naïve resource provisioning can often result in inefficient and expensive execution [21, 22]. Analysis of the use cases has identified a requirement for economical cloud usage and resulted in the formation of the second research question, **RQ2**, which asks how cloud markets can be leveraged to minimise costs. To address this research question I explore techniques to minimise the monetary cost of provisioning cloud resources.

This chapter presents an investigation into cost-aware provisioning techniques. An analysis of production Globus Genomics gateways is first presented to identify usage characteristics. I then present a cost-aware provisioning system. The system employs real-time analysis of Spot market prices and an expanded scope of instance types and availability zones. Four provisioning *scopes* are explored and it is shown that a broad search scope can substantially reduce monetary costs while also improving throughput. In addition, the approaches can lower instance termination rates, thus increasing the reliability and availability of cloud infrastructures. The provisioning system incorporates numerous cost-aware provisioning techniques and can be used to monitor HTCondor [82] queues to select instances to fulfil jobs. Leveraging the Globus Galaxies use case, multiple production Globus Genomics gateways are employed to evaluate the effectiveness of the cost-aware techniques. The cost-aware provisioning system is later incorporated into the SCRIMP service to cost-effectively select instances to perform workloads.

5.1 Globus Genomics Platform Usage

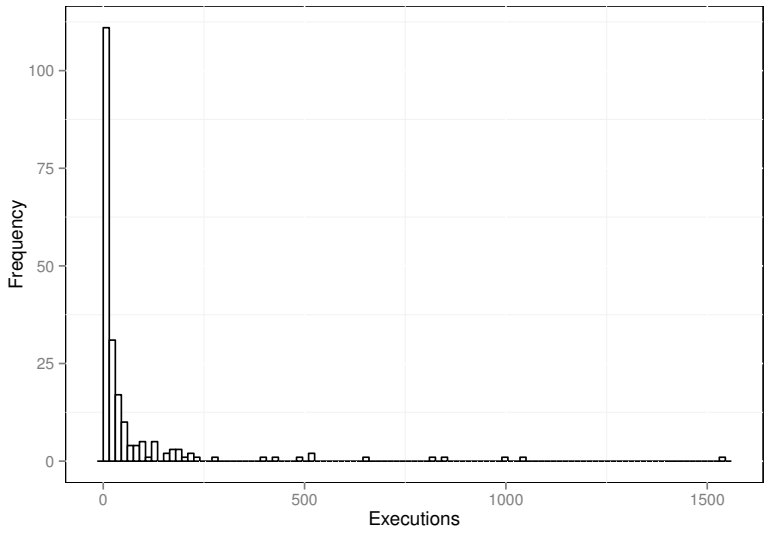
Globus Genomics [36] gateways are individual instantiations of the Globus Galaxies platform [28]. As described in Chapter 3, each gateway executes user defined workflows composed of various tools. Upon execution, workflows are decomposed into a series of computational jobs, each with varying resource requirements (e.g., compute, storage, memory). The cost-aware provisioning system aims to dynamically acquire cloud resources on which these jobs are executed while minimising cost and/or execution time. The approach builds upon rudimentary tool profiles that describe the requirements of individual workflow jobs, leverages different cloud computing acquisition and pricing models (e.g., Spot and On-demand instances), considers real-time pricing information across instance types and availability zones, repurposes Spot requests to satisfy waiting jobs, and can use On-demand instances when Spot requests are not fulfilled within a customisable period of time.

To reach a better understanding of the properties of workflows and their tools (prior to building the provisioning system), I have analysed usage data from six production Globus Genomics gateways. Globus Genomics gateways are used because they are the longest running applications built upon the Globus Galaxies platform and therefore the richest source of usage data. During analysis, the Galaxy logs and HTCondor execution traces are integrated to find tool execution frequencies and execution times. Data has been collected over 308 days on more than 14,500 completed jobs and 25,500 hours of execution.

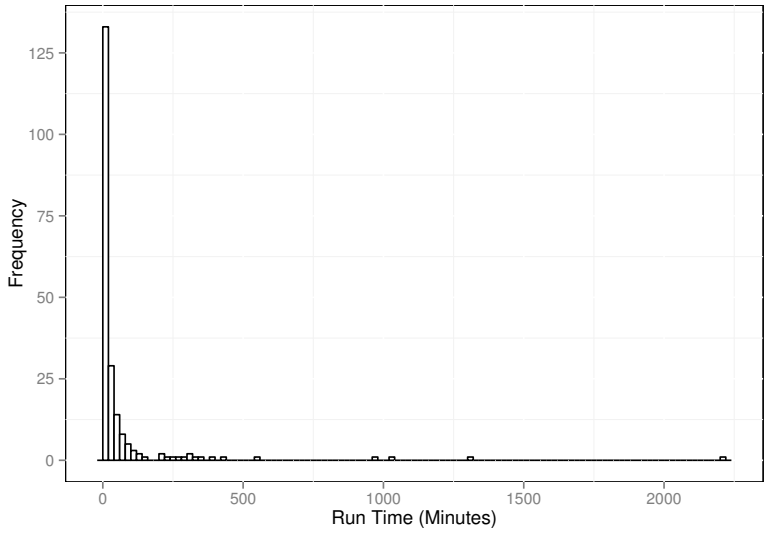
5.1.1 Tool Usage

Collectively, Globus Genomics gateways provide their users with access to over one thousand custom tools. In many cases, several tools are provided that perform the same (or a comparable) task. Each has advantages and disadvantages in different settings and are selected based on user preferences when constructing workflows. The frequency of number of tool executions and tool execution time are shown in Figure 5.1a and Figure 5.1b, respectively. These figures illustrate the long tail nature of tool execution in Globus Genomics. A small number of tools are executed many times while many are executed only a small number of times. Of the 212 tools analysed, the 10 most frequently executed account for 52.3% of all executions, while the 20 most frequently executed account for 68.1%. Similarly,

many tools execute for short periods of time, with 75 tools executing for less than one minute on average, while a small number execute for long periods of time – over 2200 minutes (35 hours) on average.



(a) Tool executions.



(b) Job execution time.

Figure 5.1: Tool usage frequency distribution.

Table 5.1: The execution requirements of production Globus Genomics tools.

Tool	Execs	Avg Run Time	vCPUs	Memory	Disk	Instance
FastQ Parallel	1047	5:48:25	32	High	High	r3.4xlarge
BWA MEM	994	4:34:49	32	High	High	r3.4xlarge
FastQC	820	0:34:53	4	Medium	Medium	c3.2xlarge
MarkDups	659	0:43:53	32	High	Medium	r3.4xlarge
ARRG	512	0:27:13	32	Medium	Low	r3.4xlarge
Sickle	511	1:56:27	4	Medium	High	m2.4xlarge
Flagstat	489	0:21:20	4	Low	Low	c3.2xlarge
SortSam	421	0:15:26	8	Medium	Low	m2.4xlarge
BuildBamIndex	394	0:23:12	4	Low	Low	c3.2xlarge
Bowtie2	273	0:15:11	32	Medium	Low	r3.4xlarge

5.1.2 Tool Requirements

Each tool offered by Globus Genomics has different memory and compute requirements, which may furthermore vary with input datasets and tool settings. It is important to understand these requirements as they directly impact the required cloud instance types. Rather than consistently launching large instance types – capable of fulfilling all jobs – we manually analysed individual tool requirements to determine the optimal instance type. To model tool requirements we have constructed primitive profiles for a set of frequently run tools as a three-tuple of compute, memory, and disk requirements. This information was collected by executing tools with different input datasets to determine general instance type requirements.

Table 5.1 shows, for ten commonly used tools, both the tool profile and the smallest instance type that can be used to satisfy their requirements. These simple profiles highlight the range of requirements of commonly used tools, with execution time ranging from 15 minutes to almost 6 hours, number of vCPUs (or *virtual CPUs*) from 4 to 32, and memory and disk requirements from low and high.

Tool profiles allow the specification of approximate requirements when submitting jobs to the HTCondor queue using ClassAds [184]. This information is in turn used by the cost-aware provisioning system to restrict the set of eligible instances to those that can satisfy job requirements. In the case of small jobs it increases the search scope and enables a wider range of instances to be considered

which can decrease cost and improve efficiency. It also allows several instances of the same tool to be run concurrently on the same large instance.

5.2 Cost-Aware Provisioning

The cost-aware elastic provisioning system combines job requirements with real-time cloud resource pricing to cost-effectively provision cloud resources. Once instances are requested, a record of the request is stored in an Amazon Relational Database Service (RDS) instance for subsequent analysis.

Algorithm 1 outlines the algorithm used to provision cloud instances. The algorithm periodically monitors an HTCondor queue for unallocated idle jobs whose wait time exceeds a customisable value. Line 12 shows it filtering and ranking viable instance types (based on profiles) from each availability zone to select the most cost-effective instance for a job. A timeout threshold prevents starvation by acquiring On-demand instances when Spot requests are not readily fulfilled. Spot request repurposing (line 27) aims to reduce overall execution time by reusing excess Spot requests. The following sections describe the key features of the provisioning system.

5.2.1 Selecting Viable Instance Types

Before submitting instance requests the provisioning system first consults predefined job profiles to determine which instance types can satisfy the requirements of the job. Generally, the use of profiles greatly increases the number of instance types that can potentially be used to service a job. It therefore increases the potential for cost reduction as additional instance types can be surveyed. 14 different instance types are used in this study, with selection restricted by compute, memory, and disk requirements. In the absence of a job profile, the provisioning system uses a system-wide default instance type that is chosen to be suitably large that it is capable of satisfying the requirements of all jobs.

5.2.2 Cost-Aware Instance Selection

Having identified a set of suitable instances, the provisioning system requests real-time Spot pricing information using AWS APIs. Spot prices are collected for each viable instance type from each availability zone. The resulting prices are

Algorithm 1 EC2 instance provisioning.

```
1: timeout = /* configurable */
2: threshold = /* configurable */
3: while true do
4:    /* periodically run */
5:    idleJobs = jobs in HTCondor queue
6:    for job in idleJobs do
7:       if job not allocated then
8:          if job queue time > timeout then
9:             launch On-demand instance
10:            cancel or repurpose outstanding Spot requests for job
11:         else
12:            eligibleIns = instance types that meet job profile
13:            for instance in eligibleIns do
14:               onDemandP = On-demand price for instance
15:               minZoneSpotP = min zone Spot price for instance
16:               if minZoneSpotP > threshold × onDemandP then
17:                  instancePrice = onDemandP
18:               else
19:                  instancePrice = minZoneSpotP
20:               end if
21:            end for
22:            sort eligibleIns by instancePrice
23:            select instance with lowest instancePrice
24:            launch instance
25:         end if
26:      else
27:         cancel or repurpose outstanding Spot requests for job
28:      end if
29:   end for
30: end while
```

compared with published On-demand prices for each instance type to determine the cheapest real-time price.

Depending on gateway-specific pre-defined policies (such as the maximum time to wait for a Spot request), the cheapest instance type in the cheapest availability zone is generally selected. It may therefore select a faster instance type than is necessary if the cost is lower than the best match. Individual gateways can be configured with maximum bid prices for each instance type ensuring that potential costs can be limited at the expense of execution time. As some applications are more sensitive to termination and gateway administrators may have different levels of risk aversion the provisioning system can be pre-configured with default bidding policies that determine what acquisition model to use: On-demand, Spot, or a combination of both. This work is later extended to using predictive techniques to compute a bid (*cf.* Section 7.3.2).

5.2.3 Reverting to On-demand Instances

The price of Spot instances fluctuates over time depending on demand and the available pool of excess resources. Popular instance types can sometimes have much higher Spot prices than their On-demand equivalents. For example, the price of the m2.4xlarge instance type frequently exceeds $6, more than six times the On-demand price of $0.98, behaviour that is likely attributed to legacy applications that were configured to use this instance type when it was first introduced and have not yet been migrated to newer instance types. As the Spot price increases, so too does the time required to fulfil Spot requests, especially if the bid price is lower than the current Spot price. Moreover, during volatile periods, the risk of having instances terminated is much higher. In response to these challenges, the cost-aware provisioning system includes functionality to revert to using On-demand instances when Spot prices begin to increase (based on analysis of Spot price history). The decision to revert to On-demand instances is based on a pre-defined wait time threshold. If Spot requests exceed this threshold, the cost-aware provisioning system will request the cheapest On-demand instance capable of satisfying the job's requirements. The threshold is configurable for each gateway and is typically set to 20 minutes. This strategy can reduce costs, improve execution time, and minimise the risk of having instances terminated during execution.

5.2.4 Over-provisioning Instance Requests

When making a Spot request it is difficult to predict whether and when the request will be fulfilled. The cost-aware provisioning system includes functionality to configure how frequently new Spot requests are made as well as how many different requests can be made simultaneously for an individual job. This strategy allows multiple instances of different types to be requested for each job (either immediately or over regular intervals). The reason for this approach is that Spot requests can take considerable time to be fulfilled and it is often faster to request a different instance type while waiting for a request to be fulfilled. Once a request is fulfilled, the provisioning system either cancels outstanding requests (if possible) or uses them to satisfy other jobs waiting to be allocated.

5.2.5 Repurposing Instance Requests

Once an instance has been provisioned or a job has been dispatched to existing instances (instances can become available once completing other workloads), unnecessary Spot requests can either be terminated or used to satisfy other idle jobs. In situations with many jobs and high throughput, instance request repurposing presents an opportunity to improve the overall execution time. That is, rather than cancelling requests and incurring the cost of requesting a new instance, the provisioning system can instead assign excess requests to other waiting jobs, assuming that job requirements are satisfied.

5.3 Data Collection

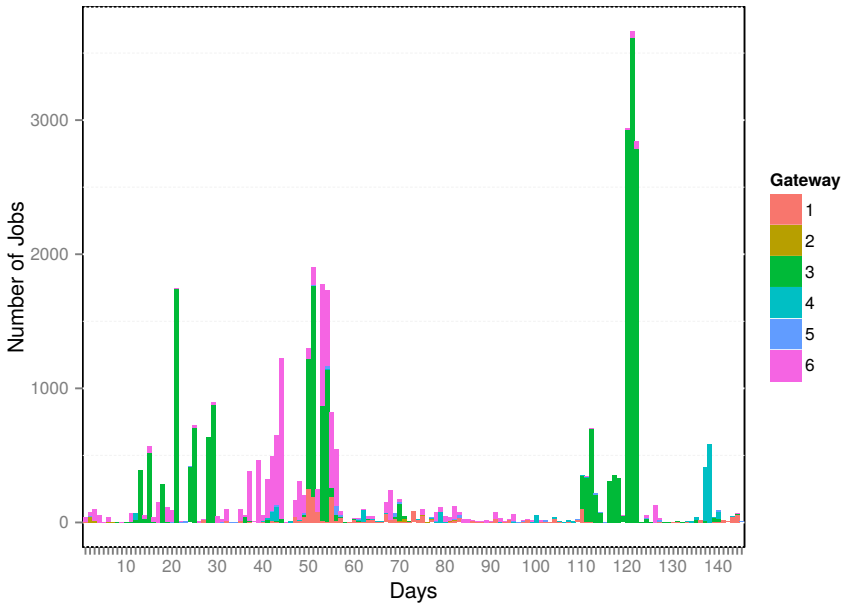
The fact that the Globus Galaxies platform is used extensively for large-scale computing provides a unique opportunity to gather usage information on which to base an evaluation of the cost-aware provisioning system. The following section describes the methods by which this information is recorded and integrated to establish a representative workload.

I instrumented the cost-aware provisioning system to record information regarding provisioning decisions, such as important event times, selected instance type, and availability zone, in an RDS database. The provisioning system logs for each request: 1) the time a request is made, 2) the time the request is satisfied, and 3) the time any new instances are started. This information is stored

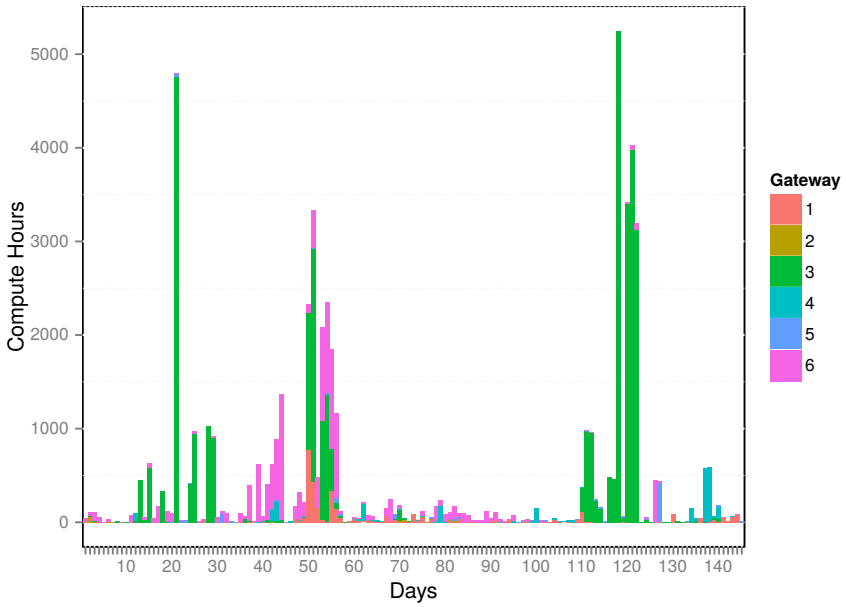
alongside Galaxy records for each execution, including the tool, input datasets, parameters, submitting user, and the identifier (condor ID) assigned to the task by HTCondor. HTCondor records, for each job that is submitted, its identifier, queue time, execution time, and the AWS instance used (the IP address of the host it is dispatched to).

The Galaxy database, HTCondor logs, and the RDS provisioning database are combined to compute the cost of every job that has has been executed by a Globus Galaxies gateway. The AWS API provides historic Spot price for each EC2 instance and availability zone for the previous 90 days. Using the historic Spot price information in conjunction with the logs, which record the time and duration of a job, the cost of the instance used to fulfil a job can be calculated. Similarly, the hypothetical cost of using other instance types to fulfil the job can also be computed. This method of computing the cost of a job enables the emulation of counter-factual provisioning schedules for comparison. Using the unique identifier assigned to every job in the HTCondor logs, every job can be mapped to the instance it was executed on, allowing the cost of the job to be computed. As HTCondor supports a multi-job *slot* model, the total cost is divided across jobs that share a single instance.

Usage data from six Globus Genomics gateways has been collected over a period of 145 days to provide a representative production workload to analyse the cost-aware provisioning system. Figure 5.2a shows the number of jobs executed and Figure 5.2b the total compute time consumed by each gateway over this period. The difference between the two figures shows the distinct usage characteristics of the gateways. For example, it is evident that gateway 1 is primarily used for long running jobs: it has higher compute hours than job executions. Within this period there have been over 35,000 job executions requiring over 53,000 compute hours. These figures highlight the sporadic nature of Globus Genomics usage, where a gateway is often used extensively for a period of days or weeks at a time (such as gateway 3). The workload includes both infrequently used and frequently used Globus Genomics gateways to analyse the provisioning system across a range of scenarios.



(a) Number of jobs executed.



(b) Compute hours used.

Figure 5.2: The number of jobs and compute hours used per day from six Globus Genomics gateways over a 145 day period.

5.4 Analysis

The cost-aware provisioning system is analysed with respect to the execution time and cost of executing workloads. Various search scopes, in terms of instance types and availability zones, are used to investigate the provisioning performance. The scopes considered are:

- **Single-Instance, Single Availability Zone (SI-SAZ)**: A baseline approach in which the provisioning system uses only one instance type (m2.4xlarge) in a single availability zone (us-east-1c).
- **Multi-Instance, Single Availability Zone (MI-SAZ)**: The provisioning system requests different instance types in a single availability zone (us-east-1c) and selects the instance with the lowest price.
- **Single-Instance, Multiple Availability Zone (SI-MAZ)**: The provisioning system requests a single instance type (m2.4xlarge) across all availability zones and selects the instance with the lowest price.
- **Multi-Instance, Multiple Availability Zone (MI-MAZ)**: The provisioning system requests different instance types across all availability zones and selects the instance type with the lowest price.

Based on analysis of the Spot history for various instance types over the past three months a bid price of $6.56 is selected to avoid termination and establish a worst-case scenario (in terms of cost) for each of the provisioning scopes. While individual gateways may use lower bid prices this will result in some reduction in cost at the expense of execution delays incurred while waiting for resources to be provisioned. It is important to note that users are not charged their bid price but rather the Spot price at the time of use. That is, usage of a high bid price adds significant financial risk but not necessarily financial cost. The following results apply this same method by re-calculating Spot prices every hour.

5.4.1 Cost

The total projected cumulative cost for each search scope across six production Globus Genomics gateways over a 145 day period is shown in Figure 5.3. The figure shows, using a naïve approach that considers only a single instance and availability zone (SI-SAZ), costs over $27,000. Whereas, expanding the scope to

multiple instance types (MI-SAZ) reduces the cost to approximately $2,250. Overall the potential savings are $25,486.75, or 92.1% from the worst-case SI-SAZ to the best-case MI-MAZ scope. In comparison to the naïve approach, incorporating multiple availability zones provides a 15.8% improvement. Unexpectedly, the use of multiple availability zones and instance types (MI-MAZ) provides little benefit over simply using multiple instance types (MI-SAZ). The results show that even simple improvements can substantially reduce the cost of executing scientific workloads in the cloud.

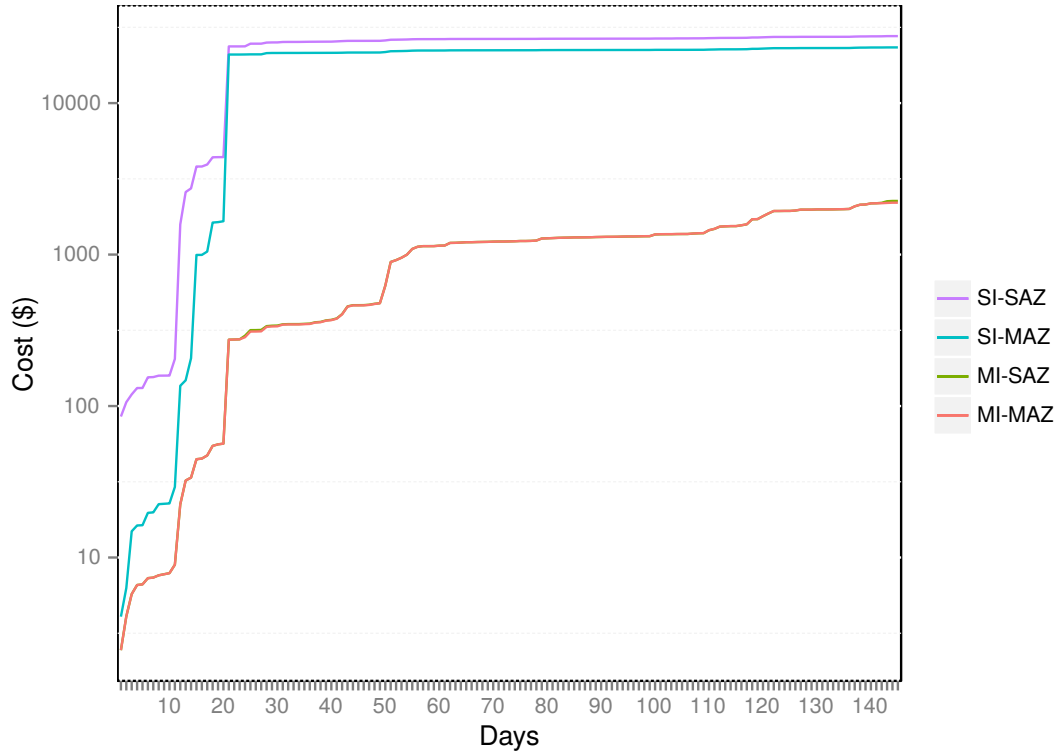


Figure 5.3: The total cumulative cost of operating six Globus Genomics gateways with different provisioning system search scopes over a 145 day period on a logarithmic scale.

Figure 5.4 shows the cumulative cost for each of the six Globus Genomics gateways. The figure shows the total cost when using the naïve, SI-SAZ, approach (solid) as well as the reduced cost achieved by the cost-aware, MI-MAZ, provisioning system (dashed) over the 145 day period. The results show that substantial reductions in cost can be achieved for each of the gateways irrespective of their workload requirements. A breakdown of each of the provisioning scopes

for each gateway is presented in Table 5.2. This shows, on average, that an individual gateway can reduce its operating costs by 75.9% (Max 95.0%, Min 43.0%) when using multiple instance types and availability zones over the baseline SI-SAZ approach.

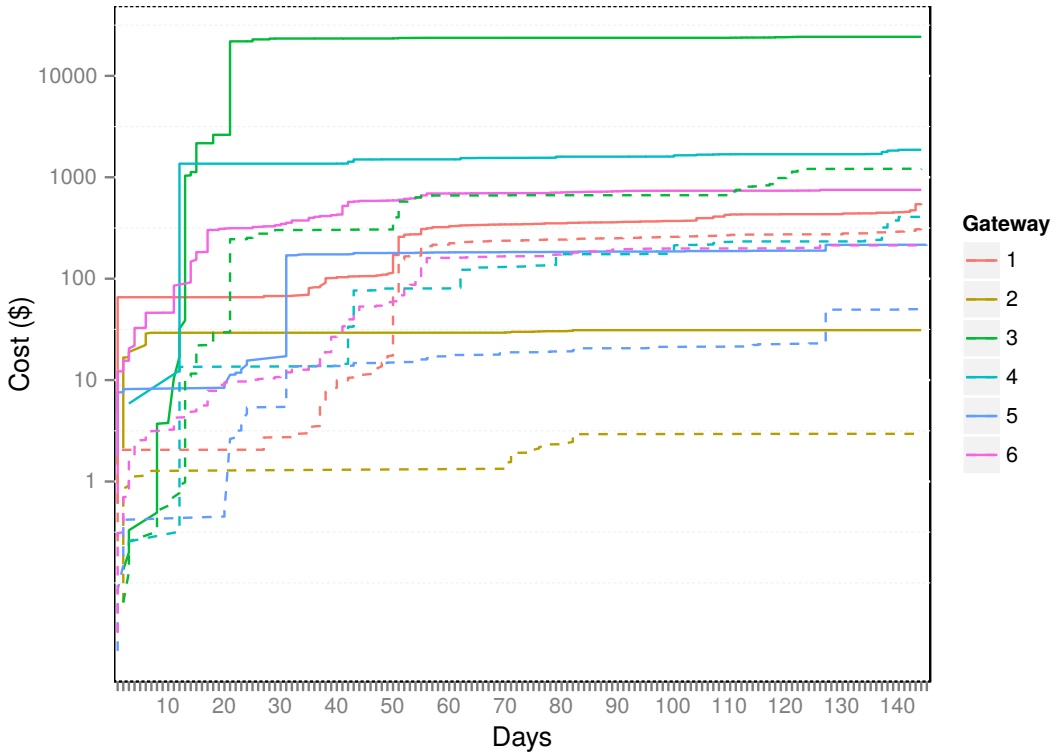


Figure 5.4: The cumulative cost of the naïve SI-SAZ (solid line) and MI-MAZ (dashed line) search scopes for each of the six Globus Genomics gateways over a 145 day period on a logarithmic scale.

5.4.2 Spot Instance Termination

Applying real Spot price traces (obtained from AWS APIs) to each of the provisioning scopes enables identification of when the Spot price would of exceeded the bid price and therefore when an instance would be terminated. Figure 5.5 shows the frequency of instance termination for each of the provisioning scopes with varying bid prices (in $0.25 increments). The results show that the single-instance scopes (utilising m2.4xlarge instances) are highly susceptible to instance termination, as demand for a specific instance type can effect the Spot price in all availability zones. The figure also demonstrates a lack of sensitivity to bid

Table 5.2: Total 145 day cost comparison between gateways.

Gateway	SI-SAZ	SI-MAZ	MI-SAZ	MI-MAZ
1	551.38	341.20	371.57	314.16
2	31.08	4.86	2.95	2.95
3	24269.89	22080.71	1219.03	1213.44
4	1867.33	499.92	410.07	406.55
5	216.08	55.00	46.48	50.08
6	751.25	339.59	209.26	213.05
Total	**27686.99**	**23321.29**	**2259.36**	**2200.24**

prices as the rate of termination does not change significantly with different bid prices. This implies that when an instance type experiences an increase in demand (which increases the Spot price) it will often increase substantially. Thus, when configuring bid prices, they can be defined to be high (to avoid termination) or low if they can easily repurpose workloads. Similarly, On-demand instances should be used if the Spot price increases as it is likely to increase significantly.

5.4.3 Reverting to On-demand Instances

As shown in Figure 5.5 there is potential for Spot instances to be terminated during job execution. The cost-aware provisioning system is configured to revert to On-demand instances in some situations to minimise execution time. However, when reverting to On-demand instances there is a trade-off between execution time and cost. This trade-off is shown in Figure 5.6 and Figure 5.7. These figures show the total cost and execution time when five timeout values (None, 5, 10, 15, and 20 minutes) are used and the search scope is set to MI-MAZ. The timeout value specifies how long a job is allowed to wait for a Spot instance before reverting to an On-demand instance. The results indicate that short timeout periods can improve overall execution time at the expense of substantially increasing the operating cost of a gateway (a 5 minute timeout results in increased cost over eight times the MI-MAZ price). The substantial increase in cost is reflective of the competitive Spot pricing market in which Spot prices across availability zones rarely remain higher than On-demand prices. The effect of timeouts has also been evaluated using the SI-SAZ scope and found that the total cost was lower when a 5 minute timeout value was used. This was due to a prolonged increased Spot

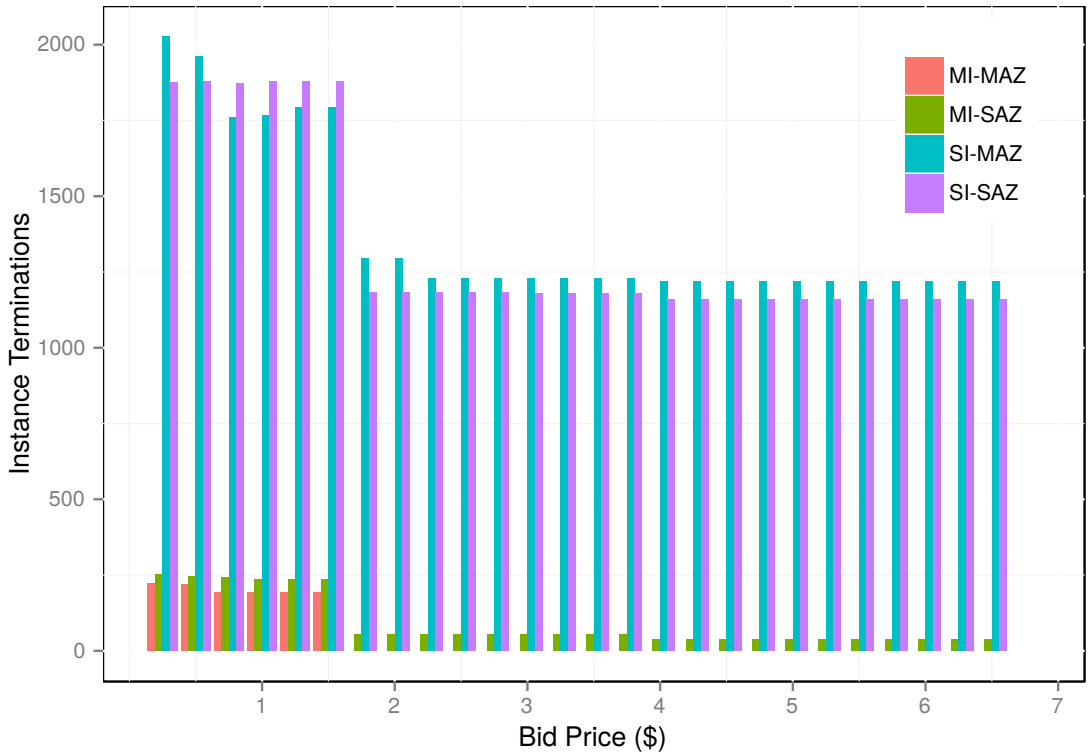


Figure 5.5: The number of Spot instance terminations for each search scope with $0.25 bid price increments.

price in the selected availability zone and a lack of alternative availability zones in which to provision new instances.

5.4.4 Production Usage

To analyse the performance of the cost-aware provisioning system in its entirety I compare the performance of three production Globus Genomics gateways before and after the provisioning system was deployed. Without the provisioning system the gateways employ a simple, yet common, approach that acquires a single instance type (in this case m2.4xlarge) in a single availability zone. This provisioner operates periodically by polling the HTCondor queue to find idle jobs. If the job can not be scheduled to idle instances, the provisioner acquires an instance to fulfil the workload. With the cost-aware provisioning system deployed the gateways select the cheapest instance type and availability zone, allows request repurposing, and has the ability to revert to On-demand instances when needed. In both cases the length of time that jobs must wait before being allo-

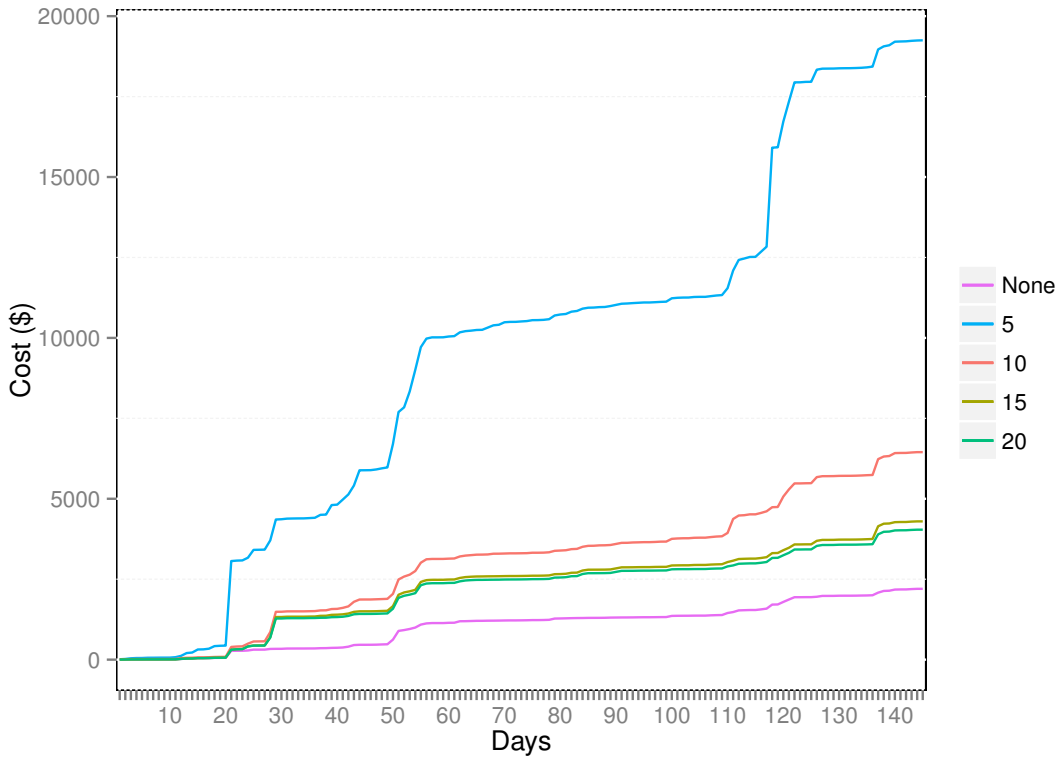


Figure 5.6: Cumulative cost when using On-demand instances with different timeout periods (minutes) using MI-MAZ scope.

cated to a provisioned resource is recorded. Only jobs that wait for a resource to be acquired are used, rather than those that may be allocated to existing resources. Logs are analysed from a 168 day period where the simple provisioner was used are compared against the logs of the subsequent 80 days where the cost-aware provisioning system has been deployed. In total 1775 jobs are allocated with the simple provisioner, and 2302 are allocated using the cost-aware provisioning system. When using the simple provisioner jobs wait, on average, 650 seconds before being executed. When using the cost-aware provisioning system the average wait time is reduced to 570 seconds, an average reduction of 80 seconds per job. This represents a 12.3% reduction in the time spent waiting for an instance to be provisioned. While some of this improvement may be attributed to different workloads and economic conditions at the time of analysis it may also be due to the ability to repurpose requests. A comparison of the two workloads in Table 5.3 shows that the average arrival time and execution time of jobs is relatively consistent.

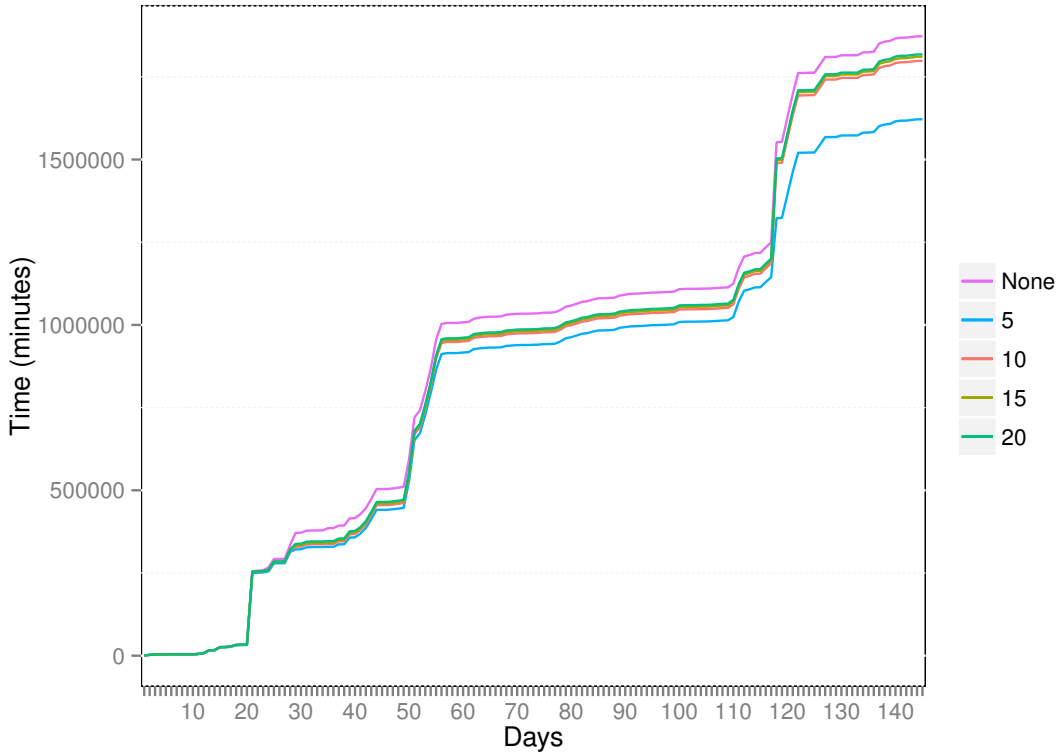


Figure 5.7: Cumulative execution time when using On-demand instances with different timeout periods (minutes) using MI-MAZ scope.

Table 5.3: Jobs allocated with and without the provisioning system.

Provisioner	Jobs	Avg Wait	Avg Exec	Avg Arrival Time
Simple	1775	0:10:50	3:54:07	13:38:59
Cost-aware	2302	0:09:30	3:37:01	13:58:05

5.5 Summary

Acquiring instances in a cost-aware manner is a challenge faced by cloud-based scientific services. The second research question investigated in this thesis, **RQ2**, asks how cloud computing markets can be exploited to minimise the cost of provisioning infrastructure to facilitate on-demand computation.

This chapter explored cost-aware techniques and presented a multi-faceted, cost-aware, provisioning system that automatically selects and provisions cloud instances. Analysis of six production Globus Genomics gateways demonstrated that the cost-aware provisioning system can reduce costs, improve execution

time, and minimise Spot instance terminations. The analysis showed that cost-aware provisioning can result in substantial economic advantages over simple provisioning approaches. Increasing the provisioning system's search scope to consider additional instance types and availability zones resulted in a 92.1% reduction in overall cost between the worst-case SI-SAZ and best-case MI-MAZ scopes. Overall, the findings presented in this chapter address **RQ2** by identifying provisioning techniques that can minimise the cost of using cloud infrastructure.

Chapter 6

Profiling Workloads

Cloud providers can accommodate the execution of a wide range of custom analytical tools with different configurations. Cloud instance types are heterogeneous, coming in many different shapes and sizes, providing unique capabilities and specialisations. This means tools can be deployed to an instance type that best suits their needs. However, in order to accurately select an instance type to optimally execute a tool, the resource requirements of the tool must be known. The challenge of selecting appropriate instance types for unknown applications prompted the third research question, **RQ3**, which asks how the resource requirements of unknown applications can be computed to determine appropriate instance types. To explore **RQ3** I investigate techniques to reveal the resource requirements of applications such that they can be mapped to instance types. The fourth research question, **RQ4**, asks how this can be achieved automatically. To investigate this question I explore techniques to dynamically incorporate resource requirements into the provisioning decisions.

This chapter presents the design, implementation, and evaluation of a novel cloud profiling service. The purpose of this service is to automate the creation of tool (any executable application or Docker [185] container) profiles, concise descriptions of the performance and CPU, memory, network, and disk requirements of a supplied tool under different environments and scenarios. These profiles can then be used to optimise instance selection, enable tool comparison, and improve provisioning and scheduling algorithms. The profiling service is self-contained and can be deployed externally, and used to automatically profile tools on different cloud instances with different input configurations. The service can automatically provision test instances, deploy and configure tools on provisioned

instances, stage input datasets, monitor tool execution, record fine-grained tool behaviour, and construct and return resource utilisation profiles.

The same application may perform very differently on distinct cloud instance types [25]. Different instances can also have quite different costs [27]. Thus, effective use of cloud infrastructure requires that the user understand how an application behaves, such that the most suitable instance type can be used and configured. Good choices become more important when large amounts of computing are to be performed, as is the case with scientific services, gateways, and portals that offer access to a set of predefined tools. For example, the Globus Galaxies platform consumed more than half a million EC2 instance hours in 2015, for a total cost of more than $150,000. The performance and cost of executing a single job can vary by an order of magnitude or more on different instance types, meaning it is easy to make expensive mistakes when choosing cloud instance types.

Five commonly used Globus Genomics tools are used to evaluate the profiling service. For each of these tools I compute profiles using representative input data and settings (derived from production usage) and evaluate the extent to which profiles can be used to improve performance and decrease cost for Globus Genomics users. Analysis shows that selecting instance types based on tool profiles can reduce execution time by up to 15.7% and costs by up to 86.6% relative to simpler heuristics that do not consider profiles. The profiling service is later used by SCRIMP to improve the provisioning and deployment processes.

6.1 Profiling Service

The information included in a tool profile can range from a binary measure of whether a tool can execute on a specific instance through to fine-grained representation of resource usage over time. Manually constructing these profiles can be time-consuming due to the numerous permutations that must be considered, as well as the frequency with which tools and instance types change or are contributed to services. For example, tool performance is dependent on instance type, specific settings of that instance (e.g., optimised storage and network), input datasets, and invocation parameters. Having identified the range of configurations to be profiled, users must then painstakingly embark on a trial-and-error analysis of instance types by deploying suitable infrastructure, configuring and installing the tool and its dependencies, and measuring tool performance and

resource usage.

To address this challenge, the profiling service is designed to enable the automated execution and monitoring of arbitrary tools over any cloud instance type. A tool can be provided as a self-contained package with the application executable and a collection of input data files, as a contextualisation script that can be used to install the tool and its dependencies on-demand, or as a Docker container. The user specifies the instance types and configurations to be evaluated and the profiling service then executes the application on all specified combinations and collects performance metrics periodically. Performance Co-Pilot (PCP) [186] is used to collect performance metrics as it is light-weight and has little overhead on application execution.

6.1.1 Architecture

The profiling service is implemented as a Web application. As shown in Figure 6.1, it includes the profiling Web service, a reliable database (hosted on AWS RDS), and a dynamic pool of provisioned worker nodes, each with a management Web service for control and monitoring tool execution.

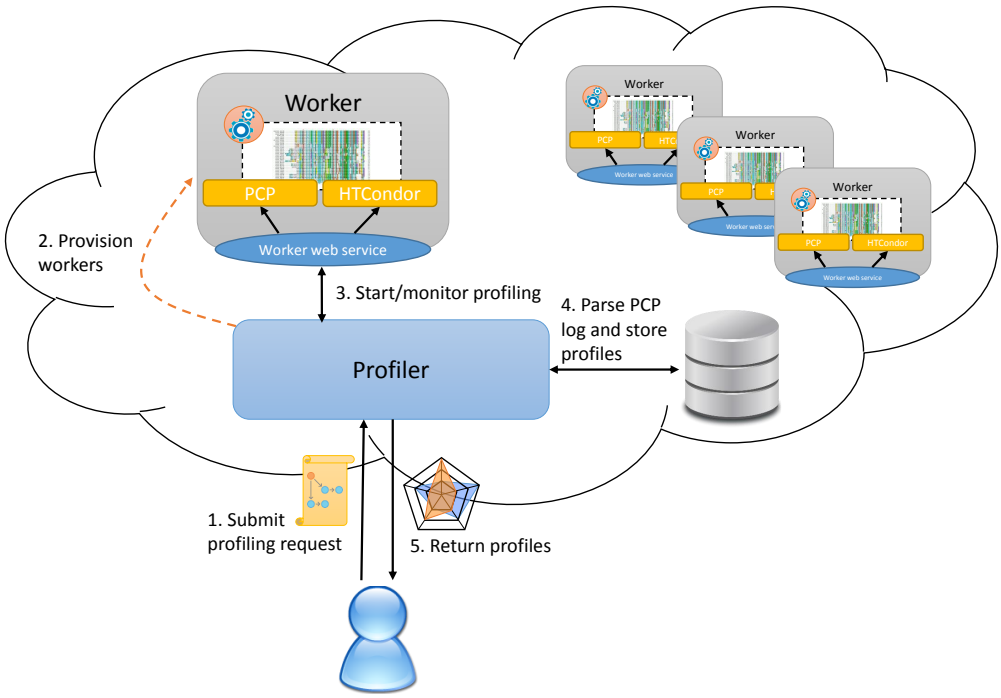


Figure 6.1: An outline of the profiling service's architecture.

The profiling service exposes a REST API for requesting and managing profiles. The service can be deployed on any cloud instance configured with access to a profile database (local or RDS) and a mounted NFS shared file system. The profiling service uses the shared file system for staging data and tools to worker nodes.

The profiling service is implemented as a multi-threaded Python application. When a profiling job is requested the service creates a new thread that is responsible for overseeing the execution and monitoring of the tool. The thread will provision EC2 instances, stage/deploy the input data and tool, monitor execution, parse monitoring logs, and store the profile in the database.

Each worker node is deployed with a dynamic Web service that allows the provisioning service to control and monitor the executed tool. It is also configured with a local HTCondor client to manage the execution of the tool locally. Cloud-init [179] is used to dynamically contextualise the worker instance after it is provisioned. This contextualisation process installs and configures the worker Web service, HTCondor, and PCP, as well as mounting the shared file system.

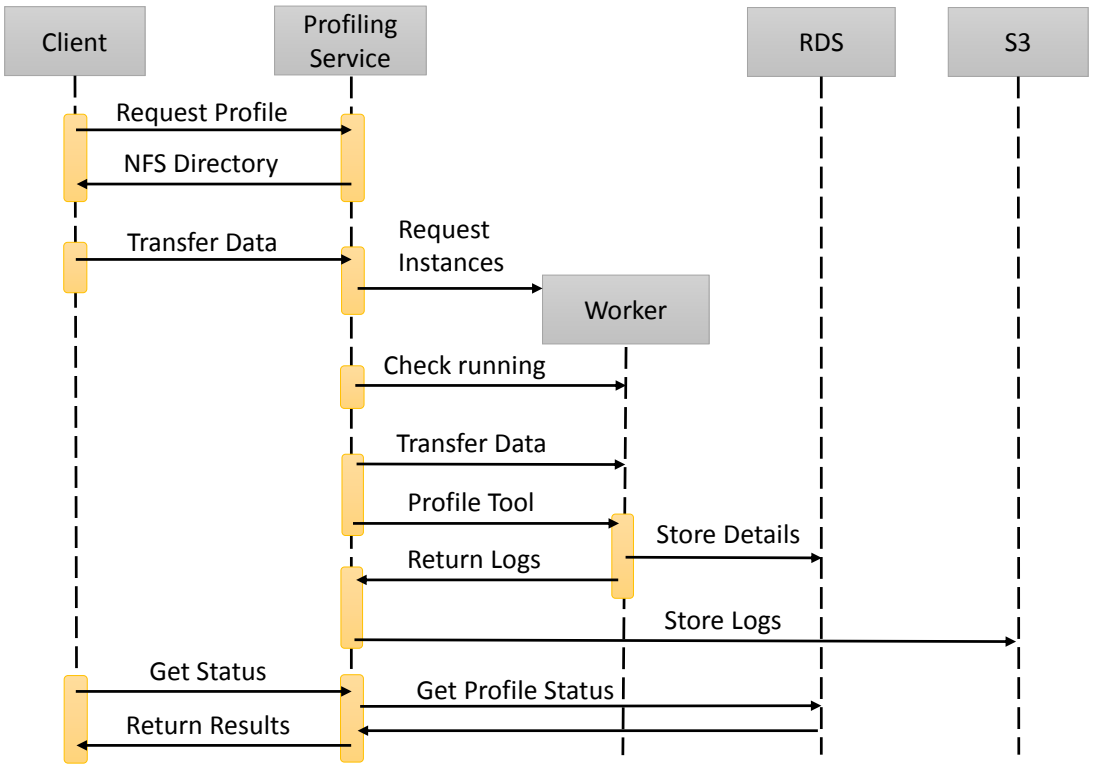


Figure 6.2: A sequence diagram showing the steps involved when the profiling service profiles a tool.

6.1.2 Profiling Process

Figure 6.2 shows the profiling process. A user requests a new profile by submitting a JSON profile request (Listing 6.1) specifying the tool, instance types, and instance and tool configurations. The profiling service includes a light-weight client application to simplify interactions with the profiling service API. The client interprets the user's JSON profile requests to initiate and manage the employment of the profiling service.

In response to a profile request the profiling service creates a unique working directory for each profiling job and returns this path to the client. The client can then optionally upload the tool and any input files used for execution. Alternatively, the client may choose to upload a contextualisation script for deploying the tool automatically. A profiling service thread is started for each instance type and configuration to manage provisioning and tool execution. The job request may optionally specify other properties such as target availability zones and cloud acquisition model: either On-demand or Spot pricing. In the case of Spot pricing, the request may also optionally specify a bid price. If no bid price is set the maximum bid is used. The techniques described in Chapter 5 are used to ensure the necessary workers are provisioned cost-effectively.

Once a cloud instance is provisioned, cloud-init is used to contextualise the worker. This process includes the installation and configuration of the worker Web service and the mounting of the shared file system. Once the Web service becomes responsive, the profiler thread initiates a request to the worker's Web service indicating the executable to be profiled and the location of the workload. The worker service then copies the input data to its local drive from the shared file system. This is done to eliminate overhead incurred from accessing the shared file system during profiling. The worker service then processes the executable and appends a command to isolate execution within the local working directory. In addition, a string template is used to swap any overriding configurable parameters based on the instance type. For example, a client may specify the number of threads to use on each instance type to make optimal use of available cores. Immediately prior to executing the workload, a PCP logger is initiated to collect performance statistics on the host machine at one second intervals. The workload is then submitted to HTCondor to be executed on a local slot. During execution, PCP appends resource usage statistics to a pre-defined log file.

```
{
 name: 'my profile',
 executable: 'my_app.sh',
 type: {script|docker}
 workloads: {
  workload_1: ['file1', 'file2'],
 }, configurations: {
  config_1: '--config 1 --config 2',
  config_2: '--config 1 --config 3',
 }, instance_types : [{
   type: 'r3.xlarge',
   override: '$threads=8',
   acquisition: 'spot',
   bid: 6
  },{
   type: 'c3.8xlarge',
         override: '$threads=32'
  }]
}
```

Listing 6.1: A partial JSON profile request.

The profiling service probes each worker periodically to determine the status of the job being profiled, and records these progress markers in the database. Once the workload completes the PCP logger is halted and the exit code and execution time are updated in the database. The PCP logs are then transferred to the shared file system and the profiler thread parses them into CSV format. The resulting CSV file is uploaded to an S3 bucket for long-term storage. Throughout the profiling process the client can request status updates. Once the profiling job is complete the client can download the resulting profiles. Listing 6.2 shows an example profile, which includes high level summary information for each instance type as well as the location of the stored CSV logs for each configuration.

The profiler is also capable of profiling Docker containers [185]. To profile a container the *type* attribute in Listing 6.1 must be set to *Docker* and the *executable* field should specify a Docker run command. When using the Docker configuration of the profiler the worker nodes are contextualised via cloud-init to start an InfluxData [187] (database) container and Google's cAdvisor [188] is used to stream Docker performance statistics to the InfluxData database. The user's Docker run command is modified to give the container a name with its job number as the suffix. This allows the container to be monitored and its perfor-

mance information to be searched in the InfluxData database. Once the container finishes running, a profile is created by querying the InfluxData's Web API in order to process the logs to determine resource utilisation. The resulting profiles are then returned to the user in the same way as described above.

```
{
 name: 'my profile',
 executable: 'my_app.sh',
 log_files: 's3/output_bucket',
 results: [  {
    workload: ['file1', 'file2'],
    configuration: '--config 1 --config 2',
    instance_type: {
      type: 'c3.8xlarge',
      override: '$threads=32',
      exit_status: 'Completed'
      execution_time: 8399
      performance: {
        memory: { ... },
        cpu: { ... },
        disk: { ... },
        network: { ... }
      }
    }
  }, ... ]
}
```

Listing 6.2: A partial JSON tool profile.

6.2 Creating Genomics Tool Profiles

The following five genomics tools are used to evaluate the capabilities of the profiling service. These tools are among the most frequently used tools on Globus Genomics, accounting for 17.7% of compute time consumed on four production gateways (summary statistics are included in Table 6.1). They also have varied resource requirements, ranging in execution time from ten minutes to more than an hour.

1. **FastQC** [189] provides quality control on raw sequence data.

Table 6.1: Globus Genomics tool execution statistics.

Tool	Times Run	Exec Freq Rank	Exec %	Avg Exec Time (hours)
BWA MEM	1473	4	5.9%	00:38:22
FastQC	1471	5	5.9%	00:29:13
MarkDups	1341	7	5.3%	00:35:43
Bowtie	85	39	0.3%	01:23:57
BWA ALN	83	40	0.3%	00:20:40

2. **PICARD MarkDuplicates (MarkDups)** [190] flags or removes duplicate reads in a SAM or BAM file.

3. **BWA ALN** [191] aligns short sequences to a reference sequence (e.g., the human genome).

4. **Bowtie** [192] is a short read aligner designed for high performance and memory efficiency.

5. **BWA MEM** [193] maps low-divergent sequences against large reference genomes. It chooses automatically between local and end-to-end alignments, supports paired-end reads, and performs chimeric alignment.

6.2.1 AWS Testbed

Ten different AWS instance types have been selected to create tool profiles. For consistency, only instance types that support *instance storage* are used. That is, instance types that use EBS storage (m4, c4, etc.) are not used, as this would likely have a significant affect on application performance when compared to those that use instance storage. The instances type families selected are: the M3 (general purpose), G2 (GPU enabled), C3 (compute optimised) and R3 (memory optimised). All profiling is performed in the US-East-1 AWS region, using Spot instances in the cheapest availability zone at the time of profiling. The selected instance types are described in Table 6.2.

Table 6.2: AWS instance types.

Type	Family	vCPU	Memory (GB)	Storage (GB)	Network
c3.2xlarge	Compute optimised	8	15	2 x 80	High
c3.4xlarge	Compute optimised	16	30	2 x 160	High
c3.8xlarge	Compute optimised	32	60	2 x 320	High
g2.2xlarge	GPU instances	8	15	1 x 60	High
g2.8xlarge	GPU instances	32	60	2 x 120	10 Gigabit
r3.xlarge	Memory optimised	4	30.5	1 x 80	Moderate
r3.2xlarge	Memory optimised	8	61	1 x 160	High
r3.4xlarge	Memory optimised	16	122	1 x 320	High
r3.8xlarge	Memory optimised	32	244	2 x 320	10 Gigabit
m3.2xlarge	General purpose	8	30	2 x 80	High

6.2.2 Profiles

Table 6.3 summarises profile data for the five genomics tools on four R3 instance types. These instance types range in capacity from r3.xlarge, with 4 vCPUs and 30.5GB of memory, through to r3.8xlarge, which has 32 vCPUs and 244GB of memory. Where possible, each profiled tool is dynamically configured to use all available vCPUs and memory during execution. Table 6.3 highlights the richness of the profile information collected. It also shows fairly consistent usage across instance types, indicating that execution time and resource requirements could be forecast for other instance types.

The results show the performance of a tool is not entirely dependent on instance type. For example, *Bowtie* consumes roughly the same amount of memory regardless of the amount of available memory, number of threads, and vCPUs. *FastQC* executes for roughly the same amount of time on all of the employed instances. This is because it uses only a single vCPU rather than all available vCPUs. The most memory-intensive of the tools, *BWA ALN* and *BWA MEM*, fail to execute on the smallest r3.xlarge instance type. Both of these tools load the entire reference dataset, which is over 30GB, into memory for processing. However, the r3.xlarge instance type has only 30.5GB of memory which results in the tools exhausting the machine's free memory during execution. These results present examples of where the provisioning and deployment processes could be improved by incorporating information on a tool's behaviour and resource requirements. For example, larger instances should be provisioned for data-intensive tools to

avoid execution failures and could also be partitioned to concurrently satisfy multiple tools with static requirements.

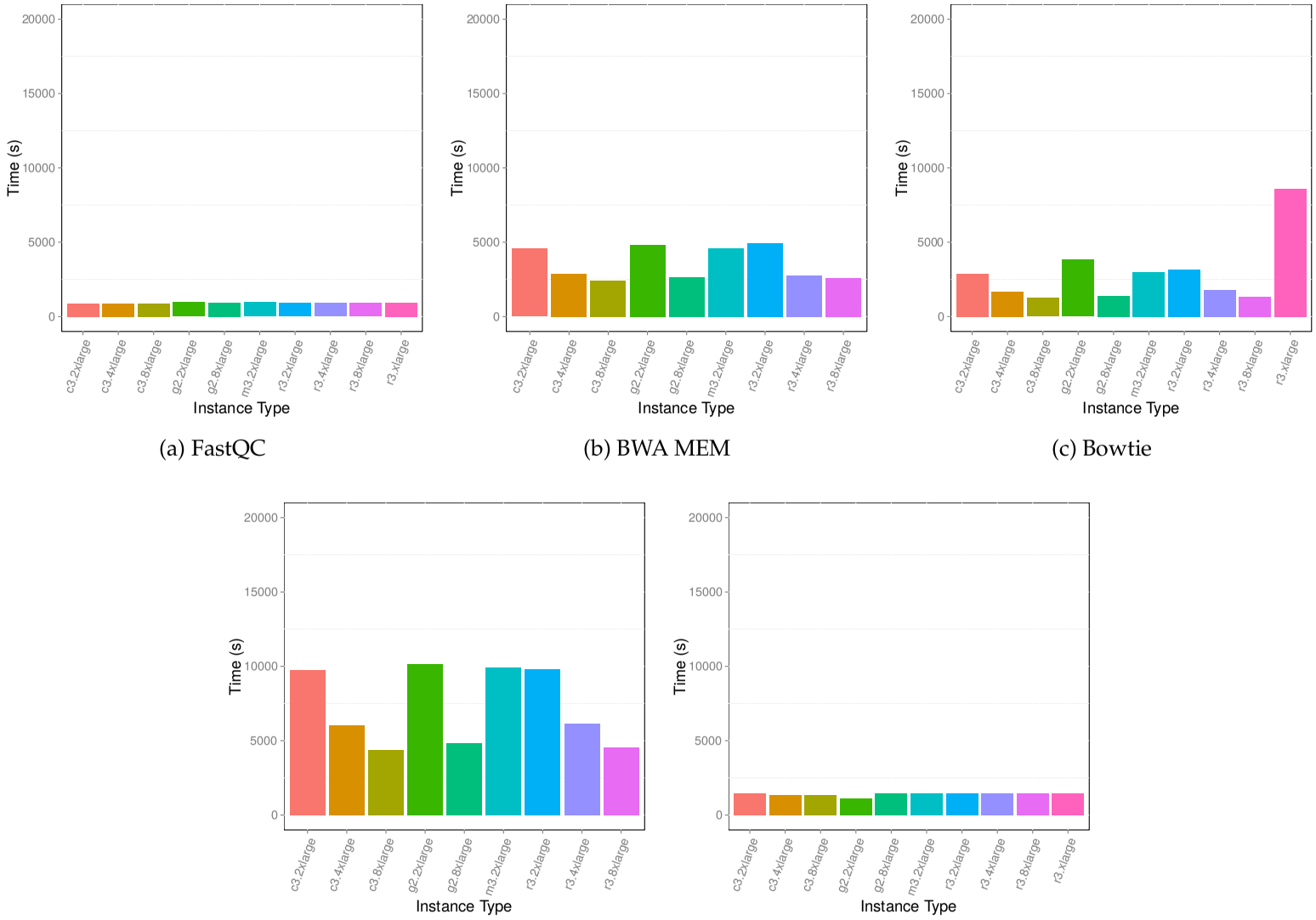
Disk and network usage are minimal for each of the five tools profiled. Network usage is low as the profiling service attempts to minimise network overhead by staging data to the local file system. In addition, the tools themselves are loosely coupled and are executed on a single instance, requiring no network communication. Disk usage is (near) identical between instance types because the tools are executed with the same input datasets and settings, thus, the resulting data written to disk is also the same. An exception is seen where small instance types show increased disk usage as the disk is used for swap space when memory requirements exceed capacity.

6.2.3 Execution Performance

Figure 6.3 shows the execution time for each tool on each instance type. As expected, the tools perform differently on different instance types. For example, *Bowtie* (Figure 6.3c) and *BWA ALN* (Figure 6.3d) exhibit a strong correlation between the number of vCPUs and execution time. In contrast, *FastQC* (Figure 6.3a), which uses only a single vCPU, performs similarly regardless of instance type. In all cases, the tools execute fastest on c3.4xlarge, c3.8xlarge, g2.8xlarge, r3.4xlarge, and r3.8xlarge. These instances have at least 16 cores and 60 GB of memory each, with the exception of c3.4xlarge with only 30 GB of memory. In fact, the largest difference in execution time across these five instance types is 31%, with an average difference across all tools of 19.48% (*Bowtie*: 31%, *BWA ALN*: 28.4%, *BWA MEM*: 21.7%, *FastQC*: 8.8%, *MarkDups*: 7.5%), indicating that these instance types can be used somewhat interchangeably for these tools. It is important to note that tool performance is heavily dependent on the input dataset used.

Table 6.3: Genomics tool profiles over R3 instances. Note: * indicates tool failure.

Instance	Tool	Time (H)	Memory (MB)	Memory %	Idle CPU %	User CPU %	Sys CPU %	Disk-R (MB)	Disk-W (MB)
r3.xlarge	Bowtie	2:23	6920.54	22.56	0.35	86.62	12.97	7.46	9526.87
r3.2xlarge	Bowtie	0:52	6011.29	9.78	0.08	76.52	23.37	0.35	10334.18
r3.4xlarge	Bowtie	0:29	6120.88	4.98	0.33	65.94	33.59	0.12	7673.24
r3.8xlarge	Bowtie	0:22	6474.33	2.63	2.51	49.73	47.48	0.42	7123.70
*r3.xlarge	BWA MEM	2:37	24314.41	79.25	0.65	98.53	0.39	11812.97	2545.70
r3.2xlarge	BWA MEM	1:22	10100.33	16.44	4.18	94.79	0.57	3700.75	8785.78
r3.4xlarge	BWA MEM	1:19	37935.04	30.85	13.77	85.30	0.49	0.35	8482.01
r3.8xlarge	BWA MEM	0:43	41279.97	16.78	47.10	49.02	2.92	0.28	9030.93
r3.xlarge	FastQC	0:15	54.23	0.18	75.05	24.17	0.53	29026.68	15.97
r3.2xlarge	FastQC	0:15	31.07	0.05	87.37	12.33	0.13	5.45	583.09
r3.4xlarge	FastQC	0:15	233.66	0.19	93.69	6.18	0.08	0.02	962.15
r3.8xlarge	FastQC	0:14	235.66	0.10	96.84	3.10	0.04	0.02	27.75
*r3.xlarge	BWA ALN	5:24	2311.53	7.53	2.76	96.98	0.20	29.04	9009.79
r3.2xlarge	BWA ALN	2:43	32760.49	53.32	22.59	76.57	0.60	3.99	31631.29
r3.4xlarge	BWA ALN	1:40	36748.67	29.89	37.68	61.68	0.52	1.18	32273.80
r3.8xlarge	BWA ALN	1:15	34278.09	13.93	53.47	45.80	0.64	1.18	31886.82
r3.xlarge	MarkDups	0:24	11512.12	37.52	70.85	27.18	0.50	4927.71	6928.91
r3.2xlarge	MarkDups	0:23	41639.11	67.77	83.46	15.93	0.24	0.22	6681.06
r3.4xlarge	MarkDups	0:22	69226.44	56.30	90.80	8.95	0.15	0.45	6989.04
r3.8xlarge	MarkDups	0:24	52198.18	21.22	93.12	6.69	0.11	0.45	7215.32



(a) FastQC (b) BWA MEM (c) Bowtie

(d) BWA ALN (e) MarkDups

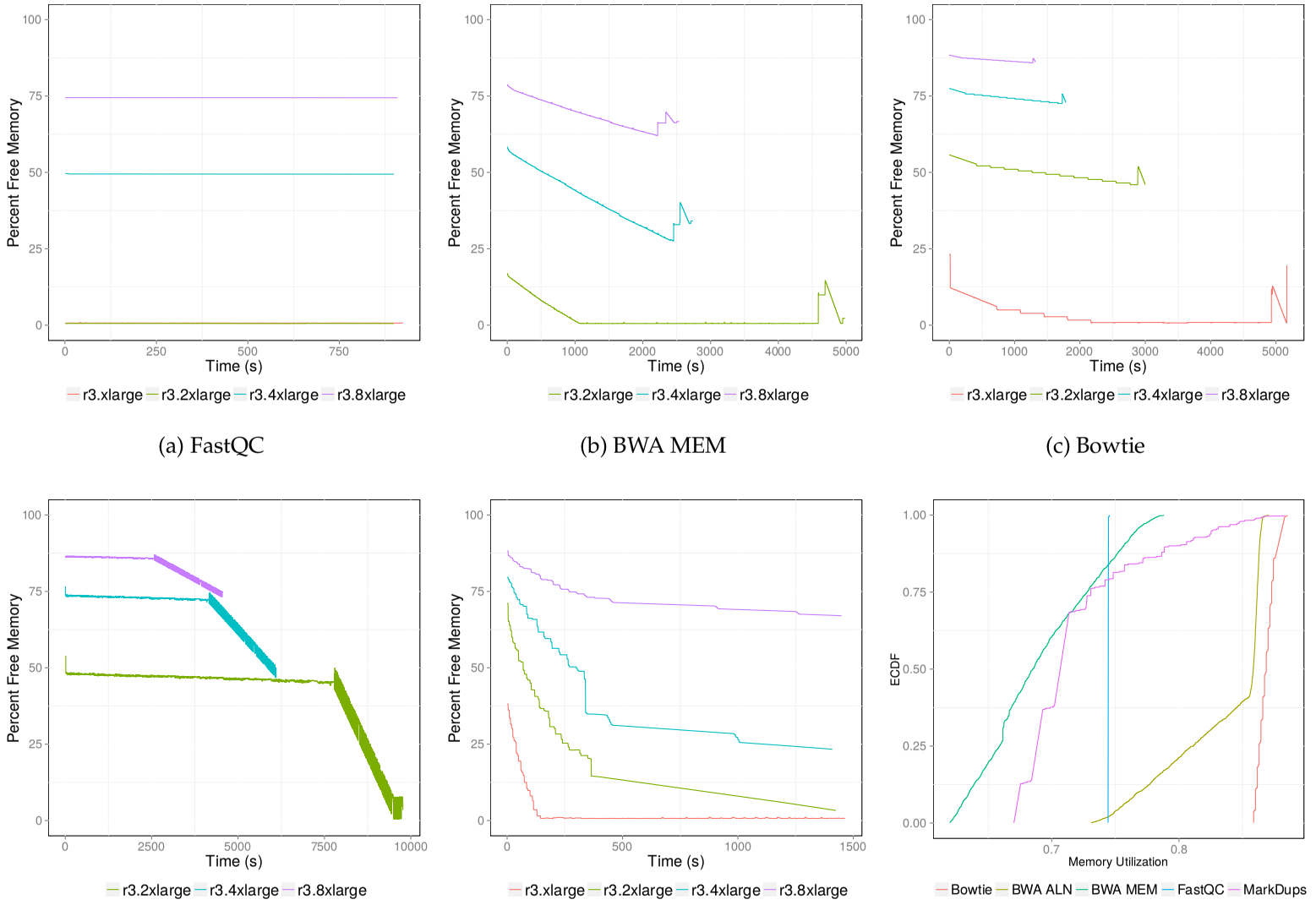
Figure 6.3: The execution time of the tools over various instance types.

6.2.4 Resource Usage

The profiling service can capture fine-grained execution characteristics. By default, the profiling service is configured to monitor resource usage every second. The following discusses investigation into the resource utilisation and behaviour of the profiled tools over the R3 family of instance types.

Figure 6.4a–6.4e show temporal memory profiles and Figure 6.4f the empirical cumulative distribution function (ECDF) of free memory for each tool on r3.8xlarge instances. The memory profile gives the amount of free memory, as a percentage of total instance memory, for each second of execution. Figure 6.4b shows that *BWA MEM* has two distinct phases of execution: first, memory usage gradually decreases over the first phase of execution, before it is re-acquired towards the end of execution for a secondary phase. This is because the *BWA MEM* tool loads the entire input dataset into memory and pipes output to a sorting phase (which reduces memory usage). Once the initial phase ends, memory is released and the sorted output is written to disk. *BWA ALN*'s memory use (Figure 6.4d) also presents an interesting pattern. This tool's memory profile is governed by two phases, with a substantial change in memory usage approximately half way through the execution. All of the tools use only a fraction of the available memory when executed on large instance types, indicating that, from a memory perspective, each tool could be executed concurrently on these instance types.

The ECDFs in Figure 6.4f show the proportion of time spent with various amounts of free memory throughout execution. Each of the five profiled tools exhibits distinct and identifiable ECDF curves. For example, the *Bowtie* tool exhibits bimodality which manifests as memory utilisation occurring with the greatest probability in two distinct phases while *FastQC* has constant memory utilisation and *MarkDups* exhibits high kurtosis (a measure of whether data is peaked or flat relative to a normal distribution).



(a) FastQC (b) BWA MEM (c) Bowtie

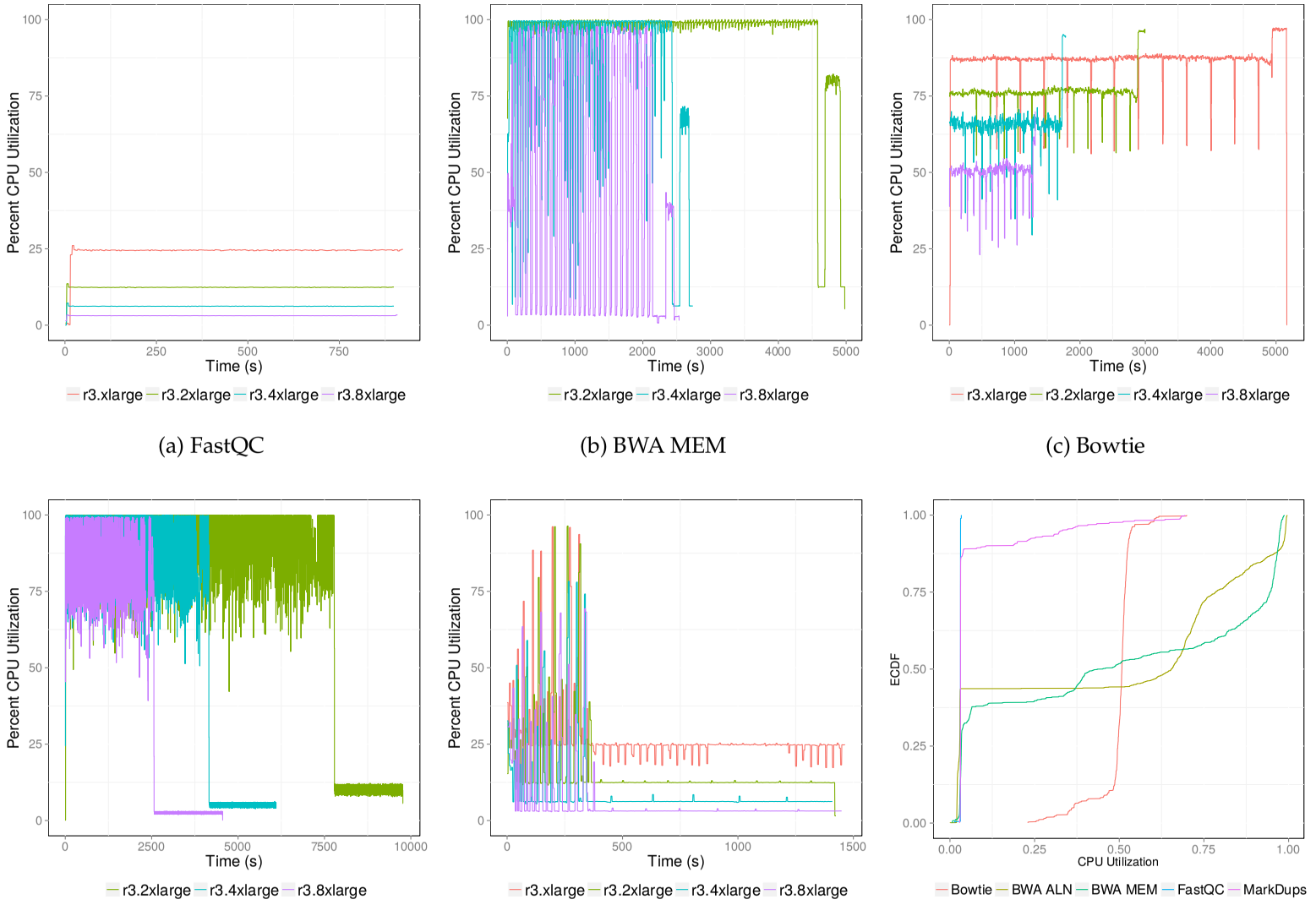
(d) BWA ALN (e) MarkDups (f) ECDF of Memory Utilisation

Figure 6.4: The percentage of free memory for each instance type during tool execution (a) - (e). Empirical cumulative distribution function of free memory for each workload over the r3.8xlarge instance type.

Figure 6.5a–6.5e plot the temporal CPU profile while Figure 6.5f shows the ECDF of CPU usage for each tool on r3.8xlarge instances. *FastQC* CPU usage appears as flat lines in Figure 6.5a and a vertical line in the ECDF Figure 6.5f with very little CPU utilisation because only a single core (out of thirty two available cores) is used. *BWA MEM* (Figure 6.5b) and *BWA ALN* (Figure 6.5d) consume most of the instance's CPU resources for approximately half of their execution time. Both tools exhibit symmetrical ECDF curves with low kurtosis for CPU utilisation, suggesting that the computational intensity of these tools (Figure 6.5f) is due to frequent fluctuations. *MarkDups* (Figure 6.5e) exhibits volatile CPU usage for the first third of its execution, from that point, CPU usage is constant and below 25%. Finally, *Bowtie* (Figure 6.5c) uses a constant amount of CPU throughout execution. Interestingly, this tool does not use all available CPU resources: it uses only 50% on the 32 vCPU r3.8xlarge but 80% on the four vCPUs instance. This result implies that CPU usage is constrained by another factor.

6.2.5 Discussion

The profiles highlight the practical benefits of selecting appropriate instance types for tools. While any one tool may execute on a range of different instance types, its resource consumption may vary across different types, as may its response to resource limits. For example, *BWA ALN* and *BWA MEM* perform significantly worse on smaller instances and fail when using the r3.xlarge instance type. Thus, from the above results, *BWA ALN* should be executed on instances with at least 60 GB of memory. The tool also executes at least 38% faster on 16 core machines than on 8 core machines. Conversely, tools such as *FastQC* cannot take advantage of enhanced capabilities (e.g., more than one vCPU) and in fact should be deployed on the smallest available instance type, with its performance varying by only 10% across all instance types. Similarly *Bowtie* uses a maximum of 8GB of memory and therefore does not need instance types with large amounts of memory.



(a) FastQC (b) BWA MEM (c) Bowtie

(d) BWA ALN (e) MarkDups (f) ECDF of CPU Utilisation

Figure 6.5: The percentage of CPU utilisation for each instance type during tool execution (a) - (e). Empirical cumulative distribution function of CPU utilisation for each workload over the r3.8xlarge instance type.

6.3 Using Profiles in Globus Genomics

In collaboration with the Globus Galaxies team I am extending the Globus Galaxies platform to encode profiles and resource requirements for tools. This is beneficial for two reasons: 1) SCRIMP can employ these profiles to select the best instance type for a tool given current economic conditions, and 2) it will enable SCRIMP to schedule concurrent execution of tools that would individually not make full use of an instance's resources. In the second case, HTCondor can be used to partition instances into multiple *slots* – units of specific computational capacity. SCRIMP can enable this model by using profiles obtained from the profiling service to dynamically assign slots to tools based on requirements derived from their profile.

When executing workloads on the cloud, execution time is only one dimension by which to assess tool performance. Often, users are equally concerned with cost. To evaluate the cost and execution performance benefits that can be achieved by using profiles, I analysed four production Globus Genomics gateways to construct a representative workload. This workload contains over 2,000 executions of the five profiled tools over a period of 90 days. The cost of executing this workload is computed in the same way as is described in Section 5.3, where workload logs are processed to determine the time at which each job was submitted. Tool profiles are used to determine the execution time of each workload on each instance type. These execution times are then rounded up to the number of required billable hours for that instance type. The cost of using each instance type is then computed using historical AWS On-demand and Spot pricing information at time of each job's submission for its billable duration.

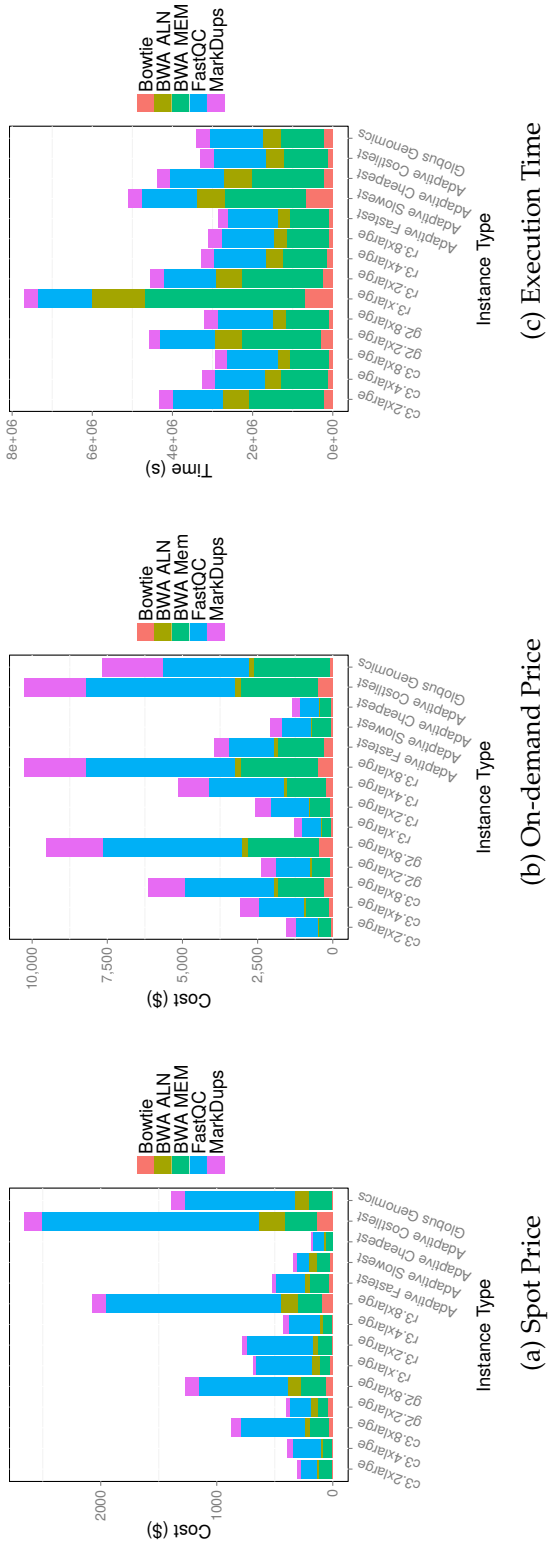
Figure 6.6 shows the total cost and execution time for the workload using each instance type. The figure also shows the execution time and cost that are obtained when using the Globus Genomics provisioning approach and several adaptive strategies that use profiles to guide provisioning decisions. The Globus Genomics provisioning approach applies a single static instance type for each gateway (in these gateways either m3.2xlarge and r3.8xlarge). The new adaptive strategies are designed to explore the trade-offs involved in selecting the fastest and slowest, vs. the cheapest and most expensive, instances. In these results the *BWA MEM* and *BWA ALN* tools are restricted to using instance types with at least 60GB of memory.

Figure 6.6a presents the total cost of executing the tools over each instance

type. As expected, using the larger instance types increase costs. Counterintuitively, the adaptive fastest solution, in which the fastest instance type for each tool is selected, is only marginally more expensive than the adaptive slowest approach – a result that can be attributed to the fact that the frequently executed *FastQC* tool has, on average (according to the above profiles), a slightly faster execution time on the pre-selected c3.4xlarge instances than on the more expensive 8xlarge instance type. The adaptive costliest approach presents the worst-case scenario for executing a workload based on Spot instance price. Its overall cost is greater than that of any individual instance type due to it selecting the most expensive instance type, regardless of tool, at the time of a job's submission.

These findings demonstrate the savings that can be achieved when tool profiles are considered in the provisioning process. As expected, the cheapest solution uses multiple instance types (each the most suitable for a specific tool at a specific time), as the projected cost is less than using any single instance type. Figure 6.6b shows the cost when using On-demand instances for each instance type and strategy. A substantial increase in cost is seen across all approaches relative to Spot prices. However, the general patterns are comparable.

Figure 6.6c shows the time required to compute the entire workload when using specific instance types and each adaptive solution. Clear differences can be seen in performance across instance types. For example, the execution time when using r3.xlarge is almost double that of other, larger instances. In contrast, execution time with standard Globus Genomics provisioning is not significantly larger than any single instance type; this is because individual gateways are often preconfigured with relatively large instance types (r3.8xlarge) and thus execution time is near optimal. The results also make clear the trade-offs between performance and cost. The cheapest solution reduces overall cost by 86.6% while increasing execution time by 29.2%, when compared to the Globus Genomics approach. The fastest solution reduces the monetary cost and execution time by 62.6% and 15.7%, respectively.



(a) Spot Price (b) On-demand Price (c) Execution Time

Figure 6.6: The cost and time of using each instance type. The four adaptive strategies (fastest, slowest, cheapest, costliest) model automated provisioning approaches in which an instance type is selected based on the price or projected tool execution time. The static *Globus Genomics* provisioning approach is not guided by tool profile information, but rather uses a pre-selected instance type for each gateway.

6.4 Summary

Globus Genomics exposes over one thousand custom tools across the various gateways. Manually determining the optimal cloud instance type for each tool is both labour intensive and error prone. A common problem shared between Globus Galaxies gateways, PDACS, and likely many other scientific services, is that users are encouraged to contribute their own analytical tools and share them with the community. It is unlikely that these tools are provided with high degrees of documentation regarding their execution, and are therefore difficult to intuitively associate with an optimal instance type.

This chapter presented an automated profiling service capable of creating rich profiles from arbitrary workloads. The profiling service manages the deployment and execution of tools across an elastic pool of provisioned and configured instances. It captures fine-grained performance statistics from the tool's execution and uses them to create profiles which can be used for instance provisioning, tool deployment, and scheduling. I have applied this approach to several genomics tools and shown that the resulting profiles can significantly improve performance and cost in real-world cloud provisioning scenarios. The findings of this chapter address the third and fourth research questions by automatically measuring the resource requirements of tools and incorporating them into the provisioning process.

Chapter 7

Provisioning as a Service

A common requirement of cloud-based applications, high performance computing algorithms, and data analytics is the need to scale across many cloud instances. To address this requirement a number of auto-scaling systems have been developed, such as Fenzo [88], Cloud Scheduler [90], and Kingfisher [96], to name a few. However, in most cases these systems are designed for a specific application and are not easily repurposed by others, many require manual configuration to optimise efficiency, and most are not able to dynamically assess cloud marketplaces to cost-effectively provision infrastructure. Without such sophistication, it is likely that there are many cloud deployments that are inefficient with respect to performance and cost.

The complex landscape of instance types and economic models creates new opportunities to optimise many aspects of the provisioning process. Cost, execution time (for jobs and related jobs), data locations, instance types, and instance terminations offer many potential trade-offs when determining the most appropriate cloud instance to provision for a given workload. For example, although the Spot market can substantially reduce operational costs it can also compromise a service's ability to reliably fulfil workloads. This trade-off prompted the fifth research question, **RQ5**, which asks how reliable clusters can be achieved when using potentially unreliable infrastructures. I explore this by investigating the use of predictive bid prices and the affect they can have on instance reliability.

This chapter presents the Scalable Cloud Resource Management and Provisioning service (SCRIMP). SCRIMP is an automated and cost-aware provisioning service for efficiently acquiring and managing elastic cloud infrastructure to execute arbitrary workloads. SCRIMP combines many of the techniques and tech-

nologies described thus far in this thesis. SCRIMP is used in production by over 30 different gateways and more than 300 researchers. It has been shown to reduce cost, execution time, and instance termination rate.In response to the growing needs for similar capabilities in other gateways and scientific applications, SCRIMP is designed with an independent, generalised, and modular architecture that can be leveraged via its APIs to efficiently provision cloud infrastructure on behalf of external applications. In this chapter I employ SCRIMP to explore the final research question, **RQ6**, by investigating how fine-grained accounting information can be maintained and used for auditing and cost reclamation.

7.1 SCRIMP

SCRIMP is designed with a modular architecture in which different execution frameworks, cloud platforms, provisioning strategies, and optimisation methods can be integrated. An overview of the SCRIMP architecture is depicted in Figure 7.1. SCRIMP also provides tune-able *knobs* for many features (run rate, thresholds, bid price, etc.), enabling customisation and optimisation of different aspects of the provisioning process. SCRIMP is delivered as a cloud-hosted service through which users may customise the deployment to the needs of their application and manage on-demand infrastructure via RESTful APIs. This chapter presents the architecture of SCRIMP and investigates the performance and cost benefits that can be obtained via execution of production workloads. I describe integrations of both HTCondor [82] and Apache Spark [84] which support provisioning of dynamic condor pools and on-demand Mesos [89] clusters, respectively.

SCRIMP is architected as a stand-alone service that can support a broad range of execution environments. As a service, users are able to configure and manage every aspect of the provisioning approach independent from a particular application or gateway. As a single organisation may operate multiple execution environments (each environment is referred to as a *tenant*), SCRIMP is able to manage multiple tenants simultaneously and can improve global efficiency by enabling the cooperative sharing of execution infrastructure.

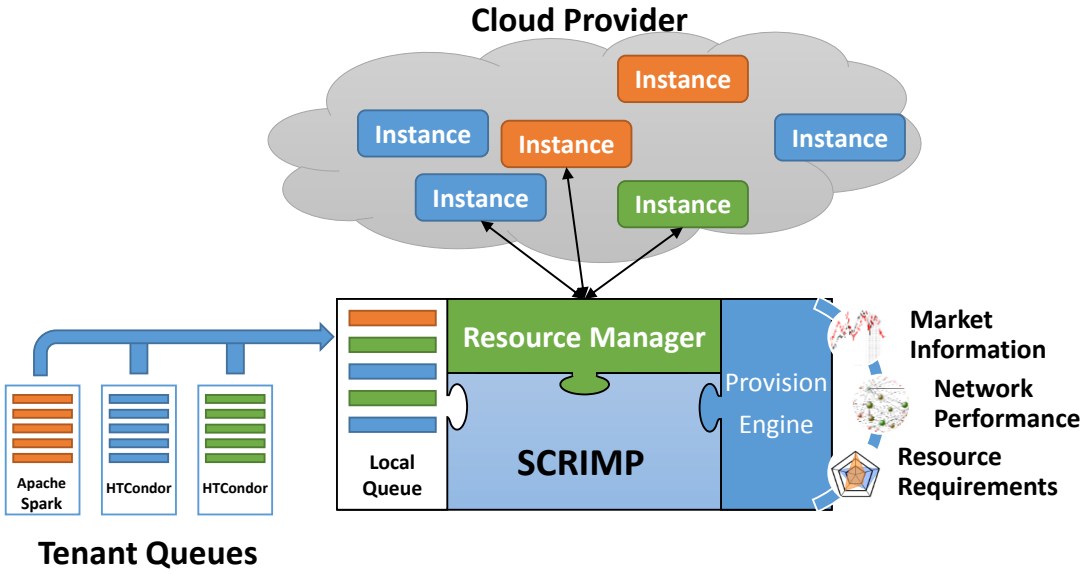


Figure 7.1: An overview of SCRIMP.

7.1.1 Execution Frameworks

SCRIMP is designed to meet a broad range of use cases and in turn supports a range of execution frameworks. To facilitate this flexibility, SCRIMP is designed to monitor arbitrary execution queues to retrieve real-time workload information. It currently incorporates two *scheduler modules*, or implementations of a queue interface, and can be easily extended. These modules leverage HTCondor and Apache Spark APIs to retrieve the list of jobs awaiting execution, information about the requirements of each job, and other queue-level metadata (e.g., wait time). These properties are harmonised into a common data model that is used by SCRIMP to make provisioning decisions.

HTCondor

HTCondor [82] is a commonly used high throughput computing framework that is built upon a master/worker model. Jobs are submitted to a master HTCondor queue. HTCondor workers are registered with a single master node. When workers are available the master's *negotiator* matches idle jobs from the queue to appropriate workers. ClassAds [184] describe the requirements and dependencies of a particular job. This model enables resource requirements, such as number of cores, memory, and disk to be expressed for every job. HTCondor is

used by the Globus Galaxies platform (and many other scientific gateways) as a scalable model for executing workflows.

Hierarchical HTCondor pools (collections of worker nodes) can be established by having several HTCondor queues report their job ClassAds to other queues. This is often used in multi-tenant deployments in which a single organisation may have many condor pools for different purposes and may wish to exchange workload between these pools when excess capacity is available. Such models can be established by linking HTCondor *collectors*, which are responsible for collecting the state (daemons, resources, etc.) of the HTCondor pool. Individual tenants are then configured to enable passing of ClassAds between collectors, which allows all jobs to be monitored and queried by the highest level collector (SCRIMP's collector).

Apache Spark

Apache Spark is a framework for for supporting parallel data-intensive applications on commodity clusters. Spark is built on top of Apache Mesos, a cluster management system that allows parallel applications to be executed on a cluster. Spark can also use other cluster management systems such as Hadoop YARN [194]. It can also operate on top of a wide variety of storage systems such as Hadoop Distributed File System (HDFS) [195].

Spark uses a task per data partition to represent a parallel operation on a dataset. Each task can be executed in parallel on one or more worker nodes. Mesos provides a machine independent representation of a distributed system. It is built upon a master/slave model in which tasks are submitted to the master and are then executed on various slaves. Spark can operate with Mesos in coarse-grained and fine-grained models. In the coarse-grained model a single long-running Spark task is executed on each Mesos worker. The Spark task is therefore responsible for scheduling sub tasks on that worker. In the fine-grained model each Spark task is executed as a Mesos task, therefore allowing workers to host multiple Spark tasks. Mesos includes a resource advertising model from which the master is able to determine the available capacity of worker nodes for submission.

7.1.2 Instance provisioning and configuration

SCRIMP relies on cloud platform APIs to request and configure instances used to execute workloads. The service is designed with a plug-in *cloud module* interface which must support several simple operations:

1. Provision an instance with particular resource capabilities.
2. Configure the instance with required software and data dependencies.
3. Request information about the running instance (e.g., IP, health check, running time).
4. Manage the instance via a unique reference (e.g., instance termination).

SCRIMP currently supports an EC2 *cloud module* that uses EC2 REST APIs to perform these operations. An alternative cloud module is also provided which simulates AWS functionality, allowing the exploration of provisioning features for little to no cost.

Irrespective of the execution framework, cloud infrastructure, and application requirements every provisioned instance must be configured to enable execution of the associated workload. In the case of HTCondor this is an HTCondor worker node, for Spark it is a Mesos slave. Moreover, application requirements (e.g., application binaries and dependencies) must be available on the provisioned instance in order to execute the job. In some cases, such as Globus Galaxies, several other dependencies are required such as a shared file system to access reference and input data.

SCRIMP relies on the cloud-init model to dynamically configure instances with software dependencies and/or settings. To expedite the configuration process, SCRIMP supports the use of predefined virtual machine images (e.g., an Amazon Machine Images (AMI)) that include a static image of pre-installed software. Typically these images include software such as the worker execution software (e.g., Spark worker node), but they may also include a suite of other applications. SCRIMP currently uses dynamic contextualisation of raw images when fulfilling HTCondor workloads and an AMI for Spark workloads as building and deploying Spark can be particularly time consuming. After provisioning either a raw image or AMI, SCRIMP uses cloud-init to configure the instance. This configuration includes registering the worker node with the master and optionally

linking the node with other shared infrastructure (e.g., NFS). SCRIMP also allows custom cloud-init scripts to be defined on a per-tenant basis.

7.1.3 Cloud Provisioning

In large-scale application scenarios there is infrequently sufficient a priori information to optimise execution. Workloads are dynamic, whereby two sequential workloads can have very different execution characteristics. There are a huge range of complex and dynamic trade-offs that can be considered at any time, for example provisioning decisions may be optimised for execution cost, execution time, data transfer cost, reliability, resource utilisation, to name a few.

Given the search space and complex optimisation decisions it can be difficult for a user to efficiently provision resources [22]. As shown in Chapter 5, simplistic provisioning approaches (such as using a single Spot instance type) can increase cost by an order of magnitude and degrade performance. Many real-world automated provisioning systems employ static instance selection strategies [196, 197]. These systems present an enormous potential for optimisation and are a key motivation for the development of SCRIMP.

There is a rich amount of information that can inform the provisioning process. Real-time instance pricing and usage information can be retrieved, application performance can be profiled and derived from execution traces, and dependencies and resource requirements can be derived from job information. All of this information can then be factored into predictions and provisioning decisions.

SCRIMP incorporates the cost-aware techniques discussed in Chapter 5 to minimise the cost of acquiring and operating cloud infrastructure. SCRIMP is also compatible with the work presented in Chapter 6, enabling custom workload profiles to be created and used during the provisioning process. These capabilities give SCRIMP the potential to reduce costs and optimise performance for many scientific services.

Predictive Bidding

Dynamic pricing models provide opportunities to provision infrastructure for reduced cost in exchange for reduced availability and reliability. However, a lack of knowledge of Spot market volatility may lead to increased costs and instance terminations [21, 22]. If future Spot prices can be accurately predicted, users can

customise the provisioning process to select more reliable Spot instances to minimise costs and terminations.

SCRIMP includes a modular instance selection interface that can be extended to implement more advanced selection and bid calculation algorithms. The default selection algorithm can be configured to assess On-demand and Spot markets and sort potential requests by the cheapest instance type and availability zone. The default bid calculation algorithm can be configured to bid either a static value or a percentage of the On-demand price.

To exemplify the advantages of this approach and explore the potential optimisations that can be achieved with Spot market forecasts, I, in collaboration with researchers from the University of California at Santa Barbara [39], have integrated support for the DrAFTS (Durability Agreements From Time Series) system [29] into SCRIMP. DrAFTS is designed to compute a bid value that guarantees (with a given probability) that an instance lifetime will exceed a specified time (i.e., that the market price of the selected instance will not surpass the bid price). DrAFTS computes this bid based on time series analysis of historical Spot prices for all instance types and availability zones. SCRIMP employs DrAFTS by querying the predicted bid prices for all suitable instance types and availability zones. SCRIMP then ranks these bid prices, in the same way that it ranks lowest cost instances when using current Spot prices, when selecting an instance to provision.

7.1.4 Resource Management

SCRIMP goes beyond the acquisition of cloud infrastructure and manages instances from the submission of requests through to their termination. Examples of resource management activities include cancelling or repurposing unused instance requests, restarting jobs if instances are terminated, terminating instances if they are not used, and migrating instances between tenants.

The resource management aspects of SCRIMP take advantage of cloud computing models to improve performance and cost. For example, the time to satisfy a Spot instance request varies and can take several minutes. However, users are not charged for instance requests that are not satisfied and these requests can be cancelled at any time. Thus, SCRIMP can over-request instances (typically of different types or in different availability zones) and monitor the requests as a group until one is satisfied. SCRIMP will then cancel the outstanding requests, unless

there are other waiting jobs, in which case those requests will be repurposed to other jobs. SCRIMP also facilitates the migration of cloud instances between cooperative tenants through the development of a novel migration framework, discussed in Section 7.3.3.

Given the unreliable nature of Spot instances it is important that SCRIMP monitors running instances and responds to terminations. In this case, jobs are returned to the job queue and a new instance will be provisioned. Another cloud-based optimisation that SCRIMP can apply relates to fully utilising the billable period of EC2 instances. EC2 instances are billed on an hourly basis. Thus, there is no benefit in terminating an instance within an hour (from a cost perspective). With this knowledge SCRIMP only terminates instances as they approach the end of the hourly billing cycle. This ensures that idle instances are kept alive and can therefore be used satisfy incoming requests immediately.

SCRIMP records all cloud interactions and associates decisions with specific workloads. A reliable RDS database stores important instance details (type, availability zone, price, etc.), event times (request, launch, termination, and migration), as well as many other details regarding job, workload, and provisioner state. This allows SCRIMP to report fine-grained usage and accounting statistics for tenants and their individual users. In addition, the cost of each instance is calculated and recorded. Thus, SCRIMP can expose detailed billing information regarding a tenant and enable administrators to analyse usage characteristics to optimise the provisioning process.

7.1.5 Provisioning algorithm

The provisioning algorithm used by SCRIMP is summarised in Algorithm 2. It is specifically designed with a number of configurable parameters which can be defined at the provisioning service level or for specific tenants. The service periodically checks the status of the job queue based on a configurable *run rate* parameter. When the provisioning service checks the global queue it identifies waiting jobs and matches them to a specific tenant. The provisioning algorithm has two primary phases. The first phase involves the management of existing infrastructure (lines 9–26). This phase first checks the state of instance requests to identify requests that have been fulfilled and either cancel or repurpose unnecessary requests (e.g., a previously idle job may have been deployed to a newly idle instance). Provisioned instances are also monitored for state changes to de-

tect terminations and terminate idle instances when necessary. Another aspect of the management phase is to migrate instances to a cooperative tenant if one tenant has excess idle instances and another has idle jobs. The second phase of the provisioning algorithm focuses on acquiring additional instances (lines 28–34). Idle jobs are checked against the RDS database to determine whether a new request is necessary (based on existing requests and tenant settings). If a request is required, job profiles are used to filter the available instance types based on their CPU, disk, and memory requirements. Instance types and their potential availability zones are then ranked by price before SCRIMP makes requests for the selected instances.

7.2 Experimental Dataset

To assemble a representative multi-tenant workload to evaluate the performance of SCRIMP I analysed the execution traces of five production Globus Genomics gateways. Specifically, I selected the busiest day from each of the gateways to construct a single dataset with a broad range of job executions. Each job's submission time, execution time, and instance requirements (e.g., required number of vCPUs specified by the gateway) are included in the dataset. The dataset is comprised of 8452 jobs over a 24 hour period. In order to reduce the execution time and cost of running experiments on a commercial cloud only the first 1000 jobs of the dataset are used. This represents a three hour and twenty minute period of job submissions, for a total of approximately eight hours of execution and 537 cumulative compute hours.

To enable the dataset to be *replayed* at different times the submission time of each job has been transformed into a relative submission time, offset by the time of the day the job was submitted. This allows the dataset to be deployed at different times while still reflecting the production distribution of jobs. The execution of the jobs can be seen in Figure 7.2. This figure shows the rate at which the first 1000 jobs are dispatched. The width of each job indicates execution duration. The ECDF of the jobs shows that the distribution of jobs is approximately normal, as shown in Figure 7.3

Algorithm 2 SCRIMP provisioning process.

```
1: runRate = /* configurable */
2: idleReq = /* configurable */
3: while true do
4:    tenantList = subscribed tenants
5:    for tenant in tenantList do
6:       jobList = idle jobs in queue
7:       requestList = EC2 instance requests
8:       # Manage acquired resources #
9:       for request in requestList do
10:         for job in jobList do
11:            if request belongs to job and request is fulfilled then
12:               Remove job from jobList
13:            end if
14:         end for
15:         if request does not belong to any job then
16:            Re-purpose or cancel request
17:         end if
18:      end for
19:      # Migrate excess resources #
20:      if jobList == empty AND instanceList != empty then
21:         for tenant in tenantList do
22:            if instanceList < jobList then
23:               Migrate an instance to an idle job
24:            end if
25:         end for
26:      end if
27:      # Provision additional resources #
28:      for job in jobList do
29:         if job not fulfilled AND job idle time > idleReq then
30:            eligibleIns = restrict instance types by job requirements
31:            instanceRequest = select best type from eligibleIns
32:            Request instanceRequest
33:         end if
34:      end for
35:   end for
36:   SLEEP runRate
37: end while
```

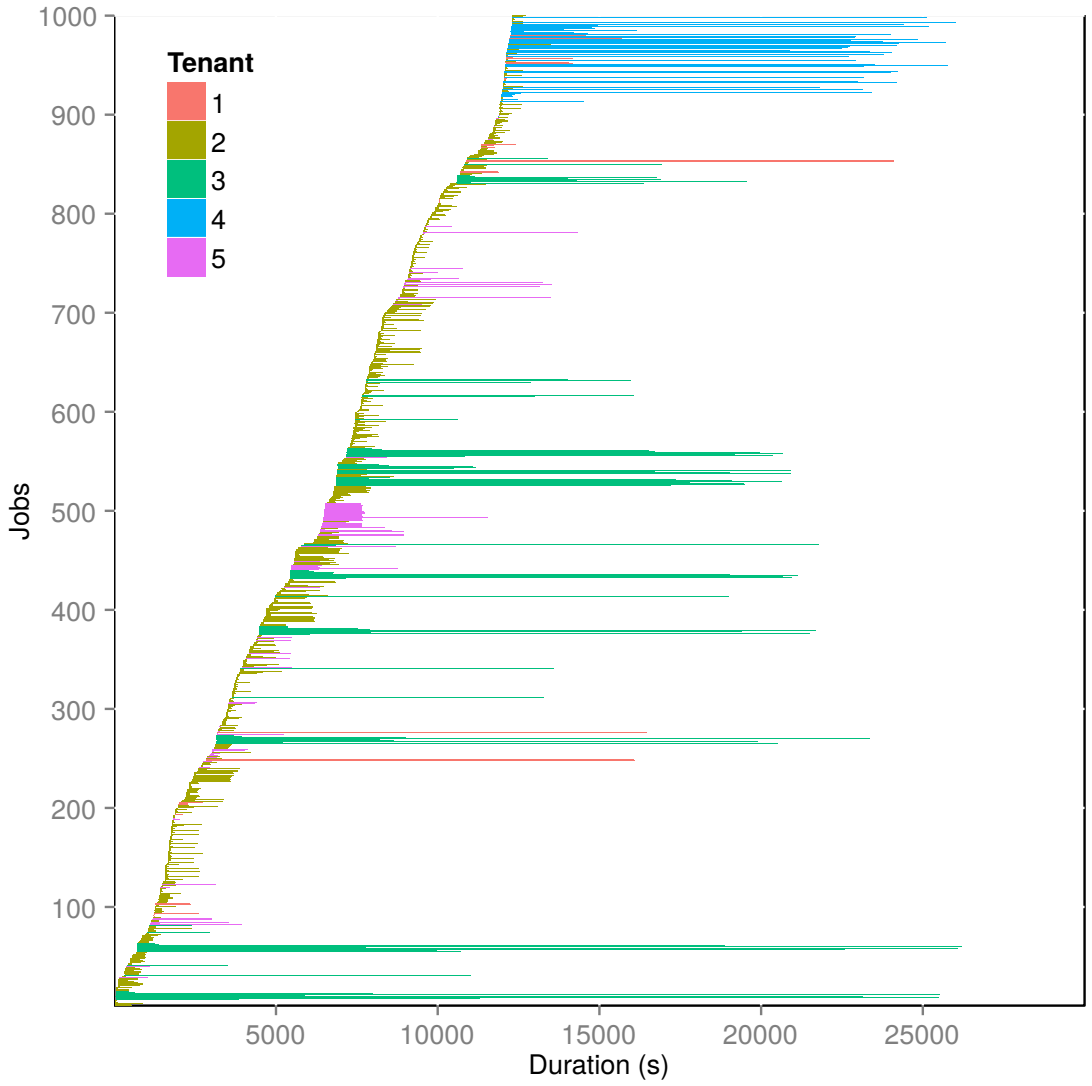


Figure 7.2: The execution time and duration of the first 1000 jobs of the dataset.

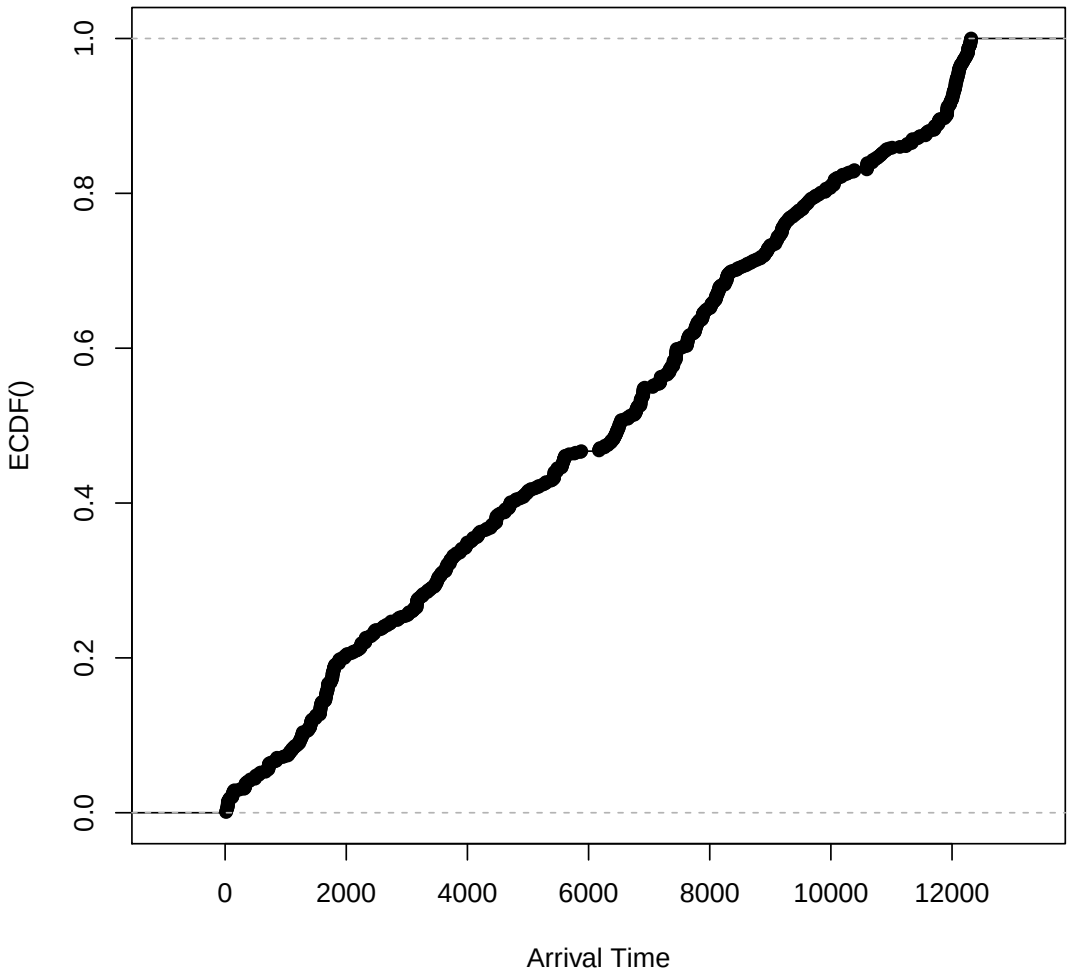


Figure 7.3: The ECDF of the first 1000 jobs relative execution time.

Many of the jobs included in this workload represent executions of proprietary tools and analysis of real datasets. Thus, executing these workloads requires both installation of these tools and access to the input datasets. To alleviate the need to manage a production execution environment, instead the execution of individual jobs is modelled by a sleep command with a simulated execution length. The duration of the sleep command is determined by the recorded execution length of the job in the original execution log. However, one difficulty with this approach is that the provisioning model supports multiple instance types whereas the job was originally executed on a single instance type. As such, the execution time for each job is likely to be very different on different instance types. To address this discrepancy the findings from Chapter 6 are used to forecast and transform the execution duration between instance types. Execution ratios are used to scale the execution duration from the recorded execution's instance type to the simulated instance type. Table 7.1 presents a summary of the execution ratios between several common instance types and the m3.2xlarge and r3.8xlarge instance types.

Table 7.1: Ratio of execution times between instance types.

Instance	m3.2xlarge	r3.8xlarge
m3.2xlarge	1.00	1.84
c3.2xlarge	0.98	1.80
c3.4xlarge	0.61	1.12
c3.8xlarge	0.51	0.94
g2.2xlarge	1.05	1.93
g2.8xlarge	0.56	1.03
r3.2xlarge	1.01	1.86
r3.4xlarge	0.64	1.18
r3.8xlarge	0.54	1.00

7.3 Evaluation

Evaluating the performance of SCRIMP is challenging for a number of reasons: firstly, the production usage of SCRIMP cannot be easily replicated due to significant costs; secondly, workloads are not always repeatable; and thirdly, jobs are

run at different times, with different cloud market conditions. A series of emulated experiments have been conducted to address these challenges. The emulated experiments utilise the reproducible workload dataset and the AWS cloud module to deploy sleep jobs to real AWS instances. To mitigate the changing market condition's influence over tests, a simulation plug-in is used to verify the results of the emulation tests. The simulation plug-in is able to simulate the AWS market at a given time and can therefore be used in place of the cloud module to compare different strategies under the same market conditions.

The experiments focus specifically on the affect of various optimisations and configurations that can be applied to SCRIMP, such as queue polling, idle times, and request repurposing. In each case the experiment process is the same. The workload described in Section 7.2 is dispatched to an HTCondor queue and the management component of SCRIMP records how each job is assigned to instances. Properties such as instance requests, instance terminations, individual job wait and run time, instance cost, and makespan are also captured and compared.

Various bidding strategies are employed in each experiment to explore the affect of static and predictive models:

- **Static**: Selects the cheapest instance at the time of job submission and uses a configurable bid price relative to the On-demand instance price. E.g., 0.8 $\times$ On-demand price.

- **Predictive**: Uses DrAFTS to compute a bid that will ensure that an instance remains active for at least one hour with 99% confidence. Selects the instance with the lowest such bid.

- **Profiles**: Uses DrAFTS to compute a bid that will ensure that an instance remains active for the duration of the job with 99% confidence. Selects the instance with the lowest such bid.

Finally, the potential benefit of facilitating the migration of instances between tenants is explored by replaying production execution traces.

7.3.1 Emulation

Emulation tests have been conducted to explore SCRIMP's tunable configuration options. It is important to note that each experiment has been run at a different time and is therefore subject to dynamic market conditions. Table 7.2 shows the

averaged results (from four executions) of the tests that have been performed for each configuration option. The table's *description* column describes the configuration parameter that is explored in each test, where *IR* (Idle Requirement) is the length of time a job must be idle before being processed, *Re* (Repurposing) is whether or not orphaned requests are reused and moved to other idle jobs once a job becomes fulfilled, *RR* (Run Rate) is the amount of time the SCRIMP waits between monitoring periods, and *AZ* (Availability Zone) is the number of availability zones the service is configured to use. The *base* test cast employs a run rate of five seconds, an idle requirement of two minutes, enables request repurposing, and can select from nine instance types (those presented in Table 7.1 across all availability zones.

Table 7.2: The performance of SCRIMP with different configuration settings.

Description	Avg Wait (s)	Requests	Instances	Spot Cost	On-demand Cost
Base	115.71	354	313	$83.17	$534.06
AZ=1	101.91	355	345	$77.36	$547.03
Re=F	224.77	563	411	$79.26	$578.81
RR=120	136.63	355	355	$84.05	$668.73
RR=600	251.21	291	291	$91.10	$541.44
IR=0	73.50	766	467	$82.36	$618.88
IR=600	376.05	359	354	$76.17	$538.75

The configurable parameters have considerable influence over the performance of SCRIMP and enable users to manage trade-offs between performance and cost. The results of the tests are further shown in Figure 7.4. The following examines these results in further detail.

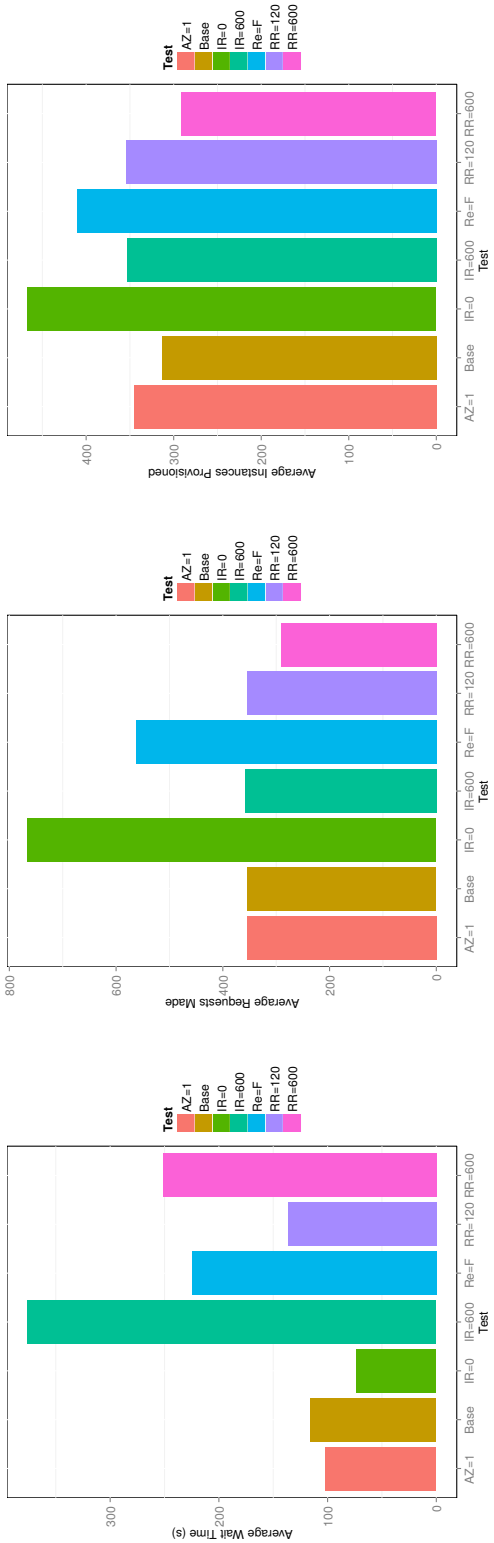
Run Rate

The run rate parameter determines SCRIMP's monitoring period for polling the queue data and instance resource information. The base case employs a run rate value of 5 seconds. Two additional test cases have been used to investigate the affect of delaying SCRIMP's run rate to once every 120 seconds and once every 600 seconds. Both of these test cases result in fewer instance requests, as shown in Figure 7.4b, and fewer provisioned instances (Figure 7.4c) than the base test case. The delayed monitoring period means SCRIMP detects idle jobs less frequently.

This provides the HTCondor negotiator a greater opportunity to schedule jobs to existing instances before SCRIMP detects them, thus resulting in fewer instances being launched. However, acquiring fewer instances results in jobs having to wait for existing instances to become idle and therefore increases the average wait time for both of these tests, as shown in Figure 7.4a. The tests show the average wait time for the 120 and 600 second run rate tests increases 18% and 117%, respectively, over the base case. Finally, a less frequent run rate means SCRIMP is unable to terminate unnecessary requests before they are fulfilled by AWS. This is reflected by the number of requests matching the number of provisioned instances for both of these tests.

Idle Time Requirement

SCRIMP enables configuration of the length of time a job must wait before it is identified as requiring resources. This configuration is designed to allow the HTCondor negotiator time to execute and therefore reuse of existing instances. Each of the emulation tests are executed with HTCondor configured to use the highest possible negotiation rate (20 seconds). The base test case employs a two minute idle time requirement to reduce the number of unnecessary instances that are requested. Two additional test cases have been explored to determine the role of the idle requirement. These execute SCRIMP with no idle requirement and a 600 second idle requirement. When no idle requirement is specified, jobs are detected when submitted and will have an instance provisioned immediately. Removing the idle requirement results in an almost 100% increase in the number of instance requests, and due to the AWS fulfilment rate, an average of 141 more instances provisioned than the base test case. Lengthening the idle time requirement to 600 seconds provides the opportunity for workloads to reuse existing infrastructure rather than launch new instances. It therefore results in far fewer requests being made. However, the greater idle requirement triples the average wait time for jobs, when compared to the base test case.



(a) Wait time. (b) Instance requests. (c) Instances provisioned. (d) Spot cost. (e) On-demand cost.

Figure 7.4: The performance of SCRIMP with various configurations.

Resource Restriction

Tenants can specify the instance types and availability zones that SCRIMP can select. A test has been conducted to investigate the influence of limiting the potential resource pool to a single availability zone. The results of this test show very little effect on SCRIMP's performance, in fact, in some of the cases the restricted executions actually performed better than the base case. These results confirm the findings presented in Chapter 6, where less than a 1% difference was found between provisioning resources across all possible availability zones and a single availability zone.

Repurposing Requests

Instance requests can often become orphaned if the job for which they were requested is scheduled to an existing instance. Rather than cancelling these requests, SCRIMP attempts to capitalise on the situation by repurposing orphaned requests to other idle jobs. The base test case includes request reuse, whereas the *Re=F* test case does not. The test case without request reuse terminates excess instance requests when a job is removed from the idle queue or when an instance request is fulfilled for it. The results show that disabling request reuse increases both the number of instance requests and provisioned instances.

Disabling request repurposing results in an increase in provisioned instances of 31.1%. Counterintuitively, the Spot instance cost, shown in Figure 7.4d, is lower than the base test case. This is due to the dynamic Spot market conditions at the time of the tests' execution. This conclusion is supported by the On-demand cost (Figure 7.4e) reflecting the higher number of provisioned instances. Additionally, the average wait time increases by 94.2% over the base test due to new requests having to be made for each idle job.

Cost & Performance

SCRIMP allows tenants to describe custom provisioning strategies to trade-off performance and cost. For example, if cost is not prohibitive, the throughput of a tenant can be improved by electing to use On-demand instances. In addition to utilising On-demand instances, SCRIMP's extensible provisioning strategies provide a range of opportunities for cost and performance customisation. The dynamic AWS environment makes it difficult to reliably compare the performance

and cost of subsequent tests. Therefore, makespan and Spot instance cost are not reliable performance and cost indicators. Instead, average wait time and On-demand cost (which is not subject to dynamic pricing) can be used to compare test results.

An example of SCRIMP's ability to trade performance and cost is demonstrated by the idle requirement test cases. These test cases show that reducing the idle requirement can substantially improve the throughput (reducing average wait time) of the service in exchange for an additional $84.82 On-demand cost.

Emulation Validation

In order to validate the emulation the simulation cloud module plug-in is used to remove market variability by allowing workloads to be replayed at the same time under the same market conditions. A comparison of executing SCRIMP with and without the simulation plug-in is given in Table 7.3. The results indicate the simulation plug-in creates 11% fewer instance requests and acquires 17% fewer instances than using the AWS cloud module to fulfil a 100 job dataset. There are two main causes for these discrepancies: firstly, the simulation plug-in uses average observed values for the time required to request, launch, and configure instances, whereas the AWS-based tests experience variable times for each. Secondly, HTCondor does not always negotiate jobs at the specified rate which makes it difficult to accurately simulate its behaviour. However, using the simulation plug-in to execute the 100 job dataset exhibits a standard deviation of 1.70 provisioned instances over seven executions. Thus, the simulation cloud module plug-in is able to produce relatively consistent results.

Table 7.3: Calibration of the simulation plug-in.

	Real		Simulated	
Jobs	Requests	Instances	Requests	Instances
1	1	1	1	1
5	5	5	5	5
10	10	10	10	10
25	11	10	16	14
50	21	21	17	16
100	45	42	40	35

The results of using the simulation cloud module plug-in to perform the previously emulated test cases are given in Table 7.4. Despite the results not matching the emulation tests exactly, they do support the emulated findings. For example removing the idle requirement in the emulated experiments resulted in 413 additional requests being made, whereas simulating the same test case resulted in an additional 440 requests. Although many of the simulation results are less similar (e.g., the improvement in average wait time for the same test was 36.0% when emulated and 15.76% when simulated), the overall results show the configurable parameters to have similar effects to the emulated tests. Therefore the simulation cloud module plug-in corroborates the findings of the emulated test cases.

Table 7.4: Simulation results.

Description	Requests	Instances	Average Wait (s)
Base	313	274	247.84
AZ=1	325	291	146.38
Re=F	392	309	317.23
RR=120	330	330	279.05
RR=600	229	229	344.68
IR=0	753	567	208.79
IR=600	231	231	365.94

7.3.2 Spot Instance Reliability

To explore the importance of Spot market bid values I have integrated and emulated multiple bidding strategies. To minimise the impact of performing tests at different times the resource management component of SCRIMP has been configured to capture the decisions of multiple bidding strategies concurrently. That is, when provisioning a new instance for an idle job each bidding strategy is executed to nominate an instance type, availability zone, and bid value, which is then recorded. For each test case only one of the bidding strategies is used to launch a new instance. The other strategies simply nominate an instance and record their decisions for later analysis.

The bidding strategies analysed with the 1000 job dataset are: *DrAFTS Cheapest*, *DrAFTS Average*, *DrAFTS Profiles*, and *Cost-aware*. The *DrAFTS Cheapest* approach queries the DrAFTS service to determine the bid price for each instance

type and availability zone assuming the instance will be required for one hour. The cheapest of these instances is then selected and its corresponding bid value is used. The *DrAFTS Average* strategy uses DrAFTS's forecast average cost of each instance over the next hour to select the instance with the lowest predicted volatility in that hour. The DrAFTS one hour predictive bid is used, however the selected instance is not necessarily the same. The *DrAFTS Profiles* strategy employs the DrAFTS service to select the cheapest instance type based on the known execution length of a job. This strategy uses profiles obtained from the profiling service, discussed in Chapter 6, the execution length of a job (scaled to each instance type) is used to select a DrAFTS bid with a 99% guarantee that the instance will persist throughout the job's execution. These bids are then sorted and the instance type with the lowest price is selected. The *Cost-aware* strategy employs the cost-based selection techniques described in Chapter 5. It selects the cheapest instance type at the time of provisioning. A simplistic bidding approach is used where a bid is determined as 80% of the instance type's On-demand price.

The experiment required 348 instances to be provisioned to satisfy the 1000 job dataset. The four bidding strategies are compared in terms of the dollar value, potential cost, financial risk, and the number of terminations their provisioning decisions would have been subject to. The potential cost is computed as the maximum value the provisioned instances could have incurred had the user been charged the bid price. Financial risk is computed as the potential cost minus the actual cost. Terminations are determined by identifying if an instance's bid price would have been exceeded by, or equal to, the Spot market price at any time during the job's execution.

The results of the experiment are presented in Table 7.5. The table shows that using predictive bidding strategies can substantially lower financial risk, with the Cost-aware bidding strategy having over 20 times the financial risk of the DrAFTS strategies. Incorporating the execution length into the bidding strategy (DrAFTS Profiles) reduces the number of Spot instance terminations from 44 to 32. There were no identified instance terminations when using the Cost-aware approach due to its use of high bid prices and relatively stable instance prices at the time of experimentation. However, the cost-aware approach presents a significantly larger financial risk.

An initial assumption supposed that EC2 instances were billed in a cumulative manner, incorporating the moving Spot market price into the instance's cost. However, it was found that EC2 instances are billed at the Spot market price at the

time the request is fulfilled and then charged the market price at the beginning of each subsequent billable hour. Therefore, the DrAFTS Average strategy was not as effective as anticipated. In fact, the results demonstrate that in these market conditions simply selecting the cheapest instance at the time of provisioning is the most cost-effective approach, with an almost 10% reduction in cost when compared to DrAFTS strategies.

DrAFTS is an ongoing research project and is not yet deployed as a production service. When conducting these experiments it was discovered that acquiring complete DrAFTS predictions for an individual job required almost ten seconds. This overhead would cumulatively grow as additional jobs were serviced, resulting in an abnormally high number of provisioned instances and excessive job wait times. To minimise this overhead, and make the results comparable to other SCRIMP experiments, a copy of the DrAFTS data was stored in a local database during testing.

Table 7.5: The result of using different bidding strategies.

Strategy	Cost	Potential Cost	Financial Risk	Terminations
DrAFTS Cheapest	$95.53	$110.77	$15.24	42
DrAFTS Average	$95.99	$111.76	$15.77	44
DrAFTS Profiles	$92.94	$117.04	$24.10	33
Cost-aware	$84.31	$418.39	$334.08	0

7.3.3 Migration

Migration allows cooperative tenants (e.g., those run by the same organisation) to exchange compute resources in order to improve global efficiency. SCRIMP is designed to detect migration opportunities (such as an excess of idle resources) and initiate the transfer of resources between two tenant's HTCondor pools.

The process by which instances are migrated has been developed in collaboration with researchers at Argonne National Laboratory and the University of Wisconsin–Madison. SCRIMP enables migration by configuring each provisioned instance with a RESTful Web service as it is deployed. SCRIMP initiates migration by first identifying a suitable candidate instance to move between two tenants. SCRIMP then communicates with the instance's Web service and instructs it to reconfigure its HTCondor worker. The Web service instructs the HTCondor

worker to advertise that it is peacefully leaving the current pool, meaning the worker will leave once idle. Once the instance has been removed from the pool, the Web service initiates a reconfiguration process on the worker's HTCondor installation. Reconfiguration provides a unique opportunity to dynamically reassign the properties of the worker's HTCondor slot (which can only be achieved when a worker is not in a pool) such that the worker can be tailored to the requirements of the target tenant. During the reconfiguration process the Web service overwrites the *condor_host* address of the worker and restarts HTCondor. When the resource restarts it attempts to communicate with the new HTCondor master to join the new pool. The process of leaving a pool, reconfiguring and restarting a worker, and joining a new pool takes less than one minute. This is substantially faster than the approximately ten minutes [31] required to create a Spot request, fulfil the request, boot the instance, and configure the worker to join a pool.

An experimental platform was developed to investigate the effectiveness of migration. The platform consistently replays execution traces by using SCRIMP and a modified version of the simulation cloud module plug-in. The modified simulation cloud module does not enforce time constraints (such as instances starting and jobs executing). Instead, it steps through time in five second increments and evaluates and updates the state of the environment (tenants, HTCondor, and AWS). This means tests can be rapidly deployed and evaluated in a short period of time. The execution traces of five Globus Genomics gateways have been used to study migration. The execution traces span a 99 day period (Jan 4th 2015 to April 14th 2015) and include 6640 distinct jobs.

Migration Simulation Calibration

Calibration tests have been performed on the experimental platform to ensure its behaviour is representative of a real deployment. A basis for comparison has been constructed by analysing the execution traces. The traces contain information regarding each job, including an identifier of the instance the job was dispatched to, the queue time, start time, and execution length. Counting the number of unique host identifiers provides an estimate of the number of instances that were used to execute the jobs. However, host names can be reused. To mitigate this problem the definition of what constitutes a unique host has been limited to identifiers that have not been used by another job within the previous eight hours. If the same host name is used by two jobs within an eight hour period, they are

assumed to be an individual instance. This analysis resulted in the identification of 2731 distinct instances collectively used by the tenants. The average wait time of each job has been computed by subtracting the job's start time from the job's queue arrival time. Doing so identifies an average wait time of 347 seconds across the 6640 jobs.

Table 7.6 shows a comparison of the analysed execution traces and the result of replaying the 6640 job dataset. The table shows that the average wait time is 98.54% accurate when simulating the execution traces with the platform. However, replaying the traces results in 12.12% more instances than were identified from analysis of the execution traces. It should be noted that a slight discrepancy between the number of recorded and simulated instances was expected. The exact number of instances launched to perform the 6640 job dataset is unknown. The execution traces do not capture instances which are provisioned but not used by any jobs. These results confirm the accuracy of the platform and support its use as a tool to evaluate the performance of the migration strategy.

Table 7.6: Calibration of the migration simulation.

	Instances		Avg Wait	
Tenant	Recorded	Simulated	Recorded	Simulated
1	364	430	302	308
2	14	12	370	306
3	62	67	276	242
4	319	292	449	407
5	1972	2261	348	347
Total	2731	3062	347	342

Migration Evaluation

The 6640 job dataset has been replayed through the platform twice, once with migration enabled and once with it disabled. In these tests the platform is responsible for identifying opportunities to migrate an instance between two tenants. The platform employs a *push* technique to achieve migration. That is, for each time step it checks each tenant to identify idle instances. It then checks other tenants to find idle jobs which have not yet had an instance provisioned for them. Migration is simulated by removing the idle instance from the first tenant and appending it

to the target tenant's pool of resources. The migrating instance is flagged as being *busy* for one minute to simulate the time required to migrate. Target tenants treat the migrating instance as a fulfilled instance request, meaning they terminate or repurpose outstanding requests.

Table 7.7 shows the results of deploying the platform with and without migration to fulfil the 6640 job dataset. During this period the platform detects 43 migration opportunities and transfers an instance between two tenants for each. The results show reductions in the overall number of instances across all but one of the tenants, with 16 fewer instances provisioned in total. Importantly, the average wait time also decreases for all but one of the tenants. The exception is due to the tenant donating too many of its resources and then having to acquire additional resources for subsequent jobs. Overall, these improvements demonstrate the potential benefits that can be gained by enabling services to collaboratively work together and share compute resources.

Table 7.7: Simulating the migration of instances between tenants.

	Instances		Avg Wait	
Tenant	Base	Migration	Base	Migration
1	430	425	308	299
2	22	18	306	260
3	67	68	242	223
4	292	285	407	410
5	2261	2250	347	346
Total	3062	3046	342	339

The results of this test show that migration can improve the global efficiency of cooperative cloud services by reducing the total number of instances (and therefore cost) as well as improving responsiveness. It is important to note that the test is conducted over a relatively short period of time with only five gateways. If applied to the collective services offered by Globus Galaxies (more than 30 gateways) over a period of years, migration could substantially reduce the overall operational costs of the gateways.

7.4 Summary

SCRIMP is an automated and cost-aware provisioning service which can be used by scientific services to manage cloud interactions while minimising costs, improving reliability, and exposing fine-grained auditing information. This chapter presented SCRIMP and investigated two research questions pertaining to reliably using unreliable cloud infrastructures and enabling the auditing of cloud utilisation. To address the fifth research question, **RQ5**, I explored the use of the predictive bidding strategy, DrAFTS. The results showed that using the forecast execution length of a workload can reduce the rate of instance terminations. Overall, DrAFTS-based bids substantially reduced the financial risk of using Spot instances and could help reduce instance terminations. The final research question, **RQ6**, asked how fine-grained account information can be maintained and used. This chapter has demonstrated the ability of SCRIMP's resource management component to record and enable the auditing of cloud utilisation. The resource management aspects of SCRIMP underpin many of the investigations conducted throughout this thesis by providing datasets to analyse and recording experimental results.

The role of SCRIMP's configurable parameters have been analysed in this chapter. A representative and repeatable dataset of workloads was defined and used to reliably evaluate the result of modifying SCRIMP's configuration. A simulation plug-in was presented to enable the comparison of tests using the same cloud market conditions. The simulation's results supported the findings of the emulated experiments.

Instance migration was explored as a means to improve global efficiency by enabling cooperative tenants to dynamically exchange resources. A platform to replay execution traces was used to evaluate the effectiveness of migration. The platform demonstrated migration was capable of reduction the number of required instances and improving the average wait time of jobs.

Chapter 8

Conclusion

This thesis presented an investigation into how scientific services can better utilise the cloud. Analysis of real-world scientific services identified a number of important challenges faced when performing on-demand computations in the cloud. Six research questions were defined to guide the research presented in this thesis. The research questions relate to a range of cloud-based issues, including: network limitations, cloud economics, provisioning models, and provisioning as a service. This thesis investigated these six research questions and describes technologies and techniques developed to address their underlying challenges.

A primary contribution of this research is SCRIMP. SCRIMP simplifies cloud usage by enabling applications and services to outsource the acquisition and management of on-demand cloud infrastructures. SCRIMP integrates many of the contributions presented throughout this thesis and has been demonstrated to reduce costs and improve throughput, reliability, and global efficiency.

This chapter reviews the research that has been presented in this thesis. The key contributions are then enumerated in Section 8.2. Finally, future research directions are proposed in Section 8.3.

8.1 Review

Chapter 3 presented the analysis of three scientific use cases. Analysis of these use cases identified a set of common challenges and were used to derive a set of research questions. The research questions explored in this thesis are as follows:

RQ1 How can opaque cloud network performance be accurately measured or inferred and used to minimise the need for, and cost of, data movement?

RQ2 How can cloud computing market models be exploited to minimise the cost of provisioning infrastructure to execute arbitrary scientific workloads?

RQ3 How can the resource requirements of unknown applications, with varying input data and configurations, be computed to determine the "best" instance types for execution?

RQ4 How can provisioning decisions be automated to consider trade-offs (e.g., cost, time, data movement) and optimally select instances for a given application?

RQ5 How can services improve the reliability of elastic clusters when employing potentially unreliable infrastructure?

RQ6 How can fine-grain accounting information be maintained for purposes of auditing and cost reclamation?

The remainder of this section discusses the research presented to address these research questions.

8.1.1 Cloud Network Limitations

Many scientific services rely on large datasets to perform analyses. However, cloud computing presents new challenges to data-intensive computing. The performance of cloud networks is often cited as a key limitation to performing scientific analyses in the cloud. In addition, cloud networks are opaque and do not publicly expose network information. This restricts the ability to exploit data-locality, which is an important optimisation feature for many distributed applications. These problems were captured by the first research question, **RQ1**. This thesis explored network tomography as a tool to reduce the limitations imposed by cloud networks, with a goal of enabling network-awareness for cloud-based services.

Chapter 4 presented an evaluation of tomographic techniques with respect to their ability to infer the network properties of commercial clouds. Two testbeds, comprised of up to 100 cloud instances, and distributed across multiple availability zones, were used to measure the effectiveness of different tomographic probing techniquesThe results of this work highlighted the noisy and inconsistent nature of an instance's network performance. Throttling of network throughput was also observed between instance types and across availability zones. The

results showed that the network performance varies considerably and that variations persist for prolonged periods of time. This provides an opportunity to react to network variations and make deployment decisions based on the measured network performance of resources.

A network health diagnostic system (NHDS) was developed and used to collect, monitor, and report on network performance. The NHDS employs a number of novel research contributions to determine and utilise network information. A set of health indicators, or tomographic probing schemes and bandwidth measurements, were analysed and evaluated to determine the influence each has when computing an overall health score. Regression testing determined appropriate weights of each indicator to reflect the priorities of data-intensive applications. Health markers were formulated and established to trigger notifications when substantial network volatility is detected. Health metrics are formulated to aggregate the information collected by each indicator into a comparable health score.

The NHDS can be deployed over a set of distributed resources to collect and compare the network performance of hosts. Once deployed, a relative health score is computed for either a specific host (e.g. the host maintaining a dataset), or all hosts in the environment. The health score provides insight into the topology and properties of the network connecting a set of resources. The deployment harness was shown to scale to over one hundred cloud resources and employs techniques to minimise the overhead incurred by using the system. The NHDS contributes to the understanding of the cloud infrastructure and can guide deployment and facilitate data-locality-based executions, alleviating the problem identified by the **RQ1**.

8.1.2 Cloud Economics

It is essential to leverage the full economic properties of a cloud provider to minimise the operational costs and establish a cost-aware scientific service. The need for economic cloud utilisation has been captured by **RQ2**. Chapter 5 explored techniques to cost-effectively select cloud resources when provisioning infrastructures. Four provisioning scopes were employed to to analyse the effect of expanding provisionable instance types and availability zones. Execution traces from six production science gateways were used in conjunction with price history data to explore different provisioning models and evaluate their cost and perfor-

mance. The effect of bid prices and their relation to instance reliability was also investigated, showing that instance reliability can be substantially improved via different bidding approaches. Over-provisioning strategies were also explored, where multiple requests are made for an individual job, as a means to improve throughput of the gateways. Overall, these techniques address the second research question and were shown to provide order of magnitude savings across the six gateways while improving reliability and increasing throughput.

The provisioning system can be deployed on a host with an HTCondor queue. The cost-aware provisioning system monitors the queue to identify idle jobs in need of resources. Customisable policies specify when an instance should be provisioned for a job. Once a set of appropriate resources is determined the system employs real-time cost information to guide the decision making process and minimise the costs of fulfilling a workload. Multiple cost-aware techniques are then employed to select an instance type, availability zone, and pricing model. Cloud-init is used to dynamically configure provisioned resources. In addition to selecting and launching a cloud resource, the system exposes an API which can be used to determine the lowest cost resource for a given set of requirements. Users can employ this system in custom provisioning systems to address the challenges associated with economically using the cloud and accomplish cost-aware cloud utilisation.

The provisioning system has two main limitations. Firstly, in recent trials, the over-provisioning technique has become less effective. This is due to the batch fulfilment process used by AWS, where multiple resources are occasionally acquired concurrently for an individual job. To counter this, the configurable run rate of the provisioning system was modified to increase its execution frequency and the duration between subsequent requests was expanded. Secondly, the system was not designed to scale over many services. The process of maintaining and upgrading the provisioning systems is tedious as it is operated independently on each infrastructure. This also means the system is restricted to a local perspective of the environment and can not make provisioning optimisations based on the global state of jobs and resources. These limitations motivated the development of SCRIMP, which is capable of concurrently serving many tenants.

8.1.3 Provisioning Models

Commercial cloud providers offer a plethora of instance types with different optimisations and configurations. While this flexibility enables resources to be specifically selected to optimally meet tool requirements, it also complicates the provisioning process. **RQ3** and **RQ4** capture the need to derive tool resource requirements to automatically and optimally select instances for arbitrary workloads. Chapter 6 explored these questions and presented an automated profiling service as the solution. Novel instrumentation of cloud instances is used to capture performance statistics during the execution of a tool. The profiling service records the memory, CPU, disk, and network utilisation throughout a tool's execution to provide detailed profiles of the tool behaviour and resource usage. A key contribution of this work was to demonstrate the advantages of using fine-grained tool profiles during the provisioning process. The logs from four production Globus Genomics gateways, with over 2000 executions of the monitored tools, were used to analyse the cost and performance benefits realised when using profiles. The results showed that employing profiles to minimise cost can achieve an 86.6% cost reduction in exchange for a 29.2% increase in execution time. Conversely, using profiles to improve throughput can reduce the execution time by 15.7% with a 62.6% reduction in cost when compared to the previous approach which used a predefined instance type for all jobs.

The automated profiling service can be used to deploy and monitor a tool as it is executed over a wide selection of cloud instance types. These records are automatically processed to construct profiles which summarise the resource requirements (such as total and maximum memory used). The profiling service leverages the cost-aware provisioning system, discussed in Chapter 5, to cost-effectively acquire the worker nodes used for profiling. It uses cloud-init to dynamically configure instances with profiling capabilities. Performance Co-Pilot is used to capture system performance and its logs are processed to construct a consumable JSON profile which is returned after execution. The service also supports the profiling of Docker containers. cAdvisor and an InfluxData are used to stream and store execution performance metrics, respectively, which is then used to create the same consumable JSON profiles as when profiling traditional executables.

The profiling service has two pertinent limitations. Firstly, the profiler is not currently capable of capturing the utilisation of specialised hardware, such as

GPUs. However, the profiler has been designed for extensibility to enable the incorporation of further monitoring components. Secondly, the profiler is unable to forecast the performance of a tool when executed with drastically different input datasets. This limitation has motivated SCRIMP's ability to employ feedback from production executions.

8.1.4 Provisioning as a Service

SCRIMP, described in Chapter 7, combines many of the contributions presented in this thesis into a single, usable, service. The service is currently used by production Globus Galaxies gateways to facilitate on-demand analyses in the cloud and is openly available on github [198]. SCRIMP presents a unique nexus in cross-over between these research areas and has been demonstrated to reduce costs, improve throughput, and improve the execution performance of scientific analyses deployed on the cloud.

SCRIMP incorporates the cost-aware provisioning system presented in Chapter 5. The provisioning system provides a model from which new provisioning and bidding strategies can be investigated to further reduce cloud computing costs. For example, SCRIMP has been instrumented to leverage DrAFTS [29] to use predictive bidding strategies and enable the exploration of **RQ5**. Predictive bidding was shown to be an effective mechanism able to reduce financial risk and improve the reliability of Spot instances. SCRIMP enables the combination of the DrAFTS service with forecast execution lengths (from profiles) which provides 99% probability guarantees that instances will not be terminated during the execution of a workload.

The profiling service allows SCRIMP to employ tool execution profiles and fine-grained resource requirements during instance provisioning. Combining tool profiles with cost-aware provisioning techniques gives SCRIMP a unique advantage over other cloud provisioning solutions. For example, SCRIMP is capable of exposing the trade-off between execution performance and monetary optimisation. This allows researchers to employ provisioning strategies which meet their requirements in terms of prioritising execution or cost.

At the heart of SCRIMP is a resource management and analytics platform. The platform is responsible for performing and recording all cloud interactions as well as monitoring the state of instances, tenants, and jobs. The data collected by the platform has been fundamental to much of the research presented in this

thesis. For example, the cost-aware research relied on gathering cost data from the instances deployed by production science gateways. The platform's records were also used to identify the most frequently executed Globus Genomics tools to guide profiling efforts. Furthermore, analysis of the platform's logs regarding production usage has enabled the creation of a dataset of genomics analysis workflows. The dataset contains information on over 8000 tool executions, including their specified requirements (CPU, memory, etc.), and can be repeatedly deployed to a queue to emulate workloads of varying intensity and size. This dataset enabled investigation of SCRIMP under a wide range of loads and configurations. The dataset was also used as the basis for evaluating the DrAFTS predictive bidding system, providing the workloads to be deployed, and then capturing the instances launched and costs incurred by each strategy.

8.1.5 Infrastructure Management

A common challenge faced by cloud-based services is reliably and efficiently managing and analysing provisioned resources. Programmatic provisioning solutions, which leverage cloud APIs and custom management scripts, are necessary to acquire, organise, and govern large cloud infrastructures. Ensuring an appropriate number of resources are launched, correctly configured, and subsequently terminated is essential for an on-demand scientific service. Additionally, recording the provisioned resource types, important event times, termination causes, overall cost, and associating instances with individual users, are crucial to adapting provisioning approaches, analysing usage, and providing accountability and auditing mechanisms.

Research presented in Chapter 7 explored techniques for managing provisioned cloud resources. One such technique was to employ a novel framework for migrating instances between cooperative tenants in order to improve global efficiency. The chapter also investigated **RQ6**, which asks how usage can be measured and how auditing capabilities can be leveraged to reduce costs. A key design goal of SCRIMP's management platform was to enable the exploration of this research question by capturing provisioning decisions and cloud usage. The management platform is capable of recording both usage as well as speculative usage, where multiple provisioning algorithms can be employed concurrently and evaluated with the same market conditions. The analytical capabilities of the management platform enable users to compare and contrast provisioning strate-

gies to optimise usage and minimise costs. Finally, a cloud simulation plug-in was also presented. The highly customisable plug-in is able to replace the cloud provider and enables provisioning techniques to be explored without incurring the cost of launching cloud resources.

8.2 Contributions

This thesis explored a number of different aspects regarding the operation of cloud-based scientific services. Research contributions have been made in multiple areas, including: scientific gateways, analysis of cloud networks, cost-aware cloud utilisation, tool profiling, and cloud provisioning. The research contributions presented in this thesis are restated here from Section 1.2. Specifically, the major contributions of this thesis are:

1. Development of three scientific services and gateways in collaboration with other researchers. The requirements of these use cases motivate the research presented in this thesis. In particular I have made the following contributions to these projects:

   (a) Design and development of a cloud-based service for reconstructing proton computed tomography (pCT) images on-demand [17]. I implemented the service which utilises more than 100 GPU-enabled cloud instances to reconstruct medical images on-demand. I also created a simulation framework to demonstrate the scalability of the service. This research was conducted in collaboration with researchers from Northern Illinois University (NIU) and Argonne National Laboratory (ANL).

   (b) Design and development of the Portal for Data Analysis Services for Cosmological Simulations (PDACS) [32, 33]. This work has allowed researchers to perform cosmological analyses using Galaxy [34] workflows and seamlessly deploy large-scale analyses over HPC resources. PDACS was developed in a collaboration with researchers from ANL, Fermi National Accelerator Laboratory (FNAL), and the National Energy Research Scientific Computing Center (NERSC).

   (c) I participate in the on-going development of the Globus Galaxies platform [28]. The Globus Galaxies platform combines a number of data

management and analysis services, such as Globus [35] and Galaxy, to simplify the creation of scientific services. It leverages cloud resources to dynamically fulfil scientific workflow requirements. The Globus Galaxies platform has been applied to a number of different scientific domains including genomics [36], climate and policy [37], traumatic brain injury [31], and cardiovascular [38] research. My key contributions have been to design and develop provisioning solutions, enable fine-grained analysis of gateways, and optimise resource selection by profiling tools. The Globus Galaxies project is a collaborative effort between researchers at the University of Chicago and ANL.

2. Development of a network tomography framework to analyse cloud networks and evaluate network performance. The primary contributions of this work are:

   (a) Development of a network tomography framework. The framework enables users to express tomographic probing sequences and efficiently deploy them over large cloud infrastructures to evaluate and infer network properties.

   (b) Analysis of tomographic information to determine properties of commercial clouds. I show that cloud instances experience substantial variations in network performance which persist for prolonged periods of time.

   (c) Formulation of health metrics to derive the relative network performance of cloud instances. These metrics are used to guide instance selection and are demonstrated to improve the execution performance of the pCT codes.

   (d) Development of a Network Health Diagnostic System (NHDS). The system can be deployed over a set of cloud instances and will compute relative health scores between the hosts in order to enable network-aware deployment decisions.

3. Analysis of cloud economics to establish cost-effective techniques to fulfil scientific workloads. I explore various approaches to achieve cost-aware provisioning and demonstrate substantial improvements in both cost and execution time over production Globus Galaxies gateways.

4. Development of an automatic tool profiling service for the cloud. The service enables users to profile executable applications and docker containers over a wide range of cloud instance types. Resource (CPU, memory, disk, and network) utilisation is monitored and used to automatically construct fine-grained tool profiles. In addition, I explore online profiling techniques to capture execution traces and dynamically adjust execution forecasts.

5. Development of a model for migrating cloud instances between scientific services. In collaboration with researchers from University of Wisconsin-Madison, the University of Chicago, and ANL, I have developed a framework for dynamic migration of cloud instances between HTCondor pools. This work has been shown to reduce the number of instances launched and improve the throughput of several Globus Galaxies services.

6. Design and development of SCRIMP—a multi-faceted service for acquiring and managing cloud infrastructure. The service combines the above contributions to improve the performance of scientific services. SCRIMP is unique in the following ways:

   (a) Incorporates real-time market information to dynamically adjust the provisioning approach and is capable of selecting resources across availability zones and acquirement models (On-demand or Spot requests).

   (b) Uses detailed tool profiles to guide provisioning. Profiles are combined with real-time market information to exploit the trade-off in cost and performance.

   (c) Captures network performance information between provisioned instances to enable network-aware decision making.

   (d) Includes a resource management and analytics platform to capture cloud interactions and monitor provisioned instances. The platform enables the analysis of different provisioning approaches and provides cloud utilisation metrics.

   (e) Integrates a predictive bidding strategy, DrAFTS [29, 39] to provide reliability guarantees for provisioned Spot instances. This work was conducted in collaboration with researchers at the University of California at Santa Barbra.

(f) Includes a custom simulation plug-in to emulate the AWS cloud and enable rapid experimentation with little resource costs.

(g) Is capable of dynamically constructing on-demand clusters (HTCondor or Apache Spark) to fulfil workloads.

(h) Enables resource migration to improve global efficiency and reduces instance acquisition time by enabling instances to be shared between tenants.

8.3 Future Work

Due to the scope of the research presented in this thesis there are numerous areas for potential extension and future work.

8.3.1 Network Tomography

Using network tomography to select clusters of cloud resources, based on their network performance, has been shown to improve the performance of scientific applications. An immediate research direction is to combine tomographic information with the resource migration capabilities of SCRIMP. This would enable resources to be shuffled between collaborative services to globally improve throughput. Collecting tomographic information between each worker and its headnode would enable resources to be migrated to optimally create cloud infrastructures.

Another goal is to further integrate network information into SCRIMP, providing forecasts and estimates on data transfers. This information could be used to guide the deployment of data-intensive workloads. For example, the location of data sources could be encoded within jobs and used by the provisioner to select the regions and availability zones of resources in order to minimise expensive cross-region executions.

Techniques to visualise the information that is inferred from the tomographic measurements should be applied. Real-time visualisations of large cloud services could aid in management decisions and provide intuitive feedback on the state and operation of large scientific services.

Further research is required to establish custom tomographic probing schemes specifically designed for cloud infrastructures. The traditional tomographic mea-

surements are typically evaluated over prototype networks to avoid interference. However, more research is required to develop new tomographic schemes which are resilient to noise and can reliably infer cloud topologies. The cloud also presents many new challenges to network tomography. For example, resources can be migrated by providers for load-balancing reasons without advising the customer. Therefore, adaptive probing schemes must be explored to meet the unique challenges of the cloud.

8.3.2 Cloud Economics

Cost-aware provisioning is essential to the success of cloud-based scientific services. Using the DrAFTS system I aim to further evaluate bidding strategies in order to increase the reliability of Spot instances while minimising the cost of operating scientific services.

Another topic for future research is to explore the ability to acquire free cloud computing resources. AWS does not bill users for partial instance hours when a spot request is preempted due to a bid being exceeded. Employing the DrAFTS service in reverse, whereby it nominates bid prices with a guarantee that the instance will be terminated within a billable hour, presents an opportunity to acquire free compute resources. This would be a novel application of research into bidding strategies with potential economic benefit.

8.3.3 Tool Profiling

The profiling service provides an automated mechanism for capturing resource requirements. However, the service requires additional work in order to support more complex configurations, such as those involving EBS storage. Future work is also needed to integrate support for delegated IAM roles to enable use of the service without the need for user AWS credentials.

The incorporation of online profile feedback into the provisioning process requires further investigation. Additional research is required to improve and evaluate the aggregation of tool profiles that are automatically captured during production executions. In order to react to changes in use, such as researchers employing different datasets, or tool optimisations, the automatic aggregation techniques should prioritise recent execution traces over historic traces.

The profiling service is currently being used to survey a large selection of tools

used in Globus Genomics deployments. Once the survey is complete, I aim to publicly release a large dataset consisting of fine-grained tool profiles and thousands of execution traces from numerous production gateways. This dataset will provide a real-world use case for researchers exploring scheduling techniques and investigating scientific deployments.

8.3.4 Infrastructure Management

The resource management platform has been demonstrated to effectively integrate the cost-aware provisioning service with the AWS cloud and the simulation plug-in. However, additional work is required to expand the capabilities of the platform to include additional cloud providers. In turn, this could enable the federation of cloud providers and present new avenues for cost-aware resource provisioning.

The cloud simulation plug-in requires further investigation in order to better validate its performance. Currently, the plug-in has been used to emulate cloud functionality and has been compared against a set of provisioning tests. More rigorous evaluation is required to ensure the performance of the plug-in is representative of the cloud provider. This plug-in should also be made publicly available as it is shown to enable rapid prototyping of provisioning approaches without the need for acquiring cloud resources and could benefit many researchers.

Additional research should be conducted to investigate the use of autonomic computing practices in order to incorporate the analytical capabilities of the service into the provisioning process. In turn, analysing resource utilisation could be used dynamically to alter the provisioning policies to match workload characteristics in order to optimise the number of instances that are acquired by a service.

8.3.5 Data Analytics

Scientific discovery is increasingly reliant on big data, computation and data-science, and large-scale computing infrastructure. Efficiently and reliably managing, curating, synthesising, and analysing data is a challenge faced in almost every scientific domain. My primary research direction is to extend SCRIMP to address these challenges. I aim to develop an extensible data-oriented analytics platform. The goal of this platform is to lower the barriers for use of new analysis

capabilities and large computational infrastructures (supercomputers, clusters, and clouds) to enable scientists to more easily conduct cutting edge computation and data-science irrespective of data sizes. This platform will make scientific data, housed in disparate repositories and storage systems, available and manageable. It will provide a fabric for integrating and synthesising heterogeneous data. It will enable access to a suite of analytics applications that can be efficiently applied to a variety of data to derive new insights. SCRIMP will underpin the platform by providing an extensible framework that facilitates the exploitation of large-scale computing resources. I aim to extend SCRIMP to incorporate multiple computing resources and enable the deployment of specialised computing modalities (e.g., HTC, MapReduce, and Many Task computing). In turn, this platform will democratise access to advanced capabilities, enabling a huge pool of researchers to access and use novel and best practice analytics techniques on leadership class computing systems.

Bibliography

[1] R. Schulte, V. Bashkirov, T. Li, Z. Liang, K. Mueller, J. Heimann, L. Johnson, B. Keeney, H. F.-W. Sadrozinski, A. Seiden, D. Williams, L. Zhang, Z. Li, S. Peggs, T. Satogata, and C. Woody, "Conceptual design of a proton computed tomography system for applications in proton radiation therapy," *IEEE Transactions on Nuclear Science*, vol. 51, no. 3, pp. 866–872, 2004.

[2] J. Towns, T. Cockerill, M. Dahan, I. Foster, K. Gaither, A. Grimshaw, V. Hazlewood, S. Lathrop, D. Lifka, G. D. Peterson, *et al.*, "Xsede: accelerating scientific discovery," *Computing in Science & Engineering*, vol. 16, no. 5, pp. 62–74, 2014.

[3] P. E. Dewdney, P. J. Hall, R. T. Schilizzi, and T. J. L. Lazio, "The square kilometre array," *Proceedings of the IEEE*, vol. 97, no. 8, pp. 1482–1496, 2009.

[4] L. Boatman, "The significance of Big Data in invertebrate zoology." http://nmnh.typepad.com/no_bones/2015/02/the-significance-of-big-data-in-invertebrate-zoology.html, 2015.

[5] R. L. Villars, C. W. Olofson, and M. Eastwood, "Big data: What it is and why you should care," *White Paper, IDC*, 2011.

[6] I. Foster, Y. Zhao, I. Raicu, and S. Lu, "Cloud computing and Grid computing 360-degree compared," in *Grid Computing Environments Workshop, 2008. GCE'08*, pp. 1–10, Ieee, 2008.

[7] V. Mayer-Schönberger and K. Cukier, *Big data: A revolution that will transform how we live, work, and think*. Houghton Mifflin Harcourt, 2013.

[8] Q. He, S. Zhou, B. Kobler, D. Duffy, and T. McGlynn, "Case study for running HPC applications in public clouds," in *Proceedings of the 19th ACM*

International Symposium on High Performance Distributed Computing, pp. 395–401, 2010.

[9] C. Vecchiola, S. Pandey, and R. Buyya, "High-performance cloud computing: A view of scientific applications," in *Pervasive Systems, Algorithms, and Networks (ISPAN), 2009 10th International Symposium on*, pp. 4–16, IEEE, 2009.

[10] A. Khajeh-Hosseini, D. Greenwood, and I. Sommerville, "Cloud migration: a case study of migrating an enterprise it system to iaas," in *3rd International Conference on Cloud Computing (CLOUD),*, pp. 450–457, IEEE, 2010.

[11] I. A. T. Hashem, I. Yaqoob, N. B. Anuar, S. Mokhtar, A. Gani, and S. U. Khan, "The rise of big data on cloud computing: Review and open research issues," *Information Systems*, vol. 47, pp. 98–115, 2015.

[12] A. Iosup, S. Ostermann, M. N. Yigitbasi, R. Prodan, T. Fahringer, and D. H. Epema, "Performance analysis of cloud computing services for many-tasks scientific computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 22, no. 6, pp. 931–945, 2011.

[13] C. Hoffa, G. Mehta, T. Freeman, E. Deelman, K. Keahey, B. Berriman, and J. Good, "On the use of cloud computing for scientific workflows," in *Proceedings of the 4th International Conference on eScience*, pp. 640–645, IEEE, 2008.

[14] S. Ostermann, A. Iosup, N. Yigitbasi, R. Prodan, T. Fahringer, and D. Epema, "A performance analysis of ec2 cloud computing services for scientific computing," in *Cloud computing*, pp. 115–131, Springer, 2009.

[15] C. Evangelinos and C. Hill, "Cloud computing for parallel scientific hpc applications: Feasibility of running coupled atmosphere-ocean climate models on amazons ec2," *ratio*, vol. 2, no. 2.40, pp. 2–34, 2008.

[16] S. C. Park and S. Y. Ryoo, "An empirical investigation of end-users switching toward cloud computing: A two factor theory perspective," *Computers in Human Behavior*, vol. 29, no. 1, pp. 160–170, 2013.

[17] R. Chard, R. Madduri, N. Karonis, K. Chard, K. Duffin, C. Ordonez, T. Uram, J. Fleischauer, I. Foster, M. Papka, and J. Winans, "Scalable pCT

image reconstruction delivered as a cloud service," *IEEE Transactions on Cloud Computing*, vol. Preprint, no. 99, pp. 1–1, 2015.

[18] I. T. Foster and R. K. Madduri, "Science as a service: how on-demand computing can accelerate discovery," in *Proceedings of the 4th ACM workshop on Scientific cloud computing*, pp. 1–2, ACM, 2013.

[19] S. Hazelhurst, "Scientific computing using virtual high-performance computing: a case study using the amazon elastic computing cloud," in *Proceedings of the 2008 annual research conference of the South African Institute of Computer Scientists and Information Technologists on IT research in developing countries: riding the wave of technology*, pp. 94–103, ACM, 2008.

[20] D. Battré, N. Frejnik, S. Goel, O. Kao, and D. Warneke, "Inferring network topologies in infrastructure as a service cloud," in *11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, pp. 604 –605, may 2011.

[21] S. Yi, A. Andrzejak, and D. Kondo, "Monetary cost-aware checkpointing and migration on Amazon cloud spot instances," *IEEE Transactions on Services Computing*, vol. 5, no. 4, pp. 512–524, 2012.

[22] H. Huang, L. Wang, B. C. Tak, L. Wang, and C. Tang, "Cap3: A cloud auto-provisioning framework for parallel processing using on-demand and spot instances," in *2013 IEEE Sixth International Conference on Cloud Computing*, pp. 228–235, IEEE, 2013.

[23] I. Foster, "Service-oriented science," *Science*, vol. 308, no. 5723, pp. 814–817, 2005.

[24] N. Wilkins-Diehr, D. Gannon, G. Klimeck, S. Oster, and S. Pamidighantam, "Teragrid science gateways and their impact on science," *Computer*, vol. 41, no. 11, pp. 32–41, 2008.

[25] K. R. Jackson, L. Ramakrishnan, K. Muriki, S. Canon, S. Cholia, J. Shalf, H. J. Wasserman, and N. J. Wright, "Performance analysis of high performance computing applications on the Amazon Web Services cloud," in *Proceedings of the 2nd International Conference on Cloud Computing Technology and Science (CloudCom)*, pp. 159–168, IEEE, 2010.

[26] "Amazon Web Services (AWS)." http://aws.amazon.com/, Accessed on April 2016.

[27] "Amazon Elastic Compute Cloud (EC2)." https://aws.amazon.com/ec2/, Accessed on April 2016.

[28] R. Madduri, K. Chard, R. Chard, L. Lacinski, A. Rodriguez, D. Sulakhe, D. Kelly, U. Dave, and I. Foster, "The Globus Galaxies platform: delivering science gateways as a service," *Concurrency and Computation: Practice and Experience*, vol. 27, no. 16, pp. 4344–4360, 2015.

[29] R. Wolski and J. Brevik, "Providing statistical reliability guarantees in the aws spot tier," in *Proceedings of the 24th High Performance Computing Symposium*, 2016.

[30] A. V. Do, J. Chen, C. Wang, Y. C. Lee, A. Zomaya, and B. B. Zhou, "Profiling applications for virtual machine placement in clouds," in *Proceedings of the IEEE International Conference on Cloud Computing (CLOUD)*, pp. 660–667, July 2011.

[31] K. Chard, R. Madduri, X. Jiang, F. Dahi, M. W. Vannier, and I. Foster, "A cloud-based image analysis gateway for traumatic brain injury research," in *Proceedings of the 9th Gateway Computing Environments Workshop*, pp. 13–16, IEEE, 2014.

[32] R. Chard, S. Sehrish, A. Rodriguez, R. Madduri, T. D. Uram, M. Paterno, K. Heitmann, S. Cholia, J. Kowalkowski, and S. Habib, "PDACS: a portal for data analysis services for cosmological simulations," in *Proceedings of the 9th Gateway Computing Environments Workshop*, pp. 30–33, IEEE, 2014.

[33] R. Madduri, A. Rodriguez, T. Uram, K. Heitmann, T. Malik, S. Sehrish, R. Chard, S. Cholia, M. Paterno, J. Kowalkowski, and S. Habib, "PDACS: a portal for data analysis services for cosmological simulations," *Computing in Science & Engineering*, vol. 17, no. 5, pp. 18–26, 2015.

[34] J. Goecks, A. Nekrutenko, J. Taylor, *et al.*, "Galaxy: a comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences," *Genome Biol*, vol. 11, no. 8, p. R86, 2010.

[35] I. Foster, "Globus Online: Accelerating and democratizing science through cloud-based services," *Internet Computing, IEEE*, vol. 15, pp. 70–73, May 2011.

[36] R. K. Madduri, D. Sulakhe, L. Lacinski, B. Liu, A. Rodriguez, K. Chard, U. J. Dave, and I. T. Foster, "Experiences building Globus Genomics: a next-generation sequencing analysis service using Galaxy, Globus, and Amazon Web Services," *Concurrency and Computation: Practice and Experience*, vol. 26, no. 13, pp. 2266–2279, 2014.

[37] R. Montella, D. Kelly, W. Xiong, A. Brizius, J. Elliott, R. Madduri, K. Maheshwari, C. Porter, P. Vilter, M. Wilde, M. Zhang, and I. Foster, "FACE-IT: A science gateway for food security research," *Concurrency and Computation: Practice and Experience*, vol. 27, no. 16, pp. 4423–4436, 2015.

[38] R. Winslow, J. Saltz, I. Foster, *et al.*, "The cardiovascular research grid (CVRG) project," *Proceedings of the AMIA Summit on Translational Bioinformatics*, pp. 77–81, 2011.

[39] R. Wolski, J. Brevik, R. Chard, and K. Chard, "Probabilistic guarantees of execution duration for amazon spot instances," in *Submitted to the IEEE International Conference on Cloud Engineering (IC2E)*, 2017.

[40] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, *et al.*, "A view of cloud computing," *Communications of the ACM*, vol. 53, no. 4, pp. 50–58, 2010.

[41] Q. Zhang, L. Cheng, and R. Boutaba, "Cloud computing: state-of-the-art and research challenges," *Journal of internet services and applications*, vol. 1, no. 1, pp. 7–18, 2010.

[42] "Microsoft Azure." https://azure.microsoft.com/, Accessed on April 2016.

[43] "Google App Engine." https://cloud.google.com/appengine/, Accessed on April 2016.

[44] "Heroku." https://heroku.com/, Accessed on April 2016.

[45] "Dropbox." https://www.dropbox.com/, Accessed on April 2016.

[46] "Google Docs." https://docs.google.com/, Accessed on April 2016.

[47] 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. 50:1–50:12, IEEE Press, 2008.

[48] K. Keahey, R. Figueiredo, J. Fortes, T. Freeman, and M. Tsugawa, "Science clouds: Early experiences in cloud computing for scientific applications," *Cloud computing and applications*, vol. 2008, pp. 825–830, 2008.

[49] K. Keahey, "Cloud computing for science.," in *SSDBM*, p. 478, 2009.

[50] H. A. Hassan, S. A. Mohamed, and W. M. Sheta, "Scalability and communication performance of hpc on azure cloud," *Egyptian Informatics Journal*, 2015.

[51] Y. El-Khamra, H. Kim, S. Jha, and M. Parashar, "Exploring the performance fluctuations of hpc workloads on clouds," in *Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on*, pp. 383 –387, 30 2010-dec. 3 2010.

[52] A. Gupta and D. Milojicic, "Evaluation of hpc applications on cloud," in *Open Cirrus Summit (OCS), 2011 Sixth*, pp. 22 –26, oct. 2011.

[53] E. Walker, "Benchmarking amazon ec2 for high-performance scientific computing," *Usenix Login*, vol. 33, no. 5, pp. 18–23, 2008.

[54] D. H. Bailey, E. Barszcz, J. T. Barton, D. S. Browning, R. L. Carter, L. Dagum, R. A. Fatoohi, P. O. Frederickson, T. A. Lasinski, R. S. Schreiber, *et al.*, "The NAS parallel benchmarks summary and preliminary results," in *Supercomputing, 1991. Supercomputing'91. Proceedings of the 1991 ACM/IEEE Conference on*, pp. 158–165, IEEE, 1991.

[55] P. Mehrotra, J. Djomehri, S. Heistand, R. Hood, H. Jin, A. Lazanoff, S. Saini, and R. Biswas, "Performance evaluation of Amazon EC2 for NASA HPC applications," in *Proceedings of the 3rd workshop on Scientific Cloud Computing*, pp. 41–50, 2012.

[56] D. Lifka, I. Foster, S. Mehringer, M. Parashar, P. Redfern, C. Stewart, and S. Tuecke, "XSEDE cloud survey report," tech. rep., Technical report, National Science Foundation, USA, 2013.

[57] L. Heilig and S. Voss, "A scientometric analysis of cloud computing literature," *Cloud Computing, IEEE Transactions on*, vol. 2, no. 3, pp. 266–278, 2014.

[58] L. Ramakrishnan, P. T. Zbiegel, S. Campbell, R. Bradshaw, R. S. Canon, S. Coghlan, I. Sakrejda, N. Desai, T. Declerck, and A. Liu, "Magellan: Experiences from a science cloud," in *Proceedings of the 2nd international workshop on Scientific cloud computing*, pp. 49–58, 2011.

[59] S. Gesing, R. Dooley, M. Pierce, J. Kruger, R. Grunzke, S. Herres-Pawlis, and A. Hoffmann, "Science gateways-leveraging modeling and simulations in hpc infrastructures via increased usability," in *High Performance Computing & Simulation (HPCS), 2015 International Conference on*, pp. 19–26, IEEE, 2015.

[60] S. Marru, L. Gunathilake, C. Herath, P. Tangchaisin, M. Pierce, C. Mattmann, R. Singh, T. Gunarathne, E. Chinthaka, R. Gardler, *et al.*, "Apache Airavata: a framework for distributed applications and computational workflows," in *Proceedings of the 2011 workshop on Gateway computing environments*, pp. 21–28, ACM, 2011.

[61] R. Dooley, M. Vaughn, D. Stanzione, S. Terry, and E. Skidmore, "Software-as-a-service: the iPlant foundation API," in *5th IEEE Workshop on Many-Task Computing on Grids and Supercomputers (MTAGS)*, Citeseer, 2012.

[62] R. Ananthakrishnan, K. Chard, I. Foster, and S. Tuecke, "Globus platform-as-a-service for collaborative science applications," *Concurrency and Computation: Practice and Experience*, vol. 27, no. 2, pp. 290–305, 2015. CPE-13-0323.R1.

[63] B. Ludäscher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger, M. Jones, E. A. Lee, J. Tao, and Y. Zhao, "Scientific workflow management and the kepler system," *Concurrency and Computation: Practice and Experience*, vol. 18, no. 10, pp. 1039–1065, 2006.

[64] K. Wolstencroft, R. Haines, D. Fellows, A. Williams, D. Withers, S. Owen, S. Soiland-Reyes, I. Dunlop, A. Nenadic, P. Fisher, *et al.*, "The taverna workflow suite: designing and executing workflows of web services on the desktop, web or in the cloud," *Nucleic acids research*, p. gkt328, 2013.

[65] L. K. Zentner, S. M. Clark, P. M. Smith, S. Shivarajapura, V. Farnsworth, K. P. Madhavan, and G. Klimeck, "Practical considerations in cloud utilization for the science gateway nanoHUB.org," in *Procceedings of the 4th IEEE International Conference on Utility and Cloud Computing (UCC)*, pp. 287–292, IEEE, 2011.

[66] E. Skidmore, S.-j. Kim, S. Kuchimanchi, S. Singaram, N. Merchant, and D. Stanzione, "iPlant atmosphere: A gateway to cloud infrastructure for the plant sciences," in *Proceedings of the 2011 ACM Workshop on Gateway Computing Environments (GCE)*, pp. 59–64, ACM, 2011.

[67] W. Wu, H. Zhang, Z. Li, and Y. Mao, "Creating a cloud-based life science gateway," in *E-Science (e-Science), 2011 IEEE 7th International Conference on*, pp. 55–61, IEEE, 2011.

[68] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," in *Proceedings of the ACM SIGCOMM conference on Data communication*, SIGCOMM '08, (New York, NY, USA), pp. 63–74, ACM, 2008.

[69] C. E. Leiserson, "Fat-trees: universal networks for hardware-efficient supercomputing," *IEEE Trans. Comput.*, vol. 34, pp. 892–901, Oct. 1985.

[70] 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," in *Proceedings of the nineteenth ACM symposium on Operating systems principles*, SOSP '03, (New York, NY, USA), pp. 164–177, ACM, 2003.

[71] A. Velte and T. Velte, *Microsoft virtualization with Hyper-V*. McGraw-Hill, Inc., 2009.

[72] D. Milojicic, I. Llorente, and R. S. Montero, "Opennebula: A cloud management tool," *Internet Computing, IEEE*, vol. 15, no. 2, pp. 11–14, 2011.

[73] O. Sefraoui, M. Aissaoui, and M. Eleuldj, "Openstack: toward an open-source solution for cloud computing," *International Journal of Computer Applications*, vol. 55, no. 3, 2012.

[74] G. Wang and T. Ng, "The impact of virtualization on network performance of amazon ec2 data center," in *INFOCOM, 2010 Proceedings IEEE*, pp. 1 –9, march 2010.

[75] L. Youseff, R. Wolski, B. Gorda, and C. Krintz, "Evaluating the performance impact of xen on mpi and process execution for hpc systems," in *Virtualization Technology in Distributed Computing, 2006. VTDC 2006. First International Workshop on*, pp. 1–1, 2006.

[76] A. Agarwal, R. Desmarais, I. Gable, D. Grundy, D. P-Brown, R. Seuster, D. C. Vanderster, A. Charbonneau, R. Enge, and R. Sobie, "Deploying hep applications using xen and globus virtual workspaces," *Journal of Physics: Conference Series*, vol. 119, no. 6, p. 062002, 2008.

[77] P. R. Luszczek, D. H. Bailey, J. J. Dongarra, J. Kepner, R. F. Lucas, R. Rabenseifner, and D. Takahashi, "The hpc challenge (hpcc) benchmark suite," in *Proceedings of the 2006 ACM/IEEE conference on Supercomputing*, p. 213, Citeseer, 2006.

[78] A. Tikotekar, G. Vallée, T. Naughton, H. Ong, C. Engelmann, and S. L. Scott, "An analysis of hpc benchmarks in virtual machine environments," in *Euro-Par 2008 Workshops-Parallel Processing*, pp. 63–71, Springer, 2009.

[79] J. Ekanayake and G. Fox, "High performance parallel computing with clouds and cloud technologies.," in *CloudComp* (D. R. Avresky, M. Diaz, A. Bode, B. Ciciani, and E. Dekel, eds.), vol. 34 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, pp. 20–38, Springer, 2009.

[80] A. Li, X. Yang, S. Kandula, and M. Zhang, "Cloudcmp: comparing public cloud providers," in *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*, pp. 1–14, ACM, 2010.

[81] W. Voorsluys, S. K. Garg, and R. Buyya, "Provisioning spot market cloud resources to create cost-effective virtual clusters," in *Algorithms and Architectures for Parallel Processing*, pp. 395–408, Springer, 2011.

[82] M. J. Litzkow, M. Livny, and M. W. Mutka, "Condor–a hunter of idle workstations," in *Proceedings of the 8th International Conference on Distributed Computing Systems*, pp. 104–111, IEEE, 1988.

[83] T. White, *Hadoop: The Definitive Guide*. O'Reilly Media, Inc., 1st ed., 2009.

[84] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster computing with working sets.," *HotCloud*, vol. 10, pp. 10–10, 2010.

[85] "Python-boto." http://boto3.readthedocs.org/en/latest/index.html, Accessed on April 2016.

[86] "Amazon Spot Fleet." http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/spot-fleet.html, Accessed on April 2016.

[87] "Amazon Elastic MapReduce (EMR)." https://aws.amazon.com/elasticmapreduce/, Accessed on April 2016.

[88] "Fenzo)." https://github.com/Netflix/Fenzo/wiki, Accessed on April 2016.

[89] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. H. Katz, S. Shenker, and I. Stoica, "Mesos: A platform for fine-grained resource sharing in the data center.," in *NSDI*, vol. 11, pp. 22–22, 2011.

[90] P. Armstrong, A. Agarwal, A. Bishop, A. Charbonneau, R. Desmarais, K. Fransham, N. Hill, I. Gable, S. Gaudet, S. Goliath, *et al.*, "Cloud scheduler: a resource manager for distributed compute clouds," *arXiv preprint arXiv:1007.0050*, 2010.

[91] R. Sobie, A. Agarwal, I. Gable, C. Leavett-Brown, M. Paterson, R. Taylor, A. Charbonneau, R. Impey, and W. Podiama, "Htc scientific computing in a distributed cloud environment," in *Proceedings of the 4th ACM workshop on Scientific cloud computing*, pp. 45–52, ACM, 2013.

[92] "StarCluster." http://star.mit.edu/cluster/, Accessed on April 2016.

[93] M. D. de Assuncao, 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.

[94] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov, "The Eucalyptus open-source cloud-computing system," in *Cluster Computing and the Grid, 2009. CCGRID'09. 9th IEEE/ACM International Symposium on*, pp. 124–131, IEEE, 2009.

[95] R. N. Calheiros, R. Ranjan, and R. Buyya, "Virtual machine provisioning based on analytical performance and qos in cloud computing environments," in *Parallel Processing (ICPP), 2011 International Conference on*, pp. 295–304, IEEE, 2011.

[96] U. Sharma, P. Shenoy, S. Sahu, and A. Shaikh, "A cost-aware elasticity provisioning system for the cloud," in *Distributed Computing Systems (ICDCS), 2011 31st International Conference on*, pp. 559–570, IEEE, 2011.

[97] S. Abrishami, M. Naghibzadeh, and D. H. Epema, "Deadline-constrained workflow scheduling algorithms for infrastructure as a service clouds," *Future Generation Computer Systems*, vol. 29, no. 1, pp. 158–169, 2013.

[98] R. Van den Bossche, K. Vanmechelen, and J. Broeckhove, "Cost-optimal scheduling in hybrid IaaS clouds for deadline constrained workloads," in *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on*, pp. 228–235, IEEE, 2010.

[99] M. Mao and M. Humphrey, "Auto-scaling to minimize cost and meet application deadlines in cloud workflows," in *Proceedings of International Conference for High Performance Computing, Networking, Storage and Analysis*, p. 49, ACM, 2011.

[100] V. Arabnejad and K. Bubendorfer, "Cost effective and deadline constrained scientific workflow scheduling for commercial clouds," in *Network Computing and Applications (NCA), 2015 IEEE 14th International Symposium on*, pp. 106–113, IEEE, 2015.

[101] M. Malawski, G. Juve, E. Deelman, and J. Nabrzyski, "Cost- and deadline-constrained provisioning for scientific workflow ensembles in IaaS clouds," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, (Los Alamitos, CA, USA), pp. 22:1–22:11, IEEE Computer Society Press, 2012.

[102] R. Van den Bossche, K. Vanmechelen, and J. Broeckhove, "Online cost-efficient scheduling of deadline-constrained workloads on hybrid clouds," *Future Generation Computer Systems*, vol. 29, no. 4, pp. 973–985, 2013.

[103] N. Netjinda, B. Sirinaovakul, and T. Achalakul, "Cost optimization in cloud provisioning using particle swarm optimization," in *Electrical Engineering/-*

Electronics, Computer, Telecommunications and Information Technology (ECTICON), 2012 9th International Conference on, pp. 1–4, IEEE, 2012.

[104] W. Voorsluys and R. Buyya, "Reliable provisioning of spot instances for compute-intensive applications," in *Proceedings of the 26th International Conference on Advanced Information Networking and Applications (AINA)*, pp. 542–549, IEEE, 2012.

[105] L. M. Leslie, Y. C. Lee, and A. Y. Zomaya, "Ramp: reliability-aware elastic instance provisioning for profit maximization," *The Journal of Supercomputing*, vol. 71, no. 12, pp. 4529–4554, 2015.

[106] S. Tang, J. Yuan, and X.-Y. Li, "Towards optimal bidding strategy for amazon ec2 cloud spot instance," in *5th International Conference on Cloud Computing (CLOUD)*, pp. 91–98, IEEE, 2012.

[107] Y. Song, M. Zafer, and K.-W. Lee, "Optimal bidding in spot instance market," in *2012 Proceedings INFOCOM*, pp. 190–198, IEEE, 2012.

[108] M. Mazzucco and M. Dumas, "Achieving performance and availability guarantees with spot instances," in *13th International Conference on High Performance Computing and Communications (HPCC)*, pp. 296–303, IEEE, 2011.

[109] A. Andrzejak, D. Kondo, and S. Yi, "Decision model for cloud computing under SLA constraints," in *Proceedings of the International Symposium on Modeling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS)*, pp. 257–266, IEEE, 2010.

[110] J. Brevik, D. Nurmi, and R. Wolski, "Predicting bounds on queuing delay for batch-scheduled parallel machines," in *Proceedings of the eleventh ACM SIGPLAN symposium on Principles and practice of parallel programming*, pp. 110–118, ACM, 2006.

[111] J. Gray, "Distributed computing economics," *Queue*, vol. 6, pp. 63–68, May 2008.

[112] A. Szalay, A. Bunn, J. Gray, I. Foster, and I. Raicu, "The importance of data locality in distributed computing applications," in *NSF Workflow Workshop*, 2006.

[113] M. Taifi and J. Shi, "Mapreduce performance evaluation on a private hpc cloud," in *Parallel Processing Workshops (ICPPW), 2012 41st International Conference on*, pp. 606 –607, sept. 2012.

[114] W. Xiaohui, W. W. Li, O. Tatebe, X. Gaochao, H. Liang, and J. Jiubin, "Implementing data aware scheduling in gfarm using lsf scheduler plugin mechanism," in *International Conference on Grid Computing and Applications (GCA'05*, pp. 3–10, 2005.

[115] S. Ibrahim, H. Jin, L. Lu, S. Wu, B. He, and L. Qi, "Leen: Locality/fairness-aware key partitioning for mapreduce in the cloud," in *Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on*, pp. 17–24, 2010.

[116] D. Warneke and O. Kao, "Nephele: efficient parallel data processing in the cloud," in *Proceedings of the 2nd workshop on many-task computing on grids and supercomputers*, p. 8, ACM, 2009.

[117] Z. Guo, G. Fox, and M. Zhou, "Investigation of data locality in mapreduce," in *Cluster, Cloud and Grid Computing (CCGrid), 2012 12th IEEE/ACM International Symposium on*, pp. 419–426, 2012.

[118] R. Castro, M. Coates, G. Liang, R. Nowak, and B. Yu, "Network tomography: recent developments," *Statistical science*, vol. 19, pp. 499–517, 2004.

[119] Y. Vardi, "Network tomography: Estimating source-destination traffic intensities from link data," *Journal of the American Statistical Association*, vol. 91, no. 433, pp. pp. 365–377, 1996.

[120] N. G. Duffield, F. Lo Presti, V. Paxson, and D. Towsley, "Inferring link loss using striped unicast probes," in *INFOCOM 2001. Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies*, vol. 2, pp. 915–923, 2001.

[121] M. Coates and R. Nowak, "Network loss inference using unicast end-to-end measurement," in *Proceedings of the ITC Conference on IP Traffic, Modelling and Management*, pp. 28–1, 2000.

[122] M. Coates, R. Castro, R. Nowak, M. Gadhiok, R. King, and Y. Tsang, "Maximum likelihood network topology identification from edge-based unicast

measurements," in *Proceedings of the 2002 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, pp. 11–20, 2002.

[123] N. T. Karonis, B. R. De Supinski, I. Foster, W. Gropp, E. Lusk, and J. Bresnahan, "Exploiting hierarchy in parallel computer networks to optimize collective operation performance," in *Parallel and Distributed Processing Symposium, 2000. IPDPS 2000. Proceedings. 14th International*, pp. 377–384, IEEE, 2000.

[124] I. Foster and N. Karonis, "A Grid-enabled MPI: Message passing in heterogeneous distributed computing systems," in *Supercomputing, 1998.SC98. IEEE/ACM Conference on*, pp. 46–46, 1998.

[125] N. T. Karonis, B. Toonen, and I. Foster, "MPICH-G2: A Grid-enabled implementation of the message passing interface," *Journal of Parallel and Distributed Computing*, vol. 63, no. 5, pp. 551–563, 2003.

[126] R. Cáceres, N. G. Duffield, J. Horowitz, and D. F. Towsley, "Multicast-based inference of network-internal loss characteristics," *Information Theory, IEEE Transactions on*, vol. 45, no. 7, pp. 2462–2480, 1999.

[127] F. Lo Presti, N. Duffield, J. Horowitz, and D. Towsley, "Multicast-based inference of network-internal delay distributions," *Networking, IEEE/ACM Transactions on*, vol. 10, no. 6, pp. 761–775, 2002.

[128] Y. Tsang, M. Yildiz, P. Barford, and R. Nowak, "Network radar: tomography from round trip time measurements," in *Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*, pp. 175–180, 2004.

[129] J. Bresnahan, M. Link, R. Kettimuthu, and I. Foster, "Udt as an alternative transport protocol for gridftp," in *International Workshop on Protocols for Future, Large-Scale and Diverse Network Transports (PFLDNeT)*, pp. 21–22, Citeseer, 2009.

[130] 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, pp. 757–768, 1999.

[131] E. Blanton, S. Fahmy, G. Frederickson, and S. Gangam, "On the cost of network inference mechanisms," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 22, no. 4, pp. 662–672, 2011.

[132] B. Yao, R. Viswanathan, F. Chang, and D. Waddington, "Topology inference in the presence of anonymous routers," in *INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications. IEEE Societies*, vol. 1, pp. 353–363, IEEE, 2003.

[133] D. Battré, N. Frejnik, S. Goel, O. Kao, and D. Warneke, "Evaluation of network topology inference in opaque compute clouds through end-to-end measurements," in *IEEE International Conference on Cloud Computing (CLOUD)*, pp. 17 –24, 2011.

[134] D. F. Robinson and L. R. Foulds, "Comparison of phylogenetic trees," *Mathematical Biosciences*, vol. 53, no. 1-2, pp. 131–147, 1981.

[135] Y. Gong, B. He, and J. Zhong, "Network performance aware MPI collective communication operations in the cloud," *IEEE Transactions on Parallel and Distributed Systems*, vol. 99, no. 1, p. 11, 2013.

[136] W. Gropp, E. Lusk, D. Ashton, P. Balaji, D. Buntinas, R. Butler, A. Chan, D. Goodell, J. Krishna, G. Mercier, *et al.*, "MPICH2 users guide," *Mathematics and Computer Science Division-Argonne National Laboratory, Version 0.4*, 2005.

[137] T. A. Said and O. Rana, "Implementing migration-aware virtual machines," in *Cyber Security and Cloud Computing (CSCloud), 2015 IEEE 2nd International Conference on*, pp. 54–61, IEEE, 2015.

[138] B. Ciciani, D. Didona, P. D. Sanzo, R. Palmieri, S. Peluso, F. Quaglia, and P. Romano, "Automated workload characterization in cloud-based transactional data grids," in *26th International Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW)*, pp. 1525–1533, IEEE, 2012.

[139] W. Tang, N. Desai, D. Buettner, and Z. Lan, "Job scheduling with adjusted runtime estimates on production supercomputers," *Journal of Parallel and Distributed Computing*, vol. 73, no. 7, pp. 926–938, 2013.

[140] R. Gibbons, "A historical application profiler for use by parallel schedulers," in *Job scheduling strategies for parallel processing*, pp. 58–77, Springer, 1997.

[141] A. B. Downey, "Predicting queue times on space-sharing parallel computers," in *Parallel Processing Symposium, 1997. Proceedings., 11th International*, pp. 209–218, IEEE, 1997.

[142] W. Smith, I. Foster, and V. Taylor, "Predicting application run times using historical information," in *Job Scheduling Strategies for Parallel Processing*, pp. 122–142, Springer, 1998.

[143] L. Yang, X. Ma, and F. Mueller, "Cross-platform performance prediction of parallel applications using partial execution," in *Proceedings of the ACM/IEEE Supercomputing Conference*, pp. 40–40, Nov 2005.

[144] O. Tickoo, R. Iyer, R. Illikkal, and D. Newell, "Modeling virtual machine performance: challenges and approaches," *ACM SIGMETRICS Performance Evaluation Review*, vol. 37, no. 3, pp. 55–60, 2010.

[145] P. Barham, R. Isaacs, R. Mortier, and D. Narayanan, "Magpie: Online modelling and performance-aware systems.," in *HotOS*, pp. 85–90, 2003.

[146] P. Barham, A. Donnelly, R. Isaacs, and R. Mortier, "Using magpie for request extraction and workload modelling.," in *OSDI TODO*, vol. 4, pp. 18–18, 2004.

[147] A. Verma, L. Cherkasova, and R. H. Campbell, "ARIA: Automatic resource inference and allocation for mapreduce environments," in *Proceedings of the 8th ACM International Conference on Autonomic Computing (ICAC)*, pp. 235–244, ACM, 2011.

[148] E. Elmroth and J. Tordsson, "Grid resource brokering algorithms enabling advance reservations and resource selection based on performance predictions," *Future Generation Computer Systems*, vol. 24, no. 6, pp. 585–593, 2008.

[149] O. DeMasi, T. Samak, and D. H. Bailey, "Identifying HPC codes via performance logs and machine learning," in *Proceedings of the first workshop on Changing landscapes in HPC security*, pp. 23–30, ACM, 2013.

[150] J. Zhang and R. Figueiredo, "Application classification through monitoring and learning of resource consumption patterns," in *Proceedings of the 20th International Parallel and Distributed Processing Symposium (IPDPS)*, p. 10 pp, April 2006.

[151] T. Wood, L. Cherkasova, K. Ozonat, and P. Shenoy, "Profiling and modeling resource usage of virtualized applications," in *Proceedings of the 9th ACM/IFIP/USENIX International Conference on Middleware*, pp. 366–387, 2008.

[152] Y. Becerra, D. Carrera, and E. Ayguade, "Batch job profiling and adaptive profile enforcement for virtualized environments," in *Proceedings of hte 17th Euromicro International Conference on Parallel, Distributed and Network-based Processing*, pp. 414–418, Feb 2009.

[153] H.-L. Truong, T. Fahringer, and S. Dustdar, "Dynamic instrumentation, performance monitoring and analysis of grid scientific workflows," *Journal of Grid Computing*, vol. 3, no. 1-2, pp. 1–18, 2005.

[154] D. G. Feitelson, *Workload modeling for computer systems performance evaluation*. Cambridge University Press, 2015.

[155] "Amazon CloudWatch." https://aws.amazon.com/cloudwatch/, Accessed on April 2016.

[156] R. Buyya, J. Broberg, and A. M. Goscinski, *Cloud computing: principles and paradigms*, vol. 87. John Wiley & Sons, 2010.

[157] P. Leitner, C. Inzinger, W. Hummer, B. Satzger, and S. Dustdar, "Application-level performance monitoring of cloud services based on the complex event processing paradigm," in *Proceedings of the 5th International Conference on Service-Oriented Computing and Applications (SOCA)*, pp. 1–8, IEEE, 2012.

[158] C. Voss, N. Tsikriktsis, and M. Frohlich, "Case research in operations management," *International journal of operations & production management*, vol. 22, no. 2, pp. 195–219, 2002.

[159] V. Bashkirov, R. Schulte, G. Coutrakon, B. Erdelyi, K. Wong, H. Sadrozinski, S. Penfold, A. Rosenfeld, S. McAllister, and K. Schubert, "Development of proton computed tomography for applications in proton therapy," in *AIP Conference Proceedings*, vol. 1099, p. 460, 2009.

[160] R. W. Schulte, V. Bashkirov, M. C. L. Klock, T. Li, A. J. Wroe, I. Evseev, D. C. Williams, and T. Satogata, "Density resolution of proton computed tomography," *Medical physics*, vol. 32, p. 1035, 2005.

[161] S. Penfold, *Image reconstruction and Monte Carlo simulations in the development of proton computed Tomography for applications in proton radiation therapy*. PhD thesis, University of Wollongong, 2010.

[162] R. T. Fielding and R. N. Taylor, "Principled design of the modern web architecture," *ACM Transactions on Internet Technology (TOIT)*, vol. 2, no. 2, pp. 115–150, 2002.

[163] N. T. Karonis, K. L. Duffin, C. E. Ordoñez, B. Erdelyi, T. D. Uram, E. C. Olson, G. Coutrakon, and M. E. Papka, "Distributed and hardware accelerated computing for clinical medical imaging using proton computed tomography (pCT)," *Journal of Parallel and Distributed Computing*, vol. 73, no. 12, pp. 1605–1612, 2013.

[164] InfiniBand Trade Association and others, *InfiniBand Architecture Specification: Release 1.0*. InfiniBand Trade Association, 2000.

[165] A. Connolly, S. Habib, A. Szalay, J. Borrill, G. Fuller, N. Gnedin, K. Heitmann, D. Jacobs, D. Lamb, T. Mezzacappa, *et al.*, "Snowmass computing frontier: Computing for the cosmic frontier, astrophysics, and cosmology," *arXiv preprint arXiv:1311.2841*, 2013.

[166] S. Cholia, D. Skinner, and J. Boverhof, "NEWT: A RESTful service for building high performance computing web applications," in *Proceedings of the 5th Gateway Computing Environments Workshop (GCE)*, pp. 1–11, IEEE, 2010.

[167] K. Heitmann, D. Higdon, M. White, S. Habib, B. J. Williams, E. Lawrence, and C. Wagner, "The coyote universe. ii. cosmological models and precision emulation of the nonlinear matter power spectrum," *The Astrophysical Journal*, vol. 705, no. 1, p. 156, 2009.

[168] M. Owens and G. Allen, *SQLite*. Springer, 2010.

[169] "Shibboleth." `https://shibboleth.net/`. Accessed: 2015-11-13.

[170] K. Chard, M. Lidman, B. McCollam, J. Bryan, R. Ananthakrishnan, S. Tuecke, and I. Foster, "Globus Nexus: A platform-as-a-service provider of research identity, profile, and group management," *Future Generation Computer Systems*, pp. –, 2015.

[171] M. Wilde, I. Foster, K. Iskra, P. Beckman, Z. Zhang, A. Espinosa, M. Hategan, B. Clifford, and I. Raicu, "Parallel scripting for applications at the petascale and beyond," *Computer*, vol. 42, no. 11, pp. 50–60, 2009.

[172] A. Bestavros, J. W. Byers, and K. A. Harfoush, "Inference and labeling of metric-induced network topologies," *IEEE Transactions on Parallel and Distributed Systems*, vol. 16, no. 10, pp. 1053–1065, 2005.

[173] Y. Tsang, M. Coates, and R. D. Nowak, "Network delay tomography," *IEEE Transactions on Signal Processing*, vol. 51, no. 8, pp. 2125–2136, 2003.

[174] C. Reich, K. Bubendorfer, M. Banholzer, and R. Buyya, "A SLA-oriented management of containers for hosting stateful web services," in *Proceedings of the 3rd IEEE International Conference on e-Science and Grid Computing*, pp. 85–92, 2007.

[175] C. Reich, M. Banholzer, R. Buyya, and K. Bubendorfer, "Engineering an autonomic container for WSRF-based web services," in *International Conference on Advanced Computing and Communications, (ADCOM)*, pp. 277–282, 2007.

[176] Y. Gu, G. Jiang, V. Singh, and Y. Zhang, "Optimal probing for unicast network delay tomography," in *Proceedings of INFOCOM*, pp. 1–9, IEEE, 2010.

[177] P. Qin, B. Dai, B. Huang, G. Xu, and K. Wu, "A survey on network tomography with network coding," *IEEE Communications Surveys Tutorials*, vol. 16, pp. 1981–1995, September 2014.

[178] L. Bai and S. Roy, "A two-stage approach for network monitoring," *Journal of network and systems management*, vol. 21, no. 2, pp. 238–263, 2013.

[179] "Cloud-init." `http://cloudinit.readthedocs.io/en/latest/`. Accessed: 2015-11-13.

[180] K. P. Burnham and D. R. Anderson, *Model selection and multimodel inference: a practical information-theoretic approach*. Springer, 2002.

[181] T. W. Arnold, "Uninformative parameters and model selection using Akaike's information criterion," *The Journal of Wildlife Management*, vol. 74, no. 6, pp. 1175–1178, 2010.

[182] K.-Y. Liang and S. L. Zeger, "Longitudinal data analysis using generalized linear models," *Biometrika*, vol. 73, no. 1, pp. 13–22, 1986.

[183] "The Gluster web site." http://www.gluster.org/, Accessed on May 2015.

[184] M. Livny, J. Basney, R. Raman, and T. Tannenbaum, "Mechanisms for high throughput computing," *SPEEDUP journal*, vol. 11, no. 1, pp. 36–40, 1997.

[185] "Docker." `https://www.docker.com/`. Accessed: 2015-11-13.

[186] The Performance Co-Pilot Development Team, *Performance Co-Pilot User's and Administrator's Guide*. Silicon Graphics, Inc., 2016.

[187] "Influxdata." `https://influxdata.com/`. Accessed: 2015-11-13.

[188] "Container Advisor (cAdvisor)." `https://github.com/google/cadvisor`. Accessed: 2015-11-13.

[189] S. Andrews *et al.*, "FastQC: A quality control tool for high throughput sequence data," *Reference Source*, 2010.

[190] "PICARD mark duplicates." `http://broadinstitute.github.io/picard/`. Accessed: 2015-11-13.

[191] H. Li and R. Durbin, "Fast and accurate short read alignment with Burrows–Wheeler transform," *Bioinformatics*, vol. 25, no. 14, pp. 1754–1760, 2009.

[192] B. Langmead and S. L. Salzberg, "Fast gapped-read alignment with Bowtie 2," *Nature methods*, vol. 9, no. 4, pp. 357–359, 2012.

[193] H. Li, "Aligning sequence reads, clone sequences and assembly contigs with BWA-MEM," *arXiv preprint arXiv:1303.3997*, 2013.

[194] V. K. Vavilapalli, A. C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth, *et al.*, "Apache hadoop yarn: Yet another resource negotiator," in *Proceedings of the 4th annual Symposium on Cloud Computing*, p. 5, ACM, 2013.

[195] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The hadoop distributed file system," in *26th Symposium on Mass Storage Systems and Technologies (MSST)*, pp. 1–10, IEEE, 2010.

[196] G. Juve and E. Deelman, "Automating application deployment in infrastructure clouds," in *Third International Conference on Cloud Computing Technology and Science (CloudCom)*, pp. 658–665, IEEE, 2011.

[197] J. Kirschnick, J. M. A. Calero, L. Wilcock, and N. Edwards, "Toward an architecture for the automated provisioning of cloud services," *Communications Magazine*, vol. 48, no. 12, pp. 124–131, 2010.

[198] "GitHub." https://github.com/ryanchard/cloud-provisioner, Accessed on April 2016.