Scheduling and Resource Provisioning Algorithms for Scientific Workflows on Commercial Clouds

by

Vahid Arabnejad

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
2018

Abstract

Basic science is becoming ever more computationally intensive, increasing the need for large-scale compute and storage resources, be they within a High Performance Computer cluster, or more recently, within the cloud. Commercial clouds have increasingly become a viable platform for hosting scientific analyses and computation due to their elasticity, recent introduction of specialist hardware, and pay-as-you-go cost model. This computing paradigm therefore presents a low capital and low barrier alternative to operating dedicated eScience infrastructure. Indeed, commercial clouds now enable universal access to capabilities previously available to only large well funded research groups. While the potential benefits of cloud computing are clear, there are still significant technical hurdles associated with obtaining the best execution efficiency whilst trading off cost. In most cases, large scale scientific computation is represented as a workflow for scheduling and runtime provisioning. Such scheduling becomes an even more challenging problem on cloud systems due to the dynamic nature of the cloud, in particular, the elasticity, the pricing models (both static and dynamic), the non-homogeneous resource types and the vast array of services. This mapping of workflow tasks onto a set of provisioned instances is an example of the general scheduling problem and is NP-complete. In addition, certain runtime constraints, the most typical being the cost of the computation and the time which that computation requires to complete, must be met. This thesis addresses 'the scientific workflow scheduling problem in cloud', which is to schedule workflow tasks on cloud resources in a way that users meet their defined constraints such as budget and deadline, and providers maximize profits and resource utilization. Moreover, it explores different mechanisms and strategies for distributing defined constraints over a workflow and investigate its impact on the overall cost of the resulting schedule.

Acknowledgments

First and foremost, I offer my profoundest gratitude to my supervisor, Kris Bubendorfer, for his invaluable guidance, advice, and motivation throughout my candidature. Secondly, I would like to offer my sincere gratitude to my co-supervisor, Bryan Ng, whose wise advice and profound knowledge has made the contributions of this thesis more significant.

I am greatly appreciative for all of the support from my friends and family who have helped me through my tough times and filled me with confidence and strength when I needed the most.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Problem Statement: Workflow Scheduling and Resource Provisioning in Cloud

Scientific discovery is in the midst of a disruptive technological change, where experimental and observational research is being transformed by computational and data-intensive approaches [1]. Researchers in almost every discipline now face new opportunities [2] and challenges that impact every stage of the research lifecycle due to ever-growing data volumes and increased analytical complexity [3]. While much of this has in the past utilized dedicated High Performance Computing (HPC) systems, there is an ongoing migration of scientific computing into the various commercial clouds for a number of compelling reasons: elastic clouds offer a variety of accessible and cost-effective computing platforms [4–7], the on-demand model better fits the typically sporadic demands of researchers [8], and finally, rather than resources being used to purchase and maintain dedicated HPC equipment, they are instead used for pay-per-use computation and storage resources offered by cloud vendors. Typical commercial cloud services charge on the basis of the number of hours the resources (such as CPU, network bandwidth and amount of storage) are used. This charging model is referred to as pay-per-use. Other advantages of using

commercial clouds for scientific computation include reliability and fault tolerance, and access to specialized resources such as GPUs. The flexibility inherent in the elastic cloud model, while powerful, may also result in inefficient usage and high costs when inadequate scheduling and provisioning decisions are made [9].

The cloud presents an opportunity to accelerate scientific discovery by automating [10] computation in workflows, permitting vast numbers of complex compute and data-intensive experiments to be executed. A major challenge of the cloud paradigm for e-Science lies in limiting or minimising [9] costs while maintaining or even accelerating throughput. In fact, scheduling workflows and provisioning cloud resources naïvely can have a significant financial penalty - especially in dynamic markets such as the Amazon spot market [11]. As the fundamental workflow scheduling problem is *NP*-complete [12], optimising multiple constraints, such as cost and time, over a non-uniform set of unlimited resources is nontrivial. Indeed, this complexity leads to long computation times in order to create a reasonable schedule, therefore a heuristic scheduling approach is needed.

1.2 Research Challenges

To deal with the challenges associated with the scientific workflow scheduling in cloud, the following research problems are investigated:

- Scheduling: assigning workflow tasks to resources is known as the scheduling problem that belongs to the class of NP-complete problems [12]. Allocating workflow tasks to resources can be separated into two stages, the first being scheduling and the second is provisioning [13]. Given a set of resources, the workflow task scheduling phase aims to determine the optimal execution order and task placement with respect to user and workflow constraints [14, 15].

  Most of the studies in workflow scheduling problem are designed for

users on platforms such as grid and cluster. Pricing schemes in cloud systems are based on lease intervals from the time of provisioning, even if the instance is only used for a fraction of that period. However, cost in grid is calculated based on the accumulated cost of requested services. Therefore, those approaches are not directly applicable for scheduling workflows on cloud when unlimited resources can be leased from a public cloud on-demand. However, at the final stage of writing this thesis, Amazon instance prices have been changed [1].

- Resource Provisioning: The cloud provides on demand pay-per-use provisioning of a range of instance types, no matter where the services or requester are hosted. Dynamic provisioning of resources is a critical element when utilizing the cloud for executing large-scale and complex applications. This is essentially a problem of ensuring an appropriate set of instances is provisioned.

  The resource provisioning phase aims to determine the number and type of resources required and then reserve these resources for workflow execution [9, 16]. While the majority of cloud scheduling systems necessarily include both scheduling and provisioning stages, prior research tends to focus on the scheduling phase, under the assumption that a pre-identified pool of (often homogeneous) resources are used for execution, and with the goal of optimizing workflow execution time (makespan) without considering resource cost. In a commercial cloud environment this set of assumptions no longer holds.

- User Requirements: Research is increasingly reliant on big compute and big data, the fusion of which is known as data-intensive science. Data-intensive computing is defined as production, manipulation and analysis of data from mega bytes to peta bytes [17]. Data-intensive applications in different domains, from science to social networking, produce large

[1] $https://aws.amazon.com/blogs/aws/new-per-second-billing-for-ec2-instances-and-ebs-volumes/$

scale data that need to be analyzed and processed with parallel processing and distributed techniques. Data operations consist of loading input files, data processing, distribution and aggregation, and execution is typically modeled and characterized by workflows. While cloud platforms provide enormous elastic computing capacity, they also pose unique multi-objective challenges in respect to cost, time and data movement. Efficiently managing pay-per-use heterogeneous cloud infrastructure for large data and compute intensive applications within user requirements (defined as constraints such as budget and deadline) is a challenge that is faced in almost every research domain.

1.3 Contribution

This thesis presents novel heuristic algorithms for scientific workflow scheduling constrained by user requirements such as deadline and budget on cloud. Specifically, the major contributions of this thesis are:

1. The design and development of two new heuristic algorithms, Proportional Deadline Constrained (PDC) and Deadline Constrained Critical Path (DCCP), that manage the scheduling of workflows on dynamically provisioned cloud resources. The PDC algorithm maximizes parallelism in a workflow by separating it into logical levels and then proportionally subdividing the overall workflow deadline over different levels. Moreover, in order to show how the order of execution can influence the scheduling, particularly the cost, different policies are tested in the PDC algorithm. The DCCP algorithm uses the concept of Constrained Critical Paths (CCP) to execute a set of tasks on the same instance with the goal of reducing communication cost between instances.

2. The investigation of different ways in which to distribute budget and deadline over a workflow. In terms of budget, several new strategies are introduced for sharing or distributing budget based on the dependency

structure inherent in workflows. Moreover, trickling to redistribute unspent budget down to other levels is introduced. The main finding of my research is in the importance of biasing budget distribution to early levels in a workflow.

To distribute deadline, new strategies are introduced for deadline distribution assessed the effectiveness of these strategies in terms of cost and success rate. In general, the strategy that takes into consideration the execution time of each level as well as number of tasks in the level which yields the lowest cost.

3. The design and implementation of a new heuristic scheduling algorithm, Budget Deadline Aware Scheduling (BDAS), which focuses on addressing the unique characteristics of workflow execution on cloud platforms, such as on-demand provisioning and instance heterogeneity, while simultaneously meeting budget and deadline constraints. The novelty of this work is satisfying both budget and deadline constraints while introducing a tunable cost-time trade-off over heterogeneous instances. In addition, the stability and robustness of the presented algorithm is studied by performing sensitivity analysis.

4. The design of a Workflow as a Service (WFaaS) architecture for scheduling of multiple workflows with the aim of satisfying the deadline for each workflow in a typical cloud environment in which workflow can be submitted at any time.

1.4 Thesis Organization

The core chapters of this thesis have resulted in a set of papers published during the PhD candidature.

- **Chapter 2: Literature Review**

This chapter covers essential background and provides a detailed taxonomy on scheduling concept. This chapter then reviews related works and summarizes the research topics of workflow scheduling in cloud.

- **Chapter 3: Motivation and Problem Statement**

  This chapter describes a number of challenges related to scientific workflow scheduling that need to be considered. Then the problem of workflow scheduling and related definitions are explained.

- **Chapter 4: Deadline Constrained Workflow Scheduling**

  This chapter presents two cost-effective and deadline constrained heuristics for scheduling scientific workflow. Both heuristics consider a tradeoff of cost vs time, and includes both re-use of pre-provisioned instances and the creation of new instances on demand.

  The chapter is derived from the following publications:

  - **Vahid Arabnejad**, Kris Bubendorfer, Bryan Ng. Scheduling Deadline Constrained Scientific Workflows on Dynamically Provisioned Cloud Resources. Future Generation Computer Systems (FGCS), 2017.
  - **Vahid Arabnejad**, Kris Bubendorfer, Bryan Ng, and Kyle Chard. A deadline constrained critical path heuristic for cost-effectively scheduling workflows. In the 8th IEEE International Conference on Utility and Cloud Computing (UCC), Limassol, Cyprus, December 2015.
  - **Vahid Arabnejad** and Kris Bubendorfer. Cost effective and deadline constrained scientific workflow scheduling for commercial clouds. In the 14th IEEE International Symposium on Network Computing and Applications (NCA), Cambridge, MA USA, September 2015.

- **Chapter 5: Distribution Strategies for Scientific Workflow Scheduling**

  This chapter discusses the importance of distribution of defined constrained, budget and deadline, in terms of cost minimization, time efficiency and

success rate. Various strategies are introduced for sharing or distributing budget and deadline to different levels in a workflow.

The chapter is derived from the following publications:

- **Vahid Arabnejad**, Kris Bubendorfer, Bryan Ng. Budget Distribution Strategies for Scientific Workflow Scheduling in Commercial Clouds. In the proceeding of the IEEE 12th International Conference on eScience, Baltimore, Maryland, USA, October 2016.
- **Vahid Arabnejad**, Kris Bubendorfer, Bryan Ng. Deadline Distribution Strategies for Scientific Workflow Scheduling in Commercial Clouds. In the 9th IEEE/ACM International Conference on Utility and Cloud Computing (UCC 2016), December 6-9, 2016 Shanghai, China.
- **Vahid Arabnejad**, Kris Bubendorfer, Bryan Ng. A Budget-Aware algorithm for Scheduling Scientific Workflows in Cloud. In the the 18th IEEE International Conferences on High Performance Computing and Communications (HPCC), 12-14 December, Sydney, Australia.

- **Chapter 6: Budget Deadline Constrained Workflow Scheduling**

  When more than one constraint is requested by a user, such as budget and deadline, the problem of workflow scheduling becomes even more challenging. The main reason is conflicting behavior of these constraints, like running on faster instances leads to shorter execution time while it costs more. Hence, in order to analyze an acceptable balance between incompatible constraints, cost and time in cloud systems, this chapter presents a new heuristic scheduling algorithm for scheduling workflows constrained by both budget and deadline.

  The chapter is derived from the following publication:

  - **Vahid Arabnejad**, Kris Bubendorfer, Bryan Ng. Budget and Deadline Aware e-Science Workflow Scheduling in Cloud. IEEE Transaction of Parallel and Distributed System (TPDS), 2018.

- **Chapter 7: Dynamic Workflow Scheduling**

  Previous chapters described static scheduling as when the number of workflows are known in advance and all are submitted at the same time. However, a scheduler may have to schedule an unpredictable stream of workflows. This chapter presents a new algorithm to address scheduling of multiple workflows with the aim of satisfying the deadline for each workflow in a cloud environment in which workflows can be submitted at any time.

  The chapter is derived from the following publication:

  – **Vahid Arabnejad**, Kris Bubendorfer, Bryan Ng. Dynamic Workflow Scheduling: A Deadline and Cost-Aware Approach for Commercial Clouds. Submitted to Future Generation Computer Systems (FGCS).

Chapter 2

Workflow Scheduling: Taxonomy and Literature Review

In order to fully explain the related work in this area, I present a taxonomy that covers different criteria and parameters for scheduling problems in distributed computing platforms. This taxonomy (Figure 2.1) provides a broad summary of the different aspects and techniques in workflow scheduling. Further, recent studies that were specifically designed for scheduling scientific workflows in cloud, are categorized, compared and discussed in more detail. Numbers appearing in brackets in Figure 2.1 map sections in this chapter.

2.1 Application Model

In general, application model for scheduling in a distributed system can be considered as Bag of Tasks (BoT) in which all tasks are independent, or workflows with defined dependencies between tasks. The application model leaf of the taxonomy is presented in Figure 2.2.

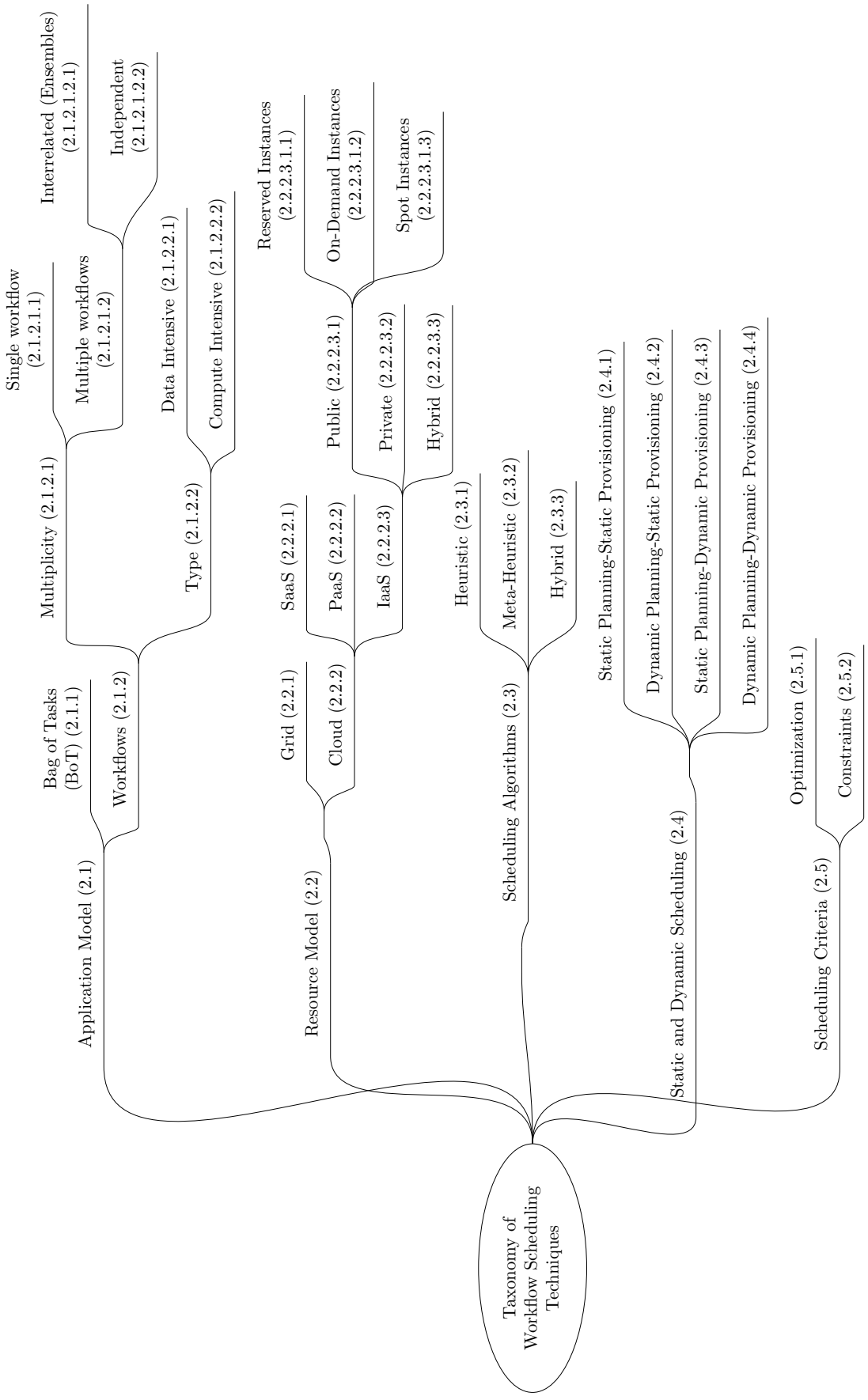


Figure 2.1: Taxonomy of Workflow Scheduling techniques

2.1.1 Bag of Tasks (BoT)

This type of application is composed of sequential and completely independent tasks. Each task for execution needs one or more input files, and input files can be shared among tasks. Likewise, each task can generate one or more outputs [18]. This means tasks can be executed in any order as there is no data communication between tasks. There are many important BoT applications, including data mining algorithms, massive searches (such as key breaking), parameter sweeps, Monte Carlo simulations, fractals calculations (such as Mandelbrot), and image manipulation applications (such as tomographic reconstruction). Comparison of different algorithms for BoT applications are presented in [19] and [20].

2.1.2 Workflows

Workflows can be used in various fields of scientific applications and analysis, therefore they are generally applied to model calculations and computations in a wide range of scientific applications, including astronomy, bioinformatics, earthquake science, and gravitational wave physics [21]. For example, an astronomical Mosaic Engine (Montage) workflow [22] was generated by NASA/IPAC. It was designed to create custom mosaics images of the sky by astronomers using input images in the Flexible Image Transport System (FITS) format.

2.1.2.1 Multiplicity

One of the main differences between presented algorithms for workflow scheduling is their ability to accept and schedule either a single workflow or multiple workflows.

2.1.2.1.1 Single workflow Designed algorithms in traditional models such as grid usually accept a single workflow and try to optimize a single objective

or multiple objectives. Each task in a workflow displays the execution of a computational process. This process can be like running a function, requesting a service, a query that is submitted to a database, or a job that is submitted to cloud or grid on the Internet to use a remote resource.

2.1.2.1.2 Multiple Workflows From the multiplicity vision, algorithms are designed to receive either interrelated or independent workflows.

2.1.2.1.2.1 Interrelated Workflows Some scientific experiments are grouped into interrelated workflows that are known as ensembles. Workflows in an ensemble typically have a similar structure, but they differ in their input data, number of tasks and individual task sizes [23]. One of the examples of workflow ensembles is Cybershake [24], which produces seismic hazard maps. A combination of hazard curves generated by each workflow in an ensemble of Cybershake experiments can create a hazard map. As an example, an ensemble of 2288 Cybershake workflows are used to produce hazard maps over 286 sites [1].

2.1.2.1.2.2 Independent Workflows Another type can be considered as independent workflows where users submit workflows at different times. These workflows are isolated from each other and they are not necessarily related to each other. The number and type of workflows are not known in advance, which need to be scheduled dynamically. I will explain more about dynamic algorithms in further sections.

2.1.2.2 Workflow Type

Large-scale scientific analyses are typically represented as workflows, which are the typical model for characterizing e-science experiments in distributed systems. Scientific workflows vary in size from a couple of tasks to thousands

[1] $http://scec.usc.edu/scecpedia/CyberShake\_Study\_13.4/$

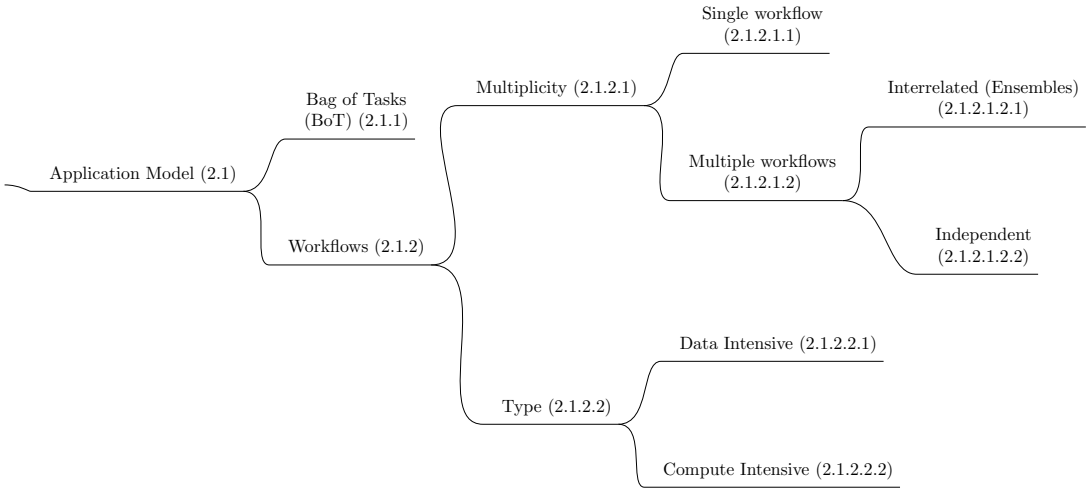


Figure 2.2: Application Model Taxonomy repeated from Figure 2.1

or million of tasks, and usually consist of data-intensive or compute-intensive applications.

2.1.2.2.1 Data-Intensive Workflows Basic research and, consequently, scientific discovery, are in the midst of a disruptive transformation [1]. Research is increasingly reliant on big compute and big data, the fusion of which is known as data-intensive science. The data-intensive workflows deal with large amounts of data in the range of megabytes to petabytes while spending most of their time performing I/O activities. An example of data-intensive experiments is the field of high energy physics. For example, in 2010, the Large Hadron Collider (LHC) produced about 13 petabytes of data [25]. To process this amount of data, almost 140 computing centers in 34 countries had collaborated.

2.1.2.2.2 Compute-Intensive Workflows Large-scale applications mainly consist of complex tasks requiring high performance computing instances are called compute-intensive workflows. In these applications, the majority of the tasks execution time deals with computation rather than I/O.

2.2 Resource Model

The distributed computing is dedicated to many projects for solving large-scale problems by using the power of a vast number of resources distributed amongst a network. Distributed computing paradigm has seen different technologies, started from desktop computing, through grid computing, and now to cloud computing. This section discusses two main resource models in distributed computing. Figure 2.3 outlines the resource model leaf, based on the taxonomy presented in 2.1.

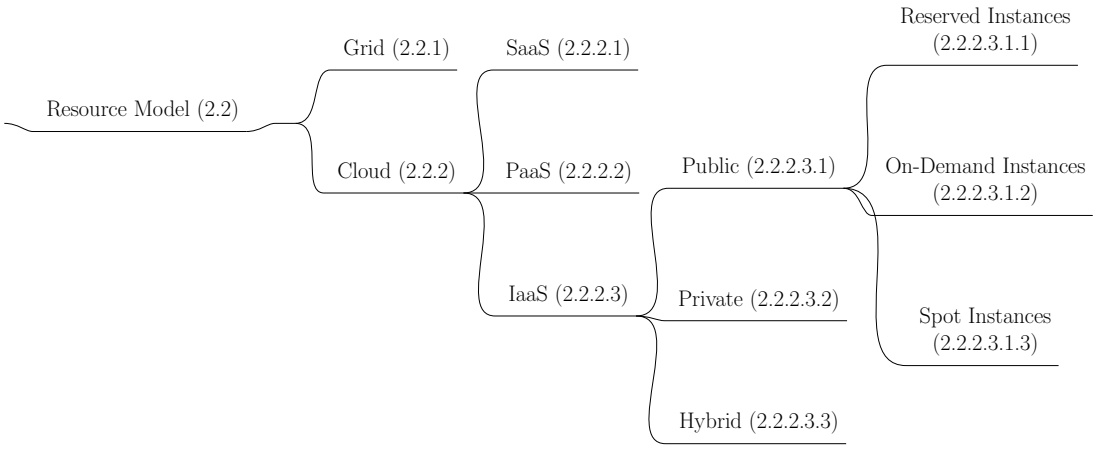


Figure 2.3: Resource Model Taxonomy repeated from Figure 2.1

2.2.1 Grid Computing Platforms

Grid provides accessing to remote and diverse high performance resources for solving large-scale problems in different domains in science. These resources, which are scalable, sharable, secure, heterogeneous and geographically distributed, are accessible as computing utilities to end users [26]. Moreover, these resources can be considered as computational abilities, data storages, software services and applications. Most resources in grids are geographically distributed and managed by different organizations. This raises some challenges in terms

of resource management and scheduling for solving large-scale problems. Some comprehensive surveys that discussed these challenges are presented in [27,28].

2.2.2 Cloud Computing Platforms

Cloud computing is the latest of computing paradigms for the delivery of resources, platforms, or applications to customers that are used in different domains, such as science, to cope with challenging issues. A comprehensive definition of cloud Computing is given by the National Institute of Standards and Technology (NIST) [29]; cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. NIST considers five essential characteristics for this cloud model:

- on-demand self-service: Users can provision computing resources, without any human interaction need from the service provider.
- broad network access: Users must be able to access resources over the network with standard tools such as mobile phones, tablets, laptops, and workstations.
- resource pooling: With a pooling strategy, multiple users can use computing resources using a multi-tenant model. Based on user demands, different physical and virtual resources can be assigned and reassigned dynamically. The customers usually do not have any information or control over the exact location of the resource provider. They could know about location in a higher level of abstraction, i.e, data-center, state, or country.
- rapid elasticity: One of the remarkable properties, which makes clouds distinct from other models such as grid, is resource elasticity. Infrastructures can be scaled up and scaled down in minutes dynamically with vari-

ation of demands. This characteristic gives a user the impression of infinite capacity of resources and available at any time in cloud.

- measured service: Resource usage can be monitored, reported, and controlled, providing transparency for participants, the consumer and the provider about the utilized service.

The most important difference between cloud and grid computing is the illusion of unlimited resources, where users can request as many resources as needed and pay for it.

The architecture of cloud computing is represented in Fig. 2.4 [30]. Three

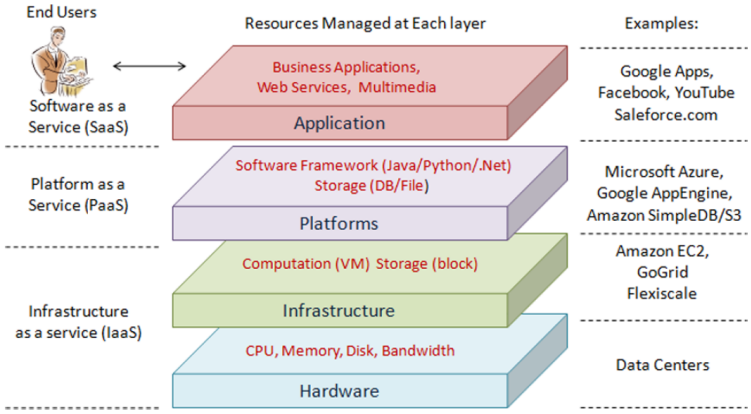


Figure 2.4: cloud computing architecture

different services are provided by cloud services:

2.2.2.1 Software as a Service (SaaS)

This service makes programs and applications accessible as a service from different customer devices from either a thin client interface, such as a web browser to users. SaaS enables users to obtain software on a subscription basis, e.g. monthly or yearly. In SaaS, users are not responsible for installing, running and managing of the applications. Google Apps and Salesforce.com are distinguished examples of this model.

2.2.2.2 Platform as a Service (PaaS)

PaaS enables users to deploy their applications on the cloud. Users' applications are created by applying different tools and programming languages that are supported by the cloud provider. The user cannot control or manage the underlying cloud infrastructure, including network, servers, operating systems, or storage, but does have control over the deployed applications and possibly application hosting environment configurations [29]. Some such examples of PaaS are Google App Engine, Microsoft Azure, Force.com.

2.2.2.3 Infrastructure as a Service (IaaS)

This service provides hardware, storage, servers and data-center space for users that usually uses the virtualization technique. In simple terms, users can lease virtual servers and storage as a service over the web. IaaS allows users in anywhere on Earth to use resources through the Internet, which proves the flexibility of cloud infrastructure. Users can deploy and run their applications with no need to control or manage the physical infrastructure. Amazon Simple Storage Service (S3) and Amazon Elastic Compute Cloud (EC2) are the outstanding instances of IaaS.

IaaS can be classified in three categories in terms of resource availability:

2.2.2.3.1 Public Cloud Resources such as computing services and storage are available to the general public over the Internet. The general public is described as either individual users or corporations. Public cloud services may be free or offered on a pay-per-usage model. Because of shared infrastructure, using public cloud can have cost-saving effects for organizations. IaaS providers offer different resource types with diverse pricing options. In this section, a brief discussion on different pricing strategies on these types is presented:

2.2.2.3.1.1 Reserved instances Users pay in advance to reserve instances upfront for a specified time intervals, for example a month or even a year. Some

cloud providers offer discounts on instance prices, referred to as the reservation option [31], if they are reserved in advance for a period of times to harvest long-term risk-free revenue. Therefore, if their demand is steady and predictable, reserved instances can achieve significant cost saving for executing applications, especially if resources can be fully utilized.

2.2.2.3.1.2 On-demand instances One of the main aims of cloud is providing on-demand access to services similar to public utility services such as electricity, water and gas [32]. The on-demand instances (also known as usage based model) allow users to rent resources on a pay-as-you-go basis according to the amount of consumed services. Compared to reserved instances, on-demand instances need a per time unit usage cost, which is more expensive, with the same configurations. In this type, providers usually charge users on a fixed rate basis. For example, Amazon EC2 instances are charged on an hourly interval from the time of provisioning, even if the instance is only used for a fraction of that period.

2.2.2.3.1.3 Spot instances Spot instances are sold based on a dynamic pricing (also called auction based) by bidding on spare and unused capacity of resources with the aim of maximizing providers' revenue. Spot instances price vary, based on users' supply and demand. If a user bid on a resource is above the current spot price, the resource is allocated to the user. However, if the current price at any time goes beyond the bid price, the resource is terminated. Therefore, users can run their jobs on spot instances until their bid exceeds the current price. Indeed, a trade-off between reliability and cost is applied in spot instances.

For users that have enough time for executing their workflows, spot instances can remarkably reduce the cost compared to on-demand instances. Compute-intensive workflows are the best candidates to run on spot instance as they are divisible. Due to the fact that these kinds of workflows deal with computation rather than I/O, failure-prone spot instances are more beneficial for execution

in terms of cost saving.

2.2.2.3.2 Private Cloud The cloud infrastructure operated solely for an organization that can be accessed by individuals whether managed internally or by a third-party and hosted internally or externally [29]. Private cloud benefits the advantages of public cloud, while keeping more control over data and process to organizations.

2.2.2.3.3 Hybrid Cloud A composition of two or more distinct cloud infrastructures (private or public) that remain unique entities, but are bound together by standardized or proprietary technology that enables data and application portability (e.g., cloud bursting for load balancing between clouds) [29]. An example of hybrid cloud deployment can be an organization that deploys crucial and classified applications in a private cloud while keeping unessential applications in the public cloud.

2.3 Scheduling Algorithms

Scheduling workflow tasks to resources while meeting workflow dependencies and constraints is known as the Workflow Scheduling Problem (WSP) – a class of problem that is known to be NP-complete [12]. In this section, I would like to give a brief introduction to general definition of scheduling algorithm types, and in further sections more discussion of each type with relevant studies will be provided.

2.3.1 Heuristic-Based Algorithms

Multi-objective problems tend to be extremely complex and depend on analysis of large datasets [33]. These problems face many conflicting objectives that must be optimized concurrently to obtain high-quality solutions. Due to time

and space complexity constraints of multi-objective problems, no optimal solution can be found in polynomial time [34]. Algorithms that are capable of finding a near optimal or approximate solution at a reasonable computation cost without being able to guarantee optimality, and possibly not feasibility for such problems, are called heuristic algorithms [35].

Most heuristic-based solutions for workflow scheduling belong to the list-based scheduling category [36] and consist of two main stages: firstly, task prioritization, and secondly, task assignment. In the first stage, a rank value is assigned to each task and then all tasks are sorted based on their rank. In the task assignment phase, tasks are allocated to suitable instances. More details on these approaches will be discussed in further sections.

Another group of heuristics for solving the workflow scheduling problem are clustering approaches. These methods usually consist of two stages, which are grouping tasks to clusters, and prioritizing tasks in the same cluster. Each cluster is a subset of workflow and schedules on a distinct resource. The main goal of clustering based approach is to avoid communication cost between tasks by executing in the same instance. Therefore, there is a conflict between reducing communication cost and maximizing parallelism.

2.3.2 Meta-Heuristic (Evolutionary) Algorithms

Heuristic algorithms are problem-dependent that usually find acceptable solutions in a reasonable amount of time. On the other hand, meta-heuristics are general-purpose and problem-independent techniques that can be applied to solve almost any optimization problem.

Some successful meta-heuristics are inspired by nature. Methods that apply the biological evolution ideas to find a solution of an optimization problem are called evolutionary algorithms. Most well-known evolutionary algorithms, including Genetic Algorithm(GA), Ant Colony Optimization(ACO) and Particle Swarm Optimization (PSO), are used to tackle the workflow scheduling problem with multiple constraints. Performance of meta-heuristics methods

are compared in [37,38], concluding that PSO outperforms other methods such as GA and ACO in most cases.

While search-based and meta-heuristic strategies produce acceptable answers, they are usually time-consuming algorithms based on their need for an initialization phase to obtain high-quality solutions. Therefore, due to the NP-complete nature of workflow scheduling [12], the time needed in respect to dimensions of the problem increases exponentially by using meta-heuristics. Another important point that needs to be considered in these algorithms is lack of ability to perform well when some changes happen during execution. Moreover, they do not respond well in dynamic scheduling situations because of the required dynamic phase. In summary, the main disadvantages of meta-heuristic are: trapping into local optima, slow convergence rate, long computational time and tuning many parameters [39].

2.3.3 Hybrid Algorithms

A hybrid algorithm is one which combines meta heuristics with other heuristic approaches by integrating them into a single algorithm. The main idea of such algorithms is to leverage the strengths of optimization algorithms [40]. However, these methods use all the search algorithms in every iteration within the convergence phase, leading to high computation time and time complexity.

2.4 Static and Dynamic Scheduling

Given a set of resources, the workflow task scheduling phase aims to determine the optimal execution order and task placement in respect to user and workflow constraints [14, 15]. The resource provisioning phase aims to determine the number and type of resources required and then reserve these resources for workflow execution [9, 16].

In this thesis, studied algorithms can be classified into four types:

2.4.1 Static Planning, Static Provisioning (SPSP)

In this category, the number of workflows is known in advance, which could be single or multiple and are grouped into interrelated workflows (ensembles). Moreover, workflow characteristics, structure, and number of tasks in each workflow are known in advance. The provisioning stage is also static, which means a pre-identified pool of (often homogeneous) resources are used for execution, and with the goal of optimizing workflow execution time (makespan) without considering resource cost. Therefore, algorithms in this category try to schedule tasks over a given static set of resources.

2.4.2 Dynamic Planning, Static Provisioning (DPSP)

Algorithms in this class are designed to receive multiple workflows forming a stream of workflows that arrive at different times. The provisioning phase is static, which means a static instance pool is determined, and the instances remain active while other workflows arrive for execution. The provisioning phase in grid is usually static and algorithms focus only on scheduling phase to optimize user's requirements rather than resource provisioning.

2.4.3 Static Planning, Dynamic Provisioning (SPDP)

Similar in principle to the first category, with the only difference being that an unlimited number of instance types (often heterogeneous) can be provisioned at runtime. Dynamic provisioning is currently supported by most commercial providers.

2.4.4 Dynamic Planning, Dynamic Provisioning (DPDP)

Independent workflows can be submitted at different times, continuously arrived for execution. Therefore, the planning phase is dynamic as no information about the total number of workflows is available in advance.

2.5 Scheduling Criteria

The two most significant classes of workflow scheduling are optimization and QoS constraint scheduling [41]:

2.5.1 Optimization

In optimization scheduling algorithms, minimizing objectives such as time and cost is the final goal. Simple heuristics such as Min-Min, Max-Min and Suffrage [42] are applied in the workflow scheduling problem to find the shortest makespan. The Min-Min algorithm calculates the Minimum Completion Time (MCT) for all resources for all tasks. The task that will be completed in the minimum of time is selected and assigned to the corresponding resource. The Max-Min algorithm is similar to Min-Min, the difference is that the task that is executed has the overall maximum completion time.

2.5.2 Constraints

QoS constrained scheduling attempts to meet user defined requirements of which deadline and budget are the most common. Deadline is the maximum amount of time users need to wait in order to receive the result of the execution of their request. Budget is the amount of money the users wish to spend when using the resources. QoS constrained workflow scheduling is closer to real-world scientific (and other) applications in contrast to optimization scheduling approaches [13].

2.6 Related Work

This section presents related work to exercise the discussed taxonomy. Scheduling workflow tasks to resources while meeting workflow dependencies and constraints is known as the Workflow Scheduling Problem (WSP) – a class of

problem that is known to be NP-complete [12]. When multiple user requirements are used as constraints, the problem of workflow scheduling becomes even more challenging. In this section, I classify recent related research into three categories: Budget Constrained, Deadline Constrained, and Deadline and Budget Constrained.

2.6.1 Deadline Constrained Scheduling

The deadline is the time by which a workflow must complete its execution. In most clouds, heterogeneous resources are available that offer different levels of performance at different price offerings. Generally, faster resources are more expensive compared to slower ones. Therefore, there is often an exploitable trade-off between execution time and the cost of resources.

One of the main strategies in the literature in deadline constrained workflow scheduling is distributing deadline among tasks [43–46]. The distribution phase consists of two steps: workflow partitioning and deadline assignment. In the workflow partitioning, tasks can be considered as independent tasks in levels versus dependent tasks into different paths.

Deadline Bottom Level (DBL) [43] and Deadline Top Level (DTL) [44] are the most popular deadline distribution heuristics. DBL categorizes tasks in bottom-top direction while DTL partition tasks in the opposite direction, top-bottom. In the DBL heuristic, tasks are grouped into different levels where there are no dependencies between tasks in each level. However, tasks in DTL are categorized into paths as synchronization task or a simple task. A synchronization task is defined as a task that has more than one parent or child [44].

In the deadline assignment step, the overall deadline is divided and distributed in proportion to the minimum execution time of each level. However, in DBL, first the primary estimation on fastest instances is calculated. Then, the difference between the user-defined deadline and the primary estimation is distributed uniformly among all levels.

In [45], Yuan et al. presented the Deadline Early Tree (DET) algorithm. In

DET, tasks are partitioned into two types: critical and non-critical activities. All tasks on the critical path are scheduled using dynamic programming under a given deadline. Non-critical tasks are backfilled between critical tasks. However, the communication time between tasks in a workflow is not taken into account by the DET scheduler.

The Hybrid Cloud Optimized Cost (HCOC) scheduling algorithm by Bittencourt and Madeira presented in [47] focuses on optimizing cloud-bursting from private to public clouds. The initial schedule starts to execute tasks on private cloud resources; if the initial schedule cannot meet the deadline, additional resources are leased from a public cloud on demand. The combination of private and public cloud models means that this work cannot be applied to a purely commercial cloud context.

In [46], Abrishami et al. presented the Infrastructure as a service (IaaS) Cloud Partial Critical Paths (IC-PCP). All tasks in a partial critical path (PCP) are scheduled to the same cheapest instance that can complete them by the given deadline. This avoids incurring communication costs for each PCP. However, the IC-PCP algorithm does not consider the boot and deployment time of VMs, even though these are created on demand. One extension of IC-PCP that attempts to further reduce cost is the Enhanced IC-PCP with Replication (EIPR) algorithm [14] in which Calheiros and Buyya use idle instances and budget surpluses to replicate tasks. Their experimental results show that the likelihood of meeting deadlines is increased by using task replication. However, task replication in EIPR comes at an opportunity cost as the resources could be used for new rather than replicated computation. Due to dynamic resource provisioning phase in [14, 46], both algorithms are placed in SPDP category (see section 2.4).

In [48], Byun et al. presented the Partitioned Balanced Time Scheduling (PBTS) algorithm that estimates the minimum number of instances required to meet the deadline in order to minimize execution cost. The PBTS algorithm has three phases, which are task selection, resource capacity estimation and the task scheduling phase. However, only one VM type is considered for provisioning and scheduling in order to simplify the estimation of resource capacity.

The Just in Time (JIT) algorithm proposed by Sahni and Vidyarthi in [49] is a dynamic cost minimization deadline constrained algorithm. The JIT algorithm, which is placed in SPDP category, attempts to combine pipeline tasks into a single task that can abrogate the data transfer time between co-located tasks. The majority of algorithms prioritize tasks to find the best candidate for execution –however, no such policy is used in JIT.

2.6.2 Budget Constrained Methods

Scheduling in cloud environment encounters some challenges not present in traditional heterogeneous environments such as the grid or HPC clusters. The cost model and resource provisioning in cloud are among the main challenging differences. For instance, pricing schemes in cloud systems are based on lease intervals from the time of provisioning, even if the instance is only used for a fraction of that period. A budget is the maximum amount of financial resource that users wish to pay to run their workflows. Algorithms in the budget constrained category attempt to minimize workflow completion time for a given budget. A significant number of cost-aware workflow scheduling algorithms in grid have been proposed, such as [15, 50]. However, cost in grid is calculated based on the accumulated cost of requested services [27]. Moreover, workflow scheduling in grid focuses on minimizing the makespan without considering the cost [51]. Grid provides a static pool of resources whose configuration is known in advance [13], therefore previous work developed for grid are placed in SPSP or DPSP category.

In [15] Sakellariou et al. presented two budget constrained algorithms, LOSS and GAIN. The algorithms start with one of two different initial assignments. The first assignment is the best assignment: a time-optimized assignment in which the execution time is the minimum possible. For example, the HEFT algorithm [51] is used as an initial assignment for the LOSS algorithm. HEFT is one of the most common scheduling algorithms and attempts to reduce the workflow makespan. The second assignment is the cheapest assignment: a cost-

optimized assignment wherein all tasks are assigned to resources having the least execution cost. For example, GAIN uses the cheapest assignment as the initial assignment. Tasks are repeatedly selected for reassignment until the user constrained budget is reached. These algorithms, however, were designed for non-elastic grid environments.

An extension of the HEFT [51] algorithm called the Cost Conscious Scheduling Heuristic (CCSH) was presented by Li et al. in [52]. The CCSH first constructs a priority list of tasks and then assigns the task with the highest priority value to the most cost-efficient virtual machine (VM). However, only one VM type and one pricing model is considered. The authors later introduced the Pareto dominance cost-efficient heuristic to the CCSH to consider different cost models [53].

In [54] and [55], Zheng et al. proposed Budget constrained Heterogeneous Earliest Finish Time (BHEFT), which is an extension of HEFT algorithm [51]. In BHEFT, a current task budget (CTB) factor is introduced to distribute spare budget among unscheduled tasks. In the budget distribution phase, the task budget and spare budget are calculated task by task. Moreover, their work is set within the context of a grid environment, which is not directly applicable to cloud environments due to differences in the cost model.

The authors in [56] and [57] presented an algorithm with budget constraints called minimum end-to-end delay under cost constraint (MED-CC). Firstly, each task in a workflow is assigned to an instance. In the next step, all critical tasks are considered for rescheduling with the proposed Critical Greedy algorithm. In [58], Zeng et al. presented a budget-aware backtracking algorithm for executing large-scale task workflows, referred to as ScaleStar. Their algorithm uses a new metric termed the Comparative Advantage (CA) to select resources in a way to minimize cost. The CA metric attempts to balance cost and execution time. This work, while developed for cloud, used a grid cost model. However, ScaleStar is a backtracking algorithm that needs to recalculate the makespan and cost for every re-scheduling step increasing time complexity. The cost model considered in [56–58] is based on the use of fractional resources. For

instance, in [57] a base processing unit, which is ten instructions per time unit, is set with price at 0.01 per time unit. However, in most of the cloud providers, like Amazon EC2, users are billed based on a longer interval such as one hour.

In [59] two auto scaling techniques to solve the budget constrained scheduling for a workload consisting of multiple workflows were proposed. In this work, budget is distributed to different workflows proportionally based on assigned priorities. Both algorithms in [59] are designed to receive workflows continuously and placing them in DPDP category.

Scheduling bags of tasks under budget constraints in cloud is presented in [60]. One of the assumptions considered by authors is tasks is preemptive, which means they can be interrupted, delayed and then re-triggered sometime later.

2.6.3 Deadline and Budget Constrained Scheduling

Based on the user-defined constraints for the budget and deadline, a combination of different solutions is proposed in [61] to customers. Their main objective is to produce a set of solutions from which a user makes a selection. However, their approach is designed for users in grid environment. Pricing schemes in cloud systems are based on lease intervals from the time of provisioning, even if the instance is only used for a fraction of that period. However, cost in grid is calculated based on the accumulated cost of requested services.

Meta-heuristics such as Genetic Algorithm(GA), Ant Colony Optimization (ACO) and Particle Swarm Optimization (PSO) are used to tackle the workflow scheduling problem with multiple constraints. In [62], the genetic algorithm approach was used to address scheduling optimization problems in workflow applications, based on two QoS constraints: deadline and budget.

In [63], Chen et al. use Ant Colony Optimization (ACO) to schedule the large-scale workflows with three QoS parameters, namely, cost, time and reliability. Users can specify two of the constraints and ACO can find an optimized answer for the third constraint. A Bi-Criteria Priority-based Particle Swarm Op-

timization (BPSO) was proposed in [64] to solve the workflow scheduling problem in cloud under the deadline and budget constraints. BPSO was shown to reduce the execution cost by up to 50% compared to several then state-of-the-art algorithms under the same deadline and budget constraint.

While search-based and meta-heuristic strategies produce acceptable answers, they are usually time-consuming algorithms based on their need for an initialization phase to reach an acceptable answer. Another important point that needs to be considered in these algorithms is that lack of ability to perform well when some changes occur during execution. Moreover, they do not respond well in dynamic scheduling because of the required dynamic phase.

The following three algorithms consider more than one constraint in scheduling workflows. In [55], Zheng and Sakellariou introduce an extended version of HEFT algorithm with both budget and deadline constraints (BHEFT). BHEFT checks if a workflow can be scheduled based on the available budget and deadlines. To select the best possible instance in BHEFT, two variables named Spare Application Budget and Current Task Budget are used. This work, while developed for cloud, used a grid cost model where the number of resources is fixed. Moreover, in their presented algorithms, there is no mechanism to control the deadline if in any steps the algorithm exceeds the user-defined deadline.

In Poola et al. [65], robustness, budget and deadline are considered as user prioritizable constraints and tasks are categorized based on partial critical paths, in a similar way to IC-PCP [46]. Three resource selection policies are used: deadline, cost and robustness, and each user can prioritize the policies independently.

The scheduling algorithms proposed by Malawski et al. in [23] aim to maximize the number of serviced workflows while meeting given budget and deadline constraints. These scheduling algorithms are designed for workflows in an Infrastructure as a Service (IaaS) cloud. However, the authors consider only one instance type rather than the wide variety of types that are currently supported by commercial providers. The algorithms presented in [23] are designed to receive multiple workflows at different times, therefore they belong to DPDP

category.

There are few heuristic-based studies addressing workflow scheduling and resource provisioning while considering both deadline and budget constraints at the same time in cloud. In [66], compromised cost-time algorithm is presented that deadline and budget are considered as QoS parameters. The main step of [66] is rescheduling tasks that causes higher time complexity. Moreover, they did not consider heterogeneous instances in the problem formulation.

Most of the workflow scheduling research efforts focused on the scheduling of the workflows under a single constraint. The review of the above-mentioned papers reveals a need for an algorithm that considers both budget and deadline constraints at the same time in cloud environment.

2.6.4 Multiple Workflows

In this section, I focus on the dynamic scheduling of multiple workflows in a cloud environment. I identify and analyze related solutions for multiple workflows on cloud with the following aspects:

- Workload: individual workflows vs. interrelated workflow instances known as ensembles. Workflows in an ensemble typically have a similar structure, but they differ in their input data, number of tasks, and individual task sizes [23].

- Arrival pattern: it determines whether workflows are submitted at the same time or continuously arrived for execution in different time intervals.

- Virtual machine (VM) types: either a finite or infinite set of heterogeneous resources (e.g. compute power, memory and input/output capacity).

- Optimization: minimizing the makespan or cost of execution are common optimization objectives.

- Constraints: user-defined requirements such as deadline and budget are the most common. Constraints can be be considered for the entire workloads or individual workflows.

In [67], different strategies are discussed for scheduling multiple workflows scheduling in grid – all workflows are submitted at the beginning of the scheduling phase. The algorithms developed in [67] are set within the context of a grid environment, which is not directly applicable to cloud environments due to the fundamental differences around elastic resource provisioning and pay-per-use charging model. This work belongs to SPSP category (see section 2.4), as the number of workflows and instances is known in advance.

Merge-based algorithms are presented in [68, 69] to handle multiple workflows. They assume all workflows are submitted at the beginning, and multiple workflows are merged into a single workflow. These approaches are static and typically achieve higher utilization (compared to dynamic algorithms) of resources because all workflows are known, and a scheduler can leverage such information for improving utilization. However, solutions developed under such assumptions are not directly transferable to dynamic environments where workflows can be submitted dynamically by different users or triggered by devices/machines at any time.

A rescheduling and task rearrangement heuristic for multiple workflows is proposed in [70] whereby it is assumed that all resources are available during the workload lifetime and ignores the provisioning of new instances during execution. Therefore, the algorithm in [70] belongs to DPSP category. Moreover, rescheduling tasks incurs higher time complexity and this additional complexity may not be acceptable for deadline constrained applications.

In [23], the authors introduced algorithms for scheduling ensembles, which are made up of inter-related workflows with the aim of serving as many high priority workflows as possible under predefined budget and deadline constraints. However, the authors consider only one instance type rather than the multiple instance types that are currently supported by commercial providers.

Another work for scheduling concurrent workflows in HPC cloud is pre-

sented in [71], which is named PCH (Path Clustering Heuristic). In this approach, tasks are clustered into different groups, and then priorities are assigned to the task groups. The main purpose of the scheduling algorithms developed in [71] is to improve the VM utilization by using unused gaps to execute tasks from other workflows. In their model, all workload is submitted for execution at the start, rather than in a stream.

A Workflow as a service (WFaaS) architecture presented in [72] focuses on scheduling continuous workflow request. In the proposed WFaaS architecture, reusing cloud resources are only allowed for tasks within the same workflow but not among other workloads' tasks. This limit on resource reuse is not necessary in dynamic scheduling of workloads. Most recently, Rodriguez et al. [73] proposed a resource provisioning and scheduling solution for multiple workflows designed for WaaS environments. Their work aims to minimize the overall cost of leasing the resources while meeting the deadline constraint of each individual workflow. Algorithms in [72, 73] are placed in DPDP category as independent workflows can be submitted at different times.

Chapter 3

Motivation and Problem Statement

3.1 Motivation

This chapter addresses a number of challenges related to scientific workflow scheduling in dynamically provisioned commercial cloud environments. The ever-increasing use of cloud computing by scientists has highlighted opportunities for improving utilization of cloud infrastructure by improving response time while decreasing the net cost of computation. Workflow scheduling in cloud environments differs from grid and cluster computing environments primarily in the elastic resource provisioning and pay-per-use charging model. Therefore, workflow scheduling in clouds requires a different approach in mapping tasks to resources. The main challenges of the cloud paradigm for running e-science applications that need to be considered are:

3.1.1 Dynamic Provisioning

Scheduling models in grids and clusters are subject to best effort scheduling. In this model, the number of required resources and estimation of time required to run a workflow is specified by users. Resource requests are located into a queue to turn for available services that can be executed. Accordingly, some tasks should wait a long time in the queue, particularly behind those that take

long time to run or need plenty of numbers of computing resources. Therefore, resource allocation and linking the tasks to resources in grids and clusters are connected together and beyond customer's control. In cloud platforms the method is reversed. Resources that are provisioned directly by users to run their computations need using a scheduler. This provisioning model enables the allocation of resources first and running several tasks, which decreases the overhead of scheduling and can significantly improve the performance [74].

3.1.2 On-demand Resources

All resources in cloud are allocated on demand. Cloud customers usually request resources based on their needs at all times. In users' view, cloud consists of unlimited resources where they can obtain as many as they want. Therefore, if not enough resource are available right away, then the demand fails. However, on-demand resources are the perfect choice for loosely-coupled applications such as workflows. These kinds of applications can start with part of the entire required resources. With this model, as many resources as possible can be allocated to a workflow and it can be started immediately.

3.1.3 Elasticity

As well as resource provisioning on demand in clouds, users can release resources on demand. This two-sided ability, called elasticity, enables workflow applications to increase or decrease the existing resource pool as workflow needs may change in time [75].

3.1.4 User Requirements

Workflows are scheduled on the basis of execution dependencies, expected execution times, available resources, and typically under either a budget or deadline constraint. The significance of these constraints is discussed with examples:

- **Budget**:

Cost plays an important role in the cloud environment due to its impact on the budget of users and the profitability of providers. Unlike grid or in-house HPC systems, most cloud providers, such as Amazon, charge users for a fixed interval from the time of provisioning, that is, users pay for the whole interval even if the instance is only used for a fraction of that period. From a user's perspective, finding a schedule with a lower makespan for a given budget is the main concern, while from the provider's perspective, maximizing utilization is the main concern. Launching too many instances does not lower makespan. Instead, it causes high scheduling overhead and low instance utilization. As one would expect, poor resource provisioning wastes available user budget.

- **Deadline**:

  There are many complex and large-scale scientific processes in different domains such as weather forecasting, climate modeling, medical modeling and disaster recovery simulation [28], which are examples of deadline-sensitive applications. Some need to be executed quickly or in a specified time. Deadline constrained applications are divided into two types:

  - Hard deadline: If deadline is missed, it could have disastrous results on life or environment.
  - Soft deadline: The application can tolerate small margins in deadline violation.

  The landfall of a storm can be predicted by coastal hazard applications and used to determine whether to send out evacuation orders. In such workflows, missing the execution deadline is considered a failure because safety and lives are at risk [76]. Weather forecast workflows [77] that forecast the weather for the next day (or week) and medical applications that analyze data related to patients activity are examples of applications with soft deadline.

The algorithms designed and developed in this thesis consider discussed

key challenges and features, to solve the scientific workflow scheduling problem in cloud.

3.2 Scientific Workflows Overview

In this part, the overview of real world applications [21] is presented . The workflows are taken from various application scopes such as astronomy [22], biology [78], gravitational physics [79] and earthquake science [24].

- Montage: An astronomical Mosaic Engine (Montage) was generated by NASA/IPAC Infrared Science Archive [22]. It was designed to create custom mosaics images of the sky by astronomers on their own laptop. Montage can be used to merge these sky images.

- Cybershake: This project is used by the Southern California Earthquake Center (SCEC) to build a model of earthquake hazards [24].

- Epigenomic: The USC epigenome center manages research [80] in mapping the epigenetic state of human cells on a genome wide scale. Epigenomic uses DNA sequence, which is split into several chunks that could be run in parallel.

- LIGO: The Laser Interferometer Gravitational Wave Observatory (LIGO) detects gravitational waves created by several events in the universe as per Einstein's theory of general relativity. The LIGO [79] is applied to study about the data received merging compact binary systems such as binary neutron stars and black holes.

- SIPHT: The bioinformatics project at Harvard University is conducting a wide search for small, untranslated RNAs (sRNAs) that regulate processes such as secretion and virulence in bacteria [78].

3.3 Problem Definition

3.3.1 Application Model

Workflows are the most widely used models for representing and managing complex distributed scientific computations [28]. A Directed Acyclic Graph (DAG) is the most common abstraction of a workflow. Using a DAG abstraction, a workflow is defined as a graph $G = (T, E)$ where $T = \{t_0, t_1, ..., t_n\}$ is a set of tasks represented by vertices and $E = \{e_{i,j} \mid t_i, t_j \in T\}$ is a set of directed edges denoting data or control dependencies between tasks. An edge $e_{i,j} \in E$ represents the precedence constraint as a directed arc between two tasks t_i and t_j where $t_i, t_j \in T$. The edge indicates that task t_j can start only after completing the execution of task t_i with all data received from t_i, and this implies that task t_i is the parent of task t_j and task t_j is the successor or child of task t_i. Each task can have one or more parents or children. Task t_i cannot start until all parents have been completed. In order to ensure the DAG has only one input and one output, two dummy tasks that have zero execution cost are added to the DAG.

3.3.2 System Model

The IaaS paradigm provides a service by offering instance types containing various amounts of CPU, memory, storage and network bandwidth at different prices. Workflows are executed on different instance types, and each instance type is associated with a set of resources.

A resource model is used based on the Amazon Elastic Compute cloud, where instances are provisioned on demand. The pricing model is a pay as you go with minimum hourly billing. Under this pricing model, if an instance is used for one minute, a user has to pay for the whole hour. A common approach is to assume cloud vendors provide access to unlimited number of instances and the instances are heterogeneous (denoted by $P = \{p_0, p_1 \dots p_h\}$, where h is the index of the instance type). It is assumed that all instances and storage services are located in the same region and also the average bandwidth between

the instances is essentially identical.

The resource provisioning problem is somehow simplified and is similar to scheduling in Grid with a limited number of processors. However, provisioning decisions are still important due to the overhead and cost associated with leasing VMs. By accessing a virtually unlimited number of VMs, algorithms need to find effective policies to manage this abundance of resources efficiently. Therfore, I considered a general scenario and assumed that we have an unlimited number of instances.

3.3.3 Definitions

Most studies on workflow scheduling assume that estimated execution time for workflow tasks is known. Task runtimes can be estimated using analytical modeling, empirical modeling and historical data. In this thesis, scientific workflows using trace data from real applications are considered [75]. Execution time (computation cost) for task t_i on instance p_j is denoted by $w_{t_i}^{p_j}$.

All immediate predecessors of task t_i are defined as:

$$pred(t_i) = \{t_j \mid (t_j, t_i) \in T\}. \tag{3.1}$$

Also, all immediate successors of task t_i are defined as:

$$succ(t_i) = \{t_j \mid (t_i, t_j) \in T\}. \tag{3.2}$$

For example in Figure 3.1, predecessors of task 11 are 7, 9 and 10, and successors of 5 are 7 and 8.

A task without any parent is an entry task and a task without any children is called an exit task. In Figure 3.1, task 0 is an entry task and task 11 is an exit task. Thus, by definition I have:

$$pred(t_{entry}) = \{\emptyset\}, \tag{3.3}$$

$$succ(t_{exit}) = \{\emptyset\}. \tag{3.4}$$

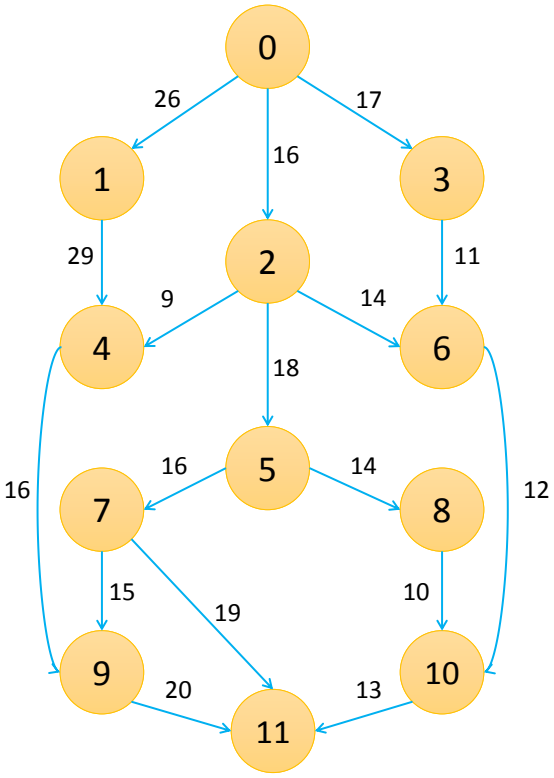


Figure 3.1: A sample DAG with 12 tasks

The completion time of a workflow is called the schedule length or makespan (denoted by $\mathcal{L}_{ms}$). Because t_{exit} is the last task that can be executed, the time until completing the exit task is defined as the makespan of a workflow.

$$\mathcal{L}_{ms} = FT\left(t_{exit}\right), \tag{3.5}$$

where $FT(t_{exit})$ is the finish time of the last task in a workflow.

The amount of data transferred from task t_i to task t_j is called communication time (denoted by $C_{i,j}$), and this time is calculated as:

$$C_{i,j} = \begin{cases} \frac{data}{\beta} & , p_i \neq p_j \\ 0 & , p_i = p_j. \end{cases} \tag{3.6}$$

If task t_i and task t_j are executed on the same instance (denoted by $p_i = p_j$), data transfer between them is local and the communication cost is defined as

zero. Otherwise, the communication cost is the ratio between the size of data ($data$) to be transferred from task t_i to t_j to the average bandwidth (β) in the data-center.

The Earliest Start Time (EST) of a task t_i is calculated on the instance with the shortest execution time and defined as:

$$EST(t_i) = \begin{cases} 0 & , t_i = t_{entry} \\ \max\limits_{t_j \in pred(t_i)} \left\{ EST(t_j) + w_{t_j} + C_{i,j} \right\} & , \text{Otherwise}, \end{cases} \tag{3.7}$$

where w_{t_j} is the execution time of task t_j on the fastest instance type.

The cost of executing task t_i on instance p_j is calculated as:

$$TaskCost_{t_i}^{p_j} = \left\lceil \frac{w_{t_i}^{p_j}}{N_t} \right\rceil * c_j, \tag{3.8}$$

where c_j is the cost of instance p_j for one time interval and N_t is the time of an interval. Finally, the overall cost of executing all tasks in a workflow (G) is defined as:

$$Cost_o = \sum_{t_i \in G} TaskCost_{t_i}^{p_j}. \tag{3.9}$$

Scientific workflows are executed on resources with the initial objective of optimizing the total execution time (makespan) of the workflow. Besides the makespan, execution cost of scientific applications also forms one of the objective functions of scheduling. Therefore, the objectives of scheduling a workflow can vary from application to application. It depends on what objective function the user wants to minimize or maximize - examples being minimizing overall workflow completion time, minimizing cost, maximizing resource utilization or throughput, and executing within the defined deadline and allocated budget. In this thesis, the objectives are to minimize workflow completion time within a given budget constraint (time optimization), to minimize cost within a given deadline (cost optimization), and meet multiple contraints without any failure.

A general form to minimize cost under deadline constraint can be formulated as:

$$minimize\{Cost_o\} \tag{3.10}$$

subject to,

$$\{\mathcal{L}_{ms}\} < Deadline \tag{3.11}$$

Proceeding the same way, a general form to minimize makespan under budget constraint can be formulated as:

$$minimize\{\mathcal{L}_{ms}\} \tag{3.12}$$

subject to,

$$\{Cost_o\} < Budget \tag{3.13}$$

Chapter 4

Deadline Constrained Workflow Scheduling

4.1 Introduction

This chapter presents two new algorithms, Proportional Deadline Constrained (PDC) and Deadline Constrained Critical Path (DCCP), for scheduling eScience workflows on commercial clouds, which focus on deadline constraints while minimizing costs. Both algorithms belong to the class of list-based scheduling algorithms [36] consisting of a task prioritization phase and a task assignment phase. The PDC algorithm maximizes parallelism in a workflow by separating it into logical levels and then proportionally subdividing the overall workflow deadline over different levels. Moreover, the performance of the PDC algorithm is improved by refining the task ranking carried out during the task prioritization step. The DCCP algorithm uses the concept of Constrained Critical Paths (CCP) to execute a set of tasks on the same instance with the goal of reducing communication cost between instances. In the DCCP algorithm, backfilling leftover capacity (residuals) is considered in provisioned instances and several different backfilling strategies are applied. Cloud providers charge resources by fixed time intervals no matter if an instance is only used for a fraction of that period. Residuals are defined as leftover computation time within allocated in-

tervals. This approach is most effective in data-intensive workflows due to the reduction in data movement.

4.2 PDC and DCCP Algorithms

In this section, I present two deadline constrained algorithms, Proportional Deadline Constrained (PDC) and Deadline Constrained Critical Path (DCCP). As a guide to this section, Figure 4.1 shows the sequence of steps for scheduling a workflow in both PDC and DCCP and indicates which sub-section details those parts of the algorithms.

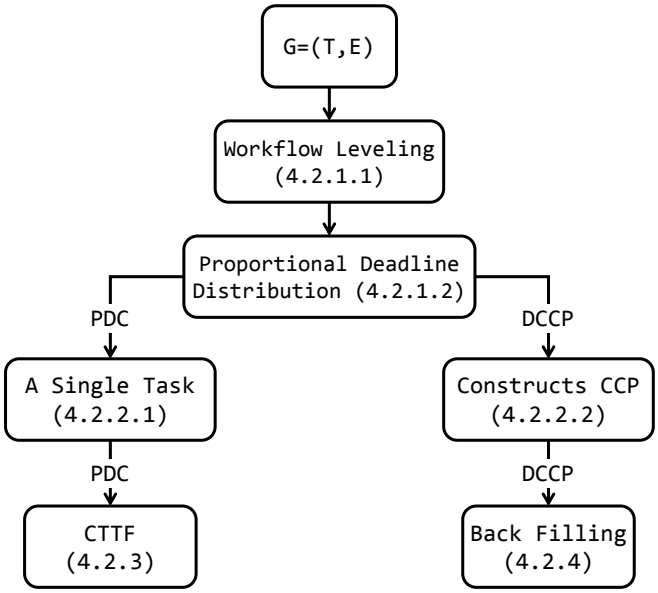


Figure 4.1: Scheduling workflow with PDC and DCCP

4.2.1 Preprocessing Step

Both DCCP and PDC use a preprocessing step for partitioning tasks. In the preprocessing step, the tasks are partitioned into different levels based on their

respective dependencies. Subsequently, the user-defined deadline T_D is distributed over the levels established in the preprocessing step. Each level gets its own level deadline and all tasks in the same level have the same level-deadline.

4.2.1.1 Workflow Leveling

I aim to maximize task parallelism by partitioning tasks so there are no dependencies between tasks in each level. Each level can therefore be thought of as a bag of tasks (BoT) containing a set of independent tasks.

There are two main algorithms for allocating tasks into different levels: Deadline Bottom Level (DBL) [43] and Deadline Top Level (DTL) [44]. The DBL and DBT algorithms categorize tasks in bottom-top direction and top-bottom direction, respectively. In this thesis, I use the DBL algorithm to partition tasks over the different levels.

The level of task t_i is described as an integer representing the maximum number of edges in the paths from task t_i to the exit task (see Fig. 3.1). The level number (denoted by N_L) associates a task to a BoT. For the exit task, the level number is always 1, and for the other tasks it is determined by:

$$N_L\left(t_i\right) = \max_{t_j \in succ(t_i)} \left\{N_L\left(t_j\right) + 1\right\}, \tag{4.1}$$

where $succ(t_i)$ denotes the set of immediate successors of task t_i. All tasks are then grouped into Task Level Sets (TLS) based on their levels:

$$\mathrm{TLS}(\ell) = \{t_i | N_L\left(t_i\right) = \ell\}, \tag{4.2}$$

where ℓ is an integer denoting the level in $[1 \ldots N_L\left(t_{entry}\right)]$.

4.2.1.2 Proportional Deadline Distribution

Once all tasks are assigned to their respective levels, the tasks are proportionally distributed across each level based on the user deadline (T_D). Each sub-deadline assigned to a level is termed the level deadline $(T_{sd}(\ell))$. In order to

meet the overall deadline, I attempt to ensure that every task in a level can complete its execution before the assigned sub-deadline. Firstly, the initial estimated deadline for each level (ℓ) is calculated as:

$$InitialT_{sd}(\ell) = \max_{t_i \in \mathrm{TLS}(\ell)} \{\mathrm{ECT}(t_i)\}, \tag{4.3}$$

where $\mathrm{ECT}(t_i)$ denotes the Earliest Completion Time (ECT) of task t_i over all instances and the ECT is defined as

$$\mathrm{ECT}(t_i) = \max_{\ell \in pred(N_L(t_i))} \{InitialT_{sd}(\ell), EST\,(t_i)\} + w_{t_i}, \tag{4.4}$$

where $EST\,(t_i)$ is defined in (3.7), $pred\,(t_i)$ denotes the set of predecessors of task t_i; w_{t_i} denotes the minimum execution duration for task t_i and ℓ indicates the parent level for all parents of t_i. The task, t_{entry} has no predecessors, its ECT is equal to zero. In equation (4.3), the maximum ECT of all tasks in a level is used as the overall estimate for that level. This duration is effectively the absolute minimum time that is required for all tasks in a level to complete execution in parallel.

After calculating the estimated deadline value for all levels, I distribute the user deadline among all tasks non-uniformly based on a deadline proportion denoted by $\propto_{deadline}$ in equation (4.5):

$$\propto_{deadline} = \frac{T_D - InitialT_{sd}(1)}{InitialT_{sd}(1)}, \tag{4.5}$$

where $InitialT_{sd}(1)$ is the level that contains the exit task.

Then the length of each level deadline is computed as a function of this deadline proportion to each level as follows:

$$T_{sd}(\ell) = InitialT_{sd}(\ell) + (\propto_{deadline} \times |InitialT_{sd}(\ell)\,|)\,. \tag{4.6}$$

Intuitively, the levels with longer executing tasks gain a larger share of the user deadline.

Policy	Description	Formula	Policy Type
Upward Rank ($rank_u$)	The length of critical path from task t_i to task t_{exit}	$rank_u(t_i) = \overline{w_i} + \max_{t_j \in succ(t_i)} (\overline{c_{i,j}} + rank_u(t_j))$ (4.7)	static
Downward Rank ($rank_d$)	Starts from the t_{entry} and is computed recursively by traversing the DAG to t_{exit}	$rank_d(t_i) = \max_{t_j \in pred(t_i)} (\overline{w_j} + \overline{c_{j,i}} + rank_d(t_j))$ (4.8)	static
Sum Rank ($rank_s$)	Sum of the upward and downward rank	$rank_s(t_i) = rank_u(t_i) + rank_d(t_i)$	static
Minimum Execution Time (MinExe)	Lowest execution time is given first priority	$\min(w_i)$	static
Maximum Execution Time (MaxExe)	Tasks with a longer execution time has higher priority	$\max(w_i)$	static
Random	Tasks are picked from the ready list at random	—	static
Earliest Completion Time (ECT)	The task that finishes first will be the best candidate for execution	$\min(FTw_i)$	dynamic
Earliest Deadline First (EDF)	Tasks with minimum EDF have highest priority	$EDF(t_i) = T_{sd}\left(N_L\left(t_i\right)\right) - EST(t_i)$	dynamic

Table 4.1: Ranks values

4.2.2 Task Prioritization

4.2.2.1 PDC Algorithm (A Single task)

In each step of the PDC algorithm, tasks that are ready to execute are placed into the task ready list. A task is ready when all of its parents have been executed and all its required data are readily accessible. In order to select a task, at first all tasks in the ready list should first be prioritized. I used eight different policies in order to show how the order of execution can influence the scheduling results, particularly the cost. These policies summarized in Table 4.1. I will discuss these results later (see section 4.3.2).

1. Upward Rank ($rank_u$): This ranking is presented in [51]. The upward rank is the length of critical path from task t_i to task t_{exit} and is calculated by equation (4.7), where $\overline{w_i}$ and $\overline{c_{i,j}}$ are the average execution time and average communication time of task t_i, respectively. The rank is called an upward rank because the ranking process starts from the exit node and ranks are calculated recursively by traversing the DAG to the entry node.

2. Downward Rank ($rank_d$): The downward rank [51] starts from the entry node and is computed recursively by traversing the DAG to the exit node. $rank_d(t_i)$ is the longest distance from t_{entry} to task t_i, excluding the computation cost of the task itself, where $rank_u(t_i)$ is the length of the critical path from task t_i to t_{exit}, including the computation cost of the task itself [51].

3. Sum Rank($rank_s$): This rank gives equal importance to both the uprank and downrank and is calculated as the arithmetic sum of $rank_u$ and $rank_d$.

4. Minimum Execution Time (MinExe): For each task in the ready list, the minimum execution time on all VMs types is calculated and the task with the lowest execution time is given first priority.

5. Maximum Execution Time (MaxExe): Similar in principle to the minimum execution time, with the only difference being that the task with a longer execution time has higher priority.

6. Random: In this policy, tasks are picked from the ready list at random.

7. Earliest Completion Time (ECT): For each task the earliest completion time on all VMs launched is calculated. The task that finishes first will be the best candidate for execution.

8. Earliest Deadline First (EDF): Tasks with a minimum EDF have highest priority among all ready tasks.

4.2.2.2 DCCP Algorithm (Multiple tasks: CCP definition)

A Critical Path (CP) is the longest path from the entry to exit node of a task graph [81]. The length of critical path ($|CP|$) is calculated as the sum of computation costs and communication costs, and can be considered as the lower bound for scheduling a workflow.

Several heuristics that utilize critical paths have been proposed for addressing the workflow scheduling problem [51, 81, 82]. The set of tasks containing only the tasks ready for scheduling constitutes a constrained critical path (CCP) [83]. In the DCCP algorithm, the CCP in a workflow is determined based on HEFT *upward rank* and *downward rank* [51], then I apply a set of new ranking methods defined as follows:
modified upward rank :

$$Mrank_u(t_i) = \overline{w_i} + \sum_{t_j \in succ(t_i)} (\overline{c_{i,j}}) + \max_{t_j \in succ(t_i)} (rank_u(t_j)) \tag{4.9}$$

modified downward rank :

$$Mrank_d(t_i) = \sum_{t_k \in pred(t_i)} (\overline{c_{k,i}}) + \max_{t_k \in pred(t_i)} (\overline{w_k} + rank_d(t_k)) \tag{4.10}$$

The difference between my modified rank and standard rank is that the modified rank aggregates a task's predecessors' or successors' communication time instead of selecting the maximum. With the modified rank, tasks with higher out-degree or in-degree have higher priorities. As a result, these tasks

are executed first with higher probability and more tasks on the next CCP can be considered as ready tasks.

In this thesis, I use the sum rank to find all CPs [51]:

$$rank_s = rank_u + rank_d \tag{4.11}$$

In DCCP, all tasks are first sorted based on their $rank_{sum}$ values and those tasks with the highest values are selected as the first CP. All tasks in the first CP are labeled as visited tasks. Proceeding in the same way, all CPs in a workflow can be found.

4.2.2.3 An illustrative example

A sample DAG is considered that contains 12 tasks as shown in Figure 4.2. The numbers associated with each edge shows the data transfer time between tasks. The data could either be direct or indirect via shared storage. For this thesis the only difference is in the absolute time required, and for simulation I only consider direct transfers. The average execution time ($\overline{w_i}$) of each task is displayed in Table 4.3.

1. A single task: Upon executing task 0, all its children are ready for execution. The different start time, end time and data transfer time (blue intervals) are shown in Figure 4.3. Different policies select different tasks (Table 4.2).

	$rank_u$	$rank_d$	$rank_s$	MinExe	MaxExe	ECT
Selected Task	t_2	t_1	t_1	t_3	t_1	t_2

Table 4.2: Selected task by different policies

2. Multiple tasks: As we mentioned before, A Critical Path (CP) is the longest path from the entry to exit node of a task graph. The first CP is obtained based on the highest sum rank, which is the aggregation of $rank_u$ and $rank_d$ and this yields the path $(0 \rightarrow 1 \rightarrow 4 \rightarrow 9 \rightarrow 11)$. Regardless of

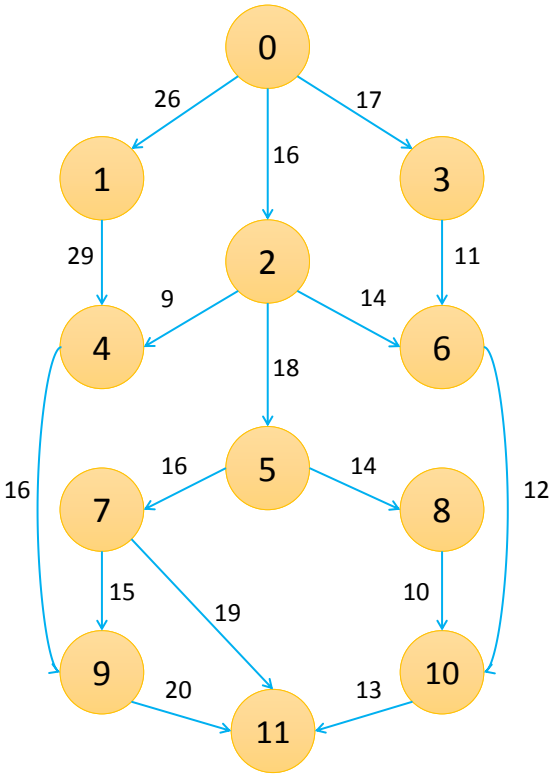


Figure 4.2: A sample DAG with 12 tasks

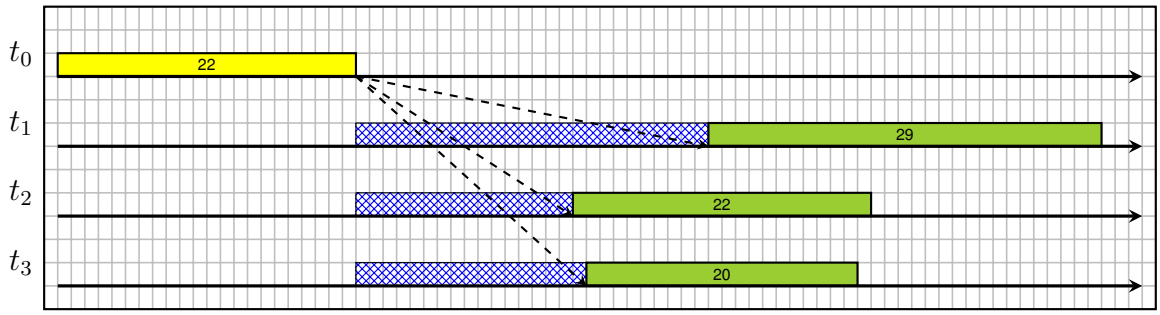


Figure 4.3: Ready tasks and rank values (shown within each bar) after execution of task 0

any previously selected tasks, proceeding in the same way, other CPs are found as displayed in Table 4.4. The next step is traversal of CPs to find CCPs in a round-robin order. The first CCP consists of $(0 \rightarrow 1)$ as other tasks in the first CP are not yet ready. For example, consider t_4, which is in the first CP. This task cannot be added to the CCP as one of its parents, t_2, has not yet been added to any CCPs. As no ready tasks can be found in the first CP, a second (new) CP is constructed. In the new CP I have t_2, which is a ready task, as its only parent has already been included in a previous CCP. Thus, the second CCP consists of three tasks $(2 \rightarrow 5 \rightarrow 8)$, having excluded t_{10} from the second CP. Similarly, other CCPs are generated by using the remaining CPs. The different CCPs calculated by my modified rank approach are presented in Table 4.5.

		Standard Rank			Modified Rank		
Task	$\overline{w_i}$	$rank_u$	$rank_d$	$rank_s$	$rank_u$	$rank_d$	$rank_s$
0	22	190	0	190	284	0	284
1	29	142	48	190	142	48	190
2	22	150	38	188	203	38	241
3	20	96	39	135	96	39	135
4	27	84	106	190	84	115	199
5	21	110	78	188	140	78	218
6	9	65	74	139	65	85	150
7	14	70	115	185	89	115	204
8	12	75	113	188	75	113	188
9	11	41	149	190	41	173	214
10	21	44	144	188	44	156	200
11	10	10	180	190	10	236	246

Table 4.3: Ranks values of tasks in Figure 3.1

Critical Path	Constrained Critical Path
0→1→4→9→11	0→1
2→5→8→10	2→5→8
3→6	3→6
7	7
	4→9
	10
	11

Table 4.4: CPs and CCPs based on standard ranks

Critical Path	Constrained Critical Path
0→2→5→7→11	0→2→5→7
1→4→9	1→4→9
3→6→10	3→6
8	8
	10
	11

Table 4.5: CPs and CCPs based on modified ranks

4.2.3 Instance Selection in PDC

At the point the algorithms perform instance selection: (i) each task is already assigned a level, (ii) the deadline for each level is already determined, and (iii) the priority of each ready task is already assigned. During instance selection, a trade-off must be made between execution time and cost. To demonstrate this trade-off, I show the expressions for both the time and the cost of executing each task on each instance type in equations (4.12) and (4.13), forming two sets of expressions for Time and Cost.

Firstly, the time needed for the current task, t_i, on the instance p_j is calculated by $\mathrm{ECT}\left(t_i, p_j\right)$. The ECT is the earliest time that a task can finish on an instance that is defined earlier in equation (4.4). Using this observation, I can then compute how much the estimated level deadline of the current task differs from the earliest completion time of task on the instance p_j:

$$\mathrm{Time}\ {}^{p_j}_{t_i} = \frac{T_{sd}\left(N_L\left(t_i\right)\right) - \mathrm{ECT}\left(t_i, p_j\right)}{T_{sd}\left(N_L\left(t_i\right)\right) - \mathrm{ECT}\left(t_i\right)}. \tag{4.12}$$

In equation (4.12), $T_{sd}(\cdot)$ is the deadline that is assigned to the level that contains the current task. Also, $\mathrm{ECT}\left(t_i\right)$ is the minimum execution time among all instances that keeps my current task on schedule.

The values of Time for task t_i are related to instance types, wherein the lower value of Time means running on a cheaper instance. The reason is that the values of $\mathrm{ECT}\left(t_i, p_j\right)$ is bigger on an instance with a lower processing capacity. Also, if the value of Time is negative, it means that the current task on the selected instance will exceed the level deadline i.e. $\mathrm{ECT}\left(t_i, p_j\right) > T_{sd}\left(N_L\left(t_i\right)\right)$.

In the expression for Cost, given earlier in equation (3.8), $TaskCost_i$ refers to the cost of scheduling the current task t_i on instance p_j. In equation (4.13), the worst cost (maximum cost) and best cost (minimum cost) of executing the task t_i among all instances are $TaskCost_{worst}$ and $TaskCost_{best}$, respectively.

$$\mathrm{Cost}\ {}^{p_j}_{t_i} = \frac{TaskCost_{worst} - TaskCost_i}{TaskCost_{worst} - TaskCost_{best}}. \tag{4.13}$$

To find the best instance, a Cost Time Trade-off Factor (CTTF) in equation (4.14) is used that considers a trade-off between cost and time.

$$\mathrm{CTTF}\ {}^{p_j}_{t_i} = \frac{\mathrm{Cost}\ {}^{p_j}_{t_i}}{\mathrm{Time}\ {}^{p_j}_{t_i}}. \tag{4.14}$$

When an instance is first provisioned, the instance is billed on an hourly interval until it is terminated. Therefore the first task assigned to an instance in a particular billing interval incurs the entire cost of that interval. As a consequence, if other tasks can be executed during that paid interval, then there is no additional execution cost for executing them. Therefore, during instance selection,

I first prioritize the reuse of such instances (i.e. when Cost in equation (4.13) is 1), provided that the level deadline is not exceeded (i.e. when Time in equation (4.12) is positive).

If there are more than one paid instances, the PDC selects the instance with the minimum execution time (faster instances). If no such instances are available, it will attempt to use a provisioned but as yet unused instance, or as a last resort it creates a new instance.

4.2.4 Instance Selection in DCCP

In the Instance Selection phase, the DCCP algorithm identifies the most appropriate instance to execute CCPs. All tasks in a CCP are executed on the same instance to minimize communication cost between them. The time needed for the current CCP (denoted by (CCP_i)) to execute on the instance p_j is calculated by $\mathrm{ECT}\,(CCP_i, p_j)$. Work in scheduling generally assumes such an estimate can be calculated. In practice this is difficult; however work is underway to profile workflow tools and underlying cloud systems to provide usable estimates for use in production systems [84].

The ECT is the earliest time that a CCP can complete execution on an instance (as defined in equation (4.4) for a single task). The differences between the estimated level deadline and earliest completion time of the current CCP on the instance p_j is determined by:

$$\mathrm{Time}\ {}^{p_j}_{CCP_i} = T_{sd}\left(N_L\left(t_i\right)\right) - \mathrm{ECT}\left(CCP_i, p_j\right), \tag{4.15}$$

where T_{sd} is the deadline that is assigned to the level (given by $N_L(\cdot)$), which contains the last task t_i on the current CCP. There is a possibility that this value may be negative, which means the current CCP exceeds the level deadline ($\mathrm{ECT}\left(CCP_i, p_j\right) > T_{sd}\left(N_L\left(t_i\right)\right)$). The cost of executing all tasks on current CCP on instance p_j is denoted by $Cost_{CCP_i,p_j}$.

$$Cost_{CCP_i,p_j} = \sum_{t_i \in CCP_i} TaskCost^{p_j}_{t_i}. \tag{4.16}$$

Three different scenarios to find the most appropriate instance can be considered:

1. Most cloud providers, such as Amazon, charge based on 60 minute interval. When a task is scheduled on an instance, the whole billing interval is charged no matter how much of the instance is used. Therefore, if other tasks can be executed on the same VM during that paid interval, their execution cost is zero. To find the best instance in DCCP, the priority is to select an instance with residuals to execute a CCP. This is subject to its earliest completion time does not exceed the level deadline. The instance with minimum ECT is selected (the fastest one).

2. A new instance is provisioned if no instances could be found in the previous step. For example, at the beginning of the scheduling to assign the first CCP, an instance should be provisioned as there are no paid instances. For this purpose, DCCP searches among instances that can meet the level deadline and select the cheapest one.

3. In tight deadlines, there is a possibility that none of the instances can meet the task level's sub-deadline (i.e. when $\mathrm{Time}\ ^{p_j}_{CCP_i}$ is negative). If this condition for a CCP is met, it does not mean that it is impossible to meet the overall user-defined deadline. Rather, it means that the sub-deadline will be violated. In this case I select the best available instance – as overall the schedule may still be met.

4.2.4.1 Backfilling in DCCP

Scheduling of a workflow that consists of dependent tasks creates resource utilization gaps between the execution of tasks. The principal reason is that tasks must wait for their data from its parents. Therefore, there are idle time slots formed between scheduled tasks on each resource. Moreover, the utilization of cloud resources depends on how tasks are placed together. Instance fragmentation and resource wasting occurs if tasks are not packed firmly. Scheduling

algorithms can consider these time slots for executing ready tasks on different resources. Consequently, filling up the idle slots decreases the makespan and maximizes the overall instance utilization.

While backfilling policies are widely used to reduce fragmentation, this has not been done previously in workflow scheduling. To my knowledge the use of backfilling strategies in DCCP is unique. In this section, I show that backfilling increases instance utilization and this improved utilization leads to cost saving. In PDC, a cost-time trade-off for instance selection is used, this approach narrows my choices of instances for backfilling. Therefore, the backfill algorithm is only used in conjunction with DCCP. Three different policies are considered that exploit such idle slots to efficiently schedule tasks, which are First Fit (FF), Best Fit (BF) and Worst Fit (WF). Each CCP can be placed in a residual according to one of the following policies:

1. First Fit (FF): a CCP can be inserted into the first gap where it fits.
2. Best Fit (BF): a CCP is placed into the schedule gap where it leaves the minimum sized residuals.
3. Worst Fit (WF): a CCP is inserted into the schedule gap where it leaves the maximum sized residuals.

In Section 4.3.3, I discuss how using of backfill policies significantly improves overall utilization.

4.2.5 Time Complexity

The time complexity of the two proposed algorithms is an important metric for benchmarking different scheduling algorithms. Consider a workflow represented by a DAG $G = (T, E)$ with n tasks. Assume that a DAG is fully connected and the maximum number of dependencies between tasks is $(n)(n-1)/2$. Processing all tasks and its dependencies requires a time complexity of $\mathcal{O}(n^2)$. Besides, processing task dependencies, other computations in PDC that must be

taken into account are the task selection phase and the instance selection phase, which are distinct from each other.

To compute the time complexity of task selection phase, all ready tasks (n) should be examined on all available processors (p) that need computation of $\mathcal{O}(np)$. Similarly, to select all workflow tasks, the time complexity of task selection is $\mathcal{O}(n^2p)$. In the resource selection phase, selected tasks are evaluated on all available instances with complexity of $\mathcal{O}(p)$. Thus, the total time complexity of resource selection is $\mathcal{O}(np)$. The total time for PDC is $\mathcal{O}(n^2 + n^2p + np)$, where the algorithm complexity is of the order $\mathcal{O}(n^2p)$. The only difference between DCCP with PDC is in calculating constrained critical paths. For this purpose, the calculation of upward and downward rank occurs with time complexity of $\mathcal{O}(n^2p)$. Therefore, the DCCP algorithm is also of the order of $\mathcal{O}(n^2p)$.

4.3 Evaluation

In this section, the performance comparison of the PDC and DCCP algorithms is presented, with the well-known IC-PCP [46] algorithm and JIT [49] algorithm. A simulator was used to compare the performance of all four algorithms. Simulations are well accepted as the first approach for evaluating new techniques for the workflow scheduling problem. It allows researchers to test the performance of newly developed algorithms under a controller setting. For this purpose, all four algorithms were implemented and evaluated in CloudSim [85].

The simulation scenario was configured as a single data-center and six different instance types. The characteristics of the instances are based on the US-east Amazon region[1] presented in Table 4.6, and were accurate in March 2016.

The average bandwidth between instances was fixed to 20 MBps [86]. The processing capacity of an EC2 unit is estimated at one Million Floating Point Operations Per Second (MFLOPS) [87]. The estimated execution times are scaled by instance type CPU performance, which means 1 second of each task in a workflow runs for 1 second on an instance with one ECU. One EC2 Compute

[1]https://aws.amazon.com/ec2/pricing/

Type	ECU	Memory(GB)	Cost($)
m3.medium	3	3.75	0.067
m4.large	6.5	8	0.126
m3.xlarge	13	15	0.266
m4.2xlarge	26	32	0.504
m4.4xlarge	53.5	64	1.008
m4.10xlarge	124.5	160	2.520

Table 4.6: Instance Types based on Amazon EC2

Unit (ECU) provides the equivalent CPU capacity of a 1.0-1.2 GHz 2007 Opteron or 2007 Xeon processor. In an ideal cloud environment, there is no provisioning delay in resource allocation. However, some factors such as the time of day, operating system, instance type, location of the data center, and number of requested resources at the same time, can cause delays in startup time [88]. Therefore, in the simulation, a 97-second boot time was used based on measurements of EC2 [88].

In order to evaluate the performance of my algorithms with a realistic load, five common scientific workflows were considered: Cybershake, Epigenomics, Montage, LIGO and SIPHT. The characteristics and task composition of these workflows have been analyzed in published works cited in the related work section [21,75]. To evaluate the performance of these algorithms, different deadlines were chosen from tight to relaxed. Additionally, the fastest schedule was calculated (denoted by FS) as a baseline schedule with the following expression:

$$FS = \sum_{t_i \in CP} (w_i^j) \tag{4.17}$$

where w_i^j is the computation cost of task t_i on the fastest instance p_j. Effectively, this baseline is the fastest possible execution – ignoring costs.

The deadline was defined as a function of the fastest schedule, and this deadline is expressed in equation (4.18) in which the deadline varies from tight to moderate to relaxed:

$$\text{deadline} = \alpha * FS, \quad 0 < \alpha < 20. \tag{4.18}$$

The deadline factor α starts from 1 in order to consider very tight deadlines (typically approaches the fastest schedule) and is increased by one up to a value of 20, which results in a very relaxed deadline.

The Amazon EC2 instances charge on an hourly interval from the time of provisioning. The simulator was configured to reflect this charging model and a time interval of 60 minutes was used in the simulations. To compare performance with respect to different workflow sizes, workflows with 50, 100, 200, 500 and 1000 tasks were evaluated. However, as these results did not vary significantly, here only workflows with 1000 tasks are presented.

The Pegasus workflow generator [75] was used to create representative workflows with the same structure as five real world scientific workflows (Cybershake, Epigenomics, Montage, LIGO and SIPHT). For each workflow structure, and each deadline factor, 100 distinct Pegasus generated workflows are scheduled in CloudSim and the performance of the scheduling algorithms are detailed in the following section.

4.3.1 Performance Metrics

To evaluate the algorithms under test, the following performance metrics were used: Success Rate (SR), Normalized Schedule Cost (NSC) and Throughput.

- Success Rate (SR): Success rate of each algorithm (SR), calculated as the ratio between the number of simulation runs that successfully met the scheduling deadline and the total number of simulation runs (denoted by

n_{Tot}), defined as:

$$SR = \frac{n\,(k)}{n_{Tot}}, \tag{4.19}$$

where $n(k)$ is the cardinality of the set k and $n_{Tot} = 100$.

- Normalized Schedule Cost (NSC): To compare the monetary cost between the algorithms, the cost of failure in meeting a deadline is considered. For this purpose, a weight is assigned to average cost returned by each algorithm. Let k denote the set of a simulation runs that successfully meets the scheduling deadline, thus the weighted cost is calculated as:

$$Cost_w = \frac{\sum_k Cost_o\,(k)}{SR}, \tag{4.20}$$

where $Cost_o\,(k)$ was defined earlier in equation (3.9) and it is the cost for the experiments that meet the deadline. Thus, the NSC is defined as:

$$Cost_{ns} = \frac{Cost_w}{\min_C}, \tag{4.21}$$

The cost of cheapest schedule (denoted by $\min_C$) is defined as scheduling of all tasks on the cheapest instance according to their precedence constraints.

- Throughput: It is the amount of work that can be done in a given deadline interval by each algorithm. Million Floating Point Operations Per Second (MFLOPS) is used as a measure of the throughput.

4.3.2 Task Selection in PDC

The task selection step is a characteristic of all list based scheduling algorithms. This section presents task selection results in PDC using a set of eight different ranking policies in order to evaluate the importance of and sensitivity to ranking. Figures 4.4 to 4.8 show the results of different task selection policies in PDC as defined in Section 4.2.2.1.

Workflows differ remarkably in their characteristics, including structure, size, computation and communication requirements. Each workflow is constructed

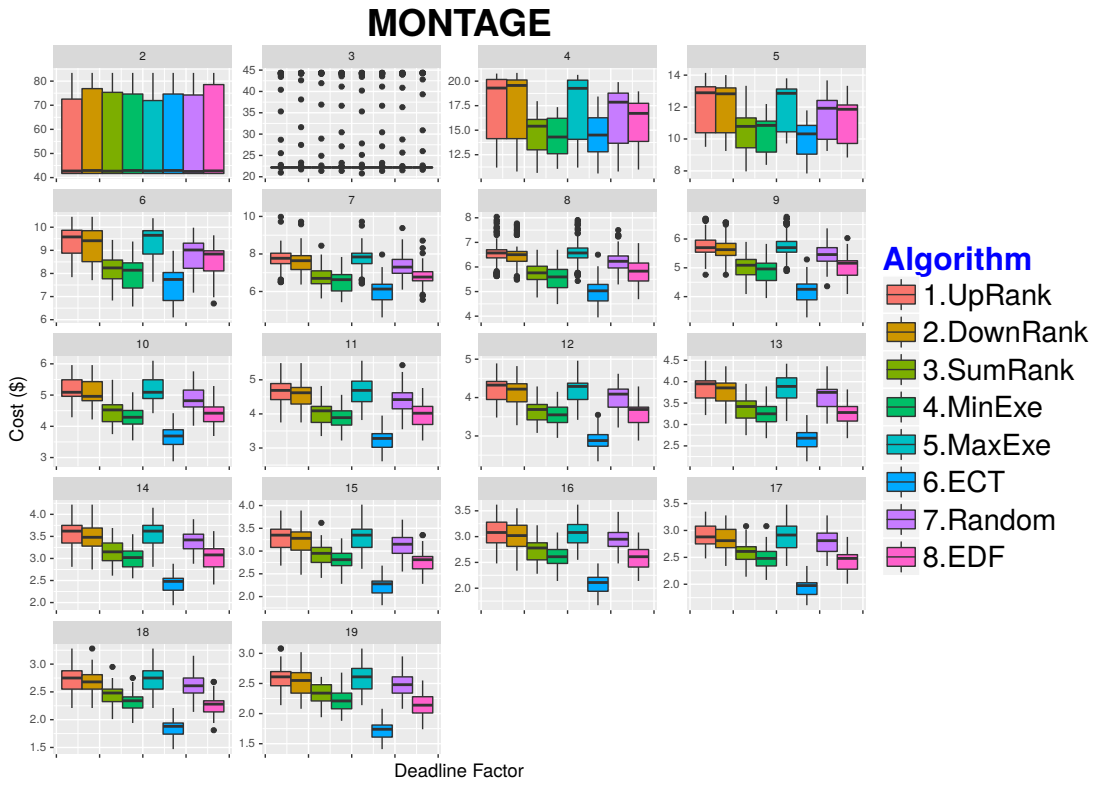


Figure 4.4: Task selection results for Montage

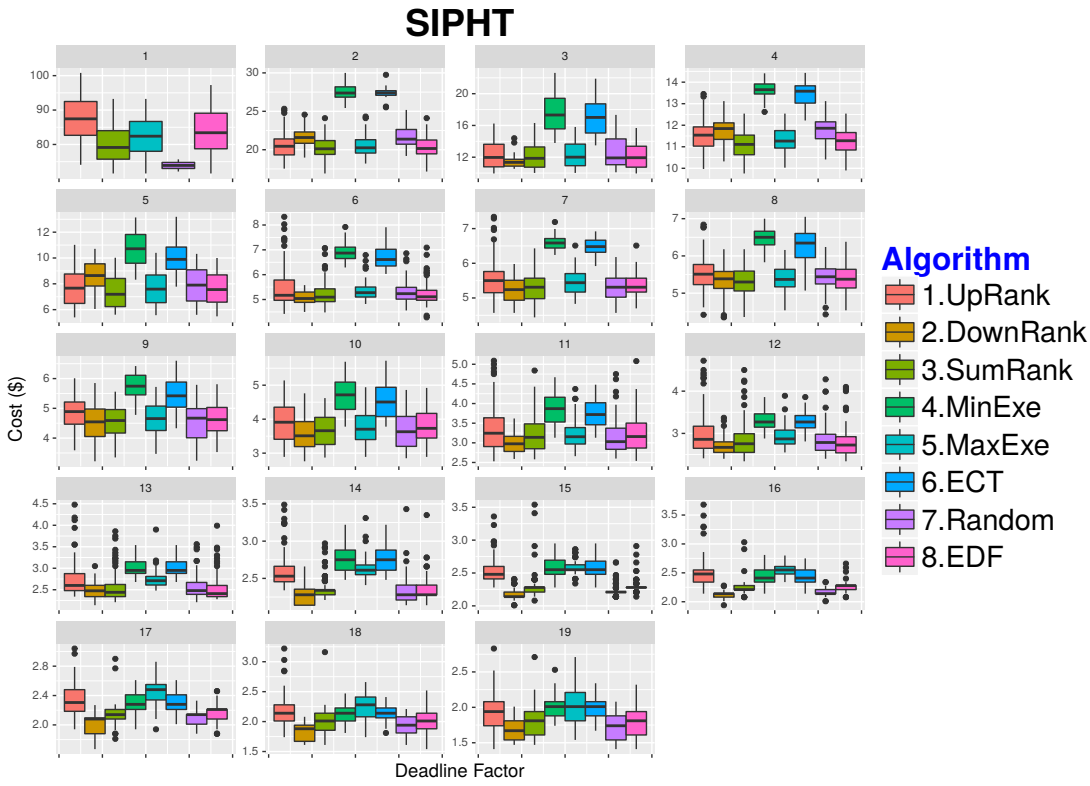


Figure 4.5: Task selection results for SIPHT

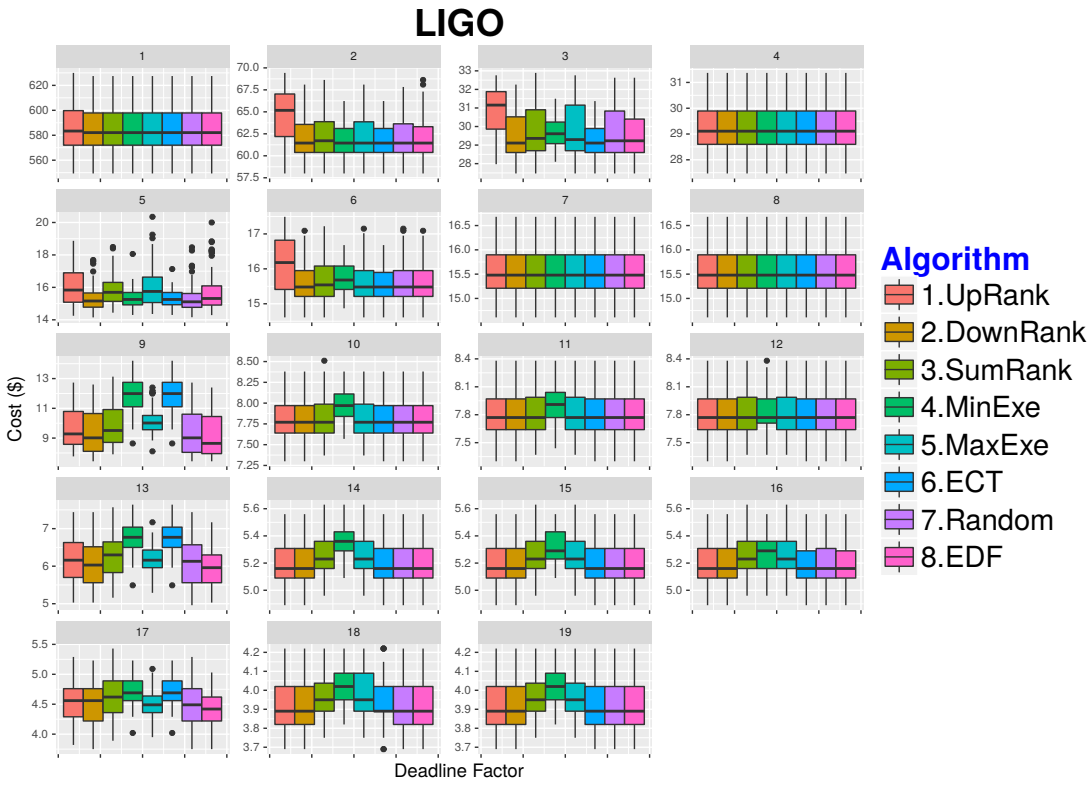


Figure 4.6: Task selection results for LIGO

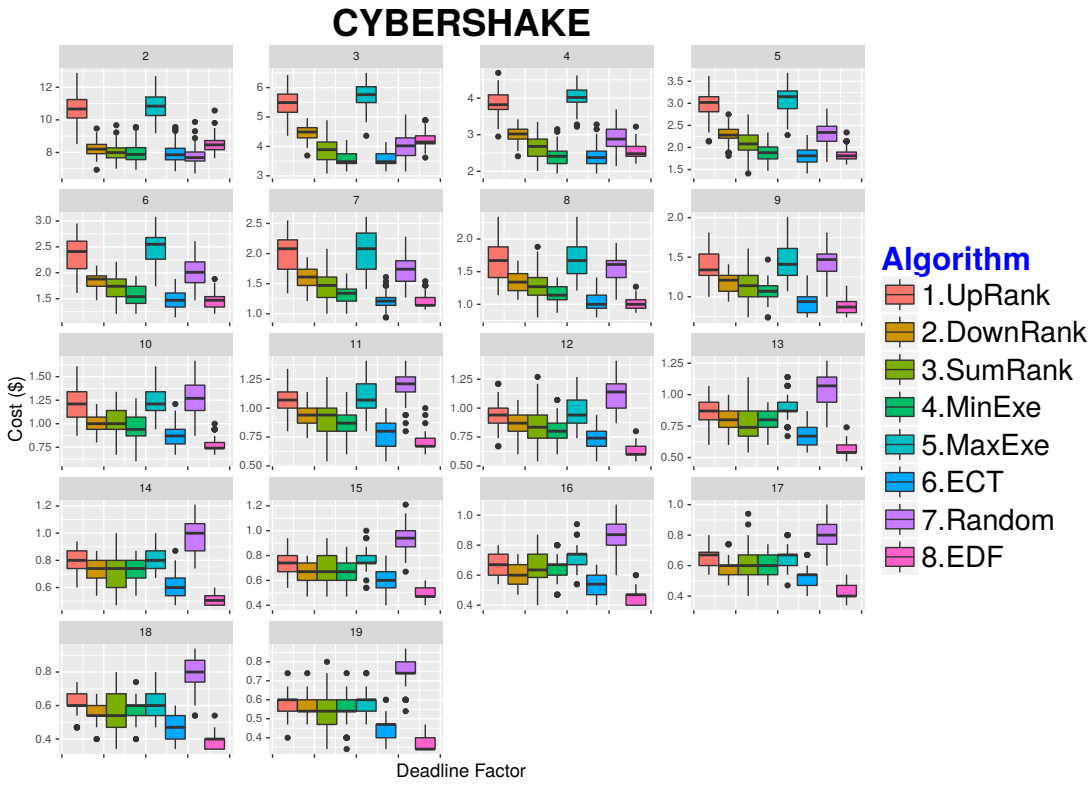


Figure 4.7: Task selection results for Cybershake

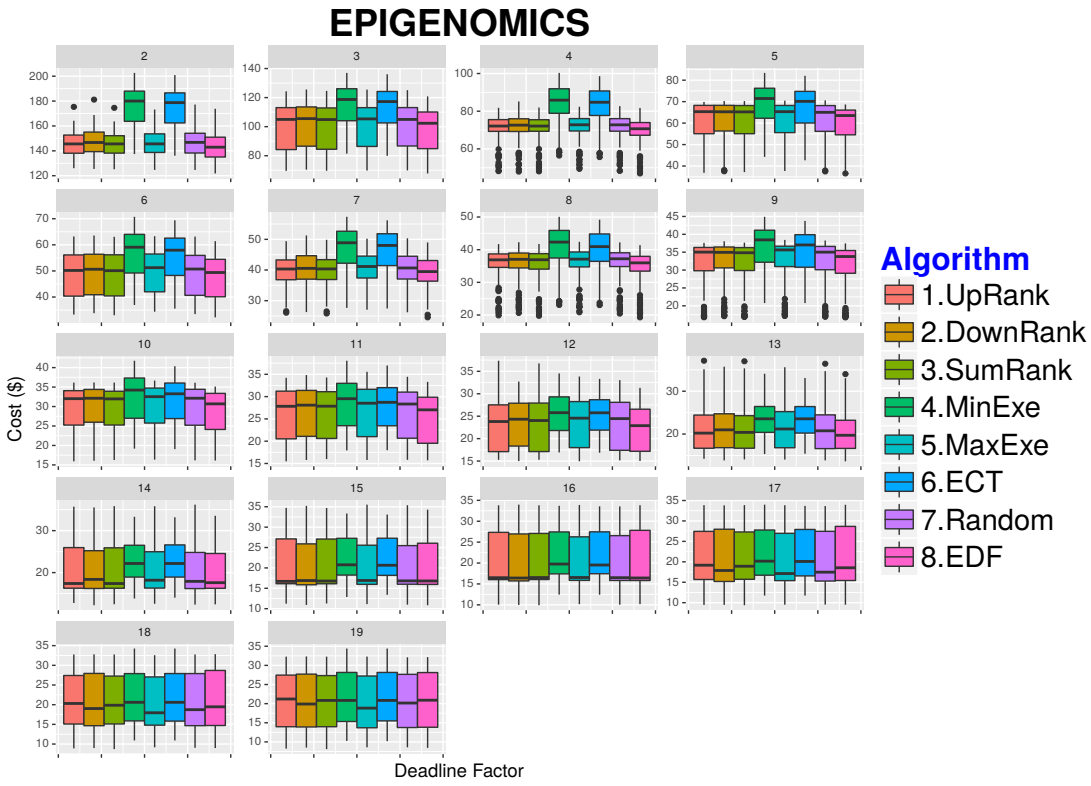


Figure 4.8: Task selection results for Epigenomics

of various components, including process, pipeline, data distribution, data aggregation and data redistribution [75]. The size of scientific workflows varies from small number of tasks taking a few minutes to execute to millions of tasks that require days to execute. Moreover, workflows also differ in terms of data transfer operations. Examples of such transfers are fetching input data, moving intermediate data generated within a workflow, and output data. For each workflow structure, and each deadline factor, 100 distinct Pegasus generated workflows were simulated using CloudSim.

The average execution time and average communication time are used for task ranking by Upward, Downward and Sum rank. Accordingly, the impact of instances that are launched during scheduling are not accounted for by these policies. These are static policies, as they do not change when additional instances are launched by the scheduler.

The Earliest Completion Time (ECT) and Earliest Deadline First (EDF) policies require continual re-computation as execution of a task on a VM leads to changes in ECT and EST for all other tasks on that VM. These are dynamic policies, as they change when additional instances are launched by the scheduler.

The dynamic ranking policies performed best on the Montage (Figure 4.4) and Cybershake (Figure 4.7) workflows, whereas the results were largely ranking agnostic in LIGO (Figure 4.6) and Epigenomics (Figure 4.8). The most interesting result was the SIPHT workflow (Figure 4.5), where the results were largely unpredictable. Nonetheless, overall the dynamic EDF policy produced the lowest costs over all workflows tested.

The results of this set of experiments suggest that the structure of workflows can significantly impact the ranking and scheduling cost. I note that the workflows where the policies performed most consistently had a high degree of structural and runtime symmetry – where each task in such a sequence usually has the same amount of data as input, and in turn generates and distributes equal information as output to its children. Indeed, the workflows for which the policy was agnostic suggest support this conjecture. The most unpredictable workflow, SIPHT, was strongly asymmetric in both structure and runtime. The

structures of these workflows can be found in [21,75].

Although cost differences may seem negligible between some of the policies, in multiples of datasets the variance could be significant. This shows that the task selection order could play a key role in minimizing the cost.

4.3.3 Backfilling in DCCP

In this section, backfilling strategies are evaluated in terms of the number of provisioned instances. Montage, LIGO and Cybershake are data-intensive datasets that usually spend most of their time on data manipulation or data I/O, and have a high number of small tasks. By contrast Epigenomics consists of mostly larger computationally intensive tasks. Due to such task granularities, residuals are used in DCCP to keep tasks operating on the same data on the same instance with the goal of reducing communication cost. A more effective backfilling strategy would ultimately provision fewer instances and thereby reduce cost. The three different strategies (FF, BF and WF) outlined in Section 4.2.4.1 are used. The simulation is limited to a single instance type (type 3, m3.xlarge in Table 4.6) to ensure that the comparison across the algorithms is fair. Two data-intensive workflows, LIGO and Montage, were chosen for evaluation with three different deadline intervals. In Figures 4.9 and 4.10, graphs in left columns show the instance utilization based on the VM creation order, and the right column shows the same sorted by utilization. The X-axis is the total number of instances and the Y-axis is the utilization rate.

The worst fit policy has the best performance because it launches fewer instances, and the number of launched instances has a direct effect on cost. Worst fit reduces further fragmentation of the residuals, leaving larger allocatable blocks. A small set of high utilized VMs leads to lower overall cost in worst fit policy compared to further low utilized VMs in other policies. Overall, the number of instances decreases as the deadline is relaxed. In a case of the moderate and relaxed deadlines in LIGO, interval 12 and 19, it is observed that worst fit needs almost the half the number of VMs. The same observation is true for

MONTAGE with interval 19, worst fit approximately requires one-third of VM numbers compared to others. Considering the benefits of cost saving in worst fit, this policy was used in DCCP algorithms for cost comparison analysis in Section 4.3.4.

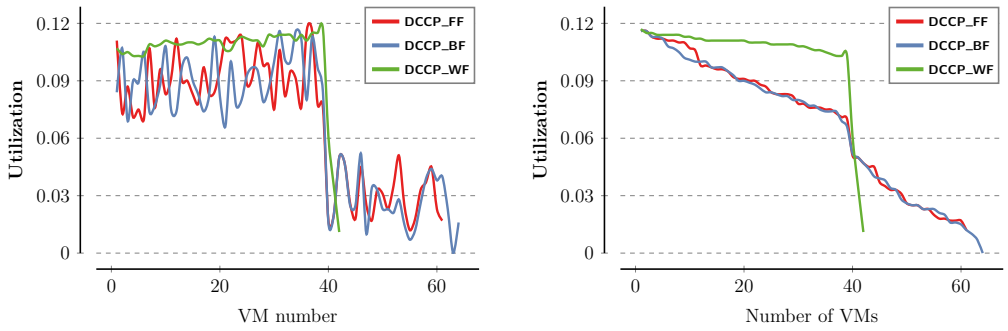
4.3.4 Cost Comparison Analysis

Ultimately the cost vs. deadline performance of each algorithm is the most significant basis for evaluating their performance. In this section, each of the algorithms is evaluated using six different instance types with different characteristics as described in Table 4.6. As expected, experimental results in Figure 4.11 show that the cost of the workflow scheduling generally decreases as the deadline factor increases.

In most cases, the PDC and DCCP algorithms outperform both IC-PCP and JIT, achieving the lowest overall cost over all workflows and deadlines. Like all heuristics, there are points at which their performance is not as good, but these are in the minority, and for small values. For example, in Cybershake, the cost of PDC at most deadlines is approximately 10% that of the cost incurred by the worst performer, JIT.

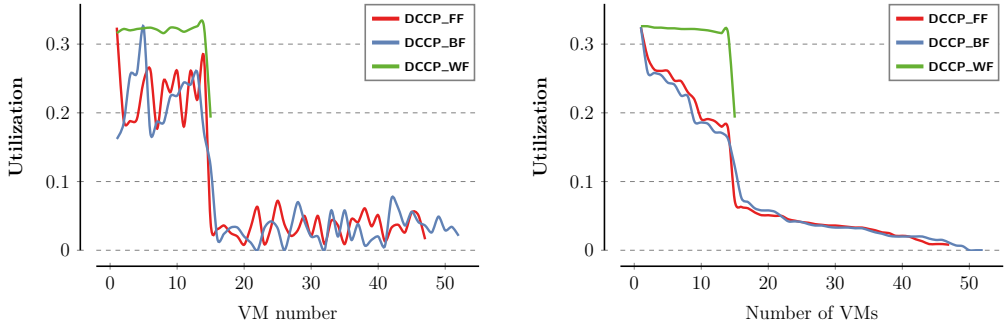
Another interesting result is from the Epigenomics workflow, where while IC-PCP achieves the lowest costs with relaxed deadlines ($12 \rightarrow 20$), this algorithm is also unable to generate any viable schedule, at any cost, for the majority of the tighter deadlines.

As a final observation, from Figure 4.11, the cost of finding a schedule when deadline is tight for Montage and Cybershake is extremely high. It can be explained by considering the structure of these workflows. For example, in Montage, more than 800 parallel tasks out of 1000 in first two levels need to be scheduled. Therefore, when deadline is tight, all algorithms need to lease many instances in parallel to finish elementary tasks that makes the schedule cost very expensive.



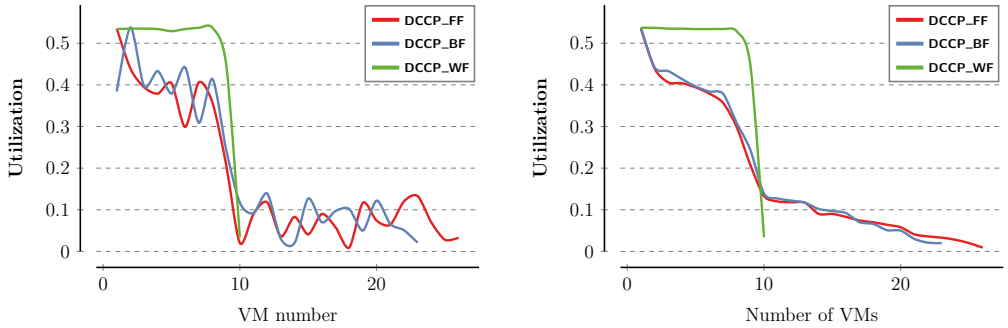
(a) LIGO (interval 5)

(b) LIGO (interval 5) sorted



(c) LIGO(interval 12)

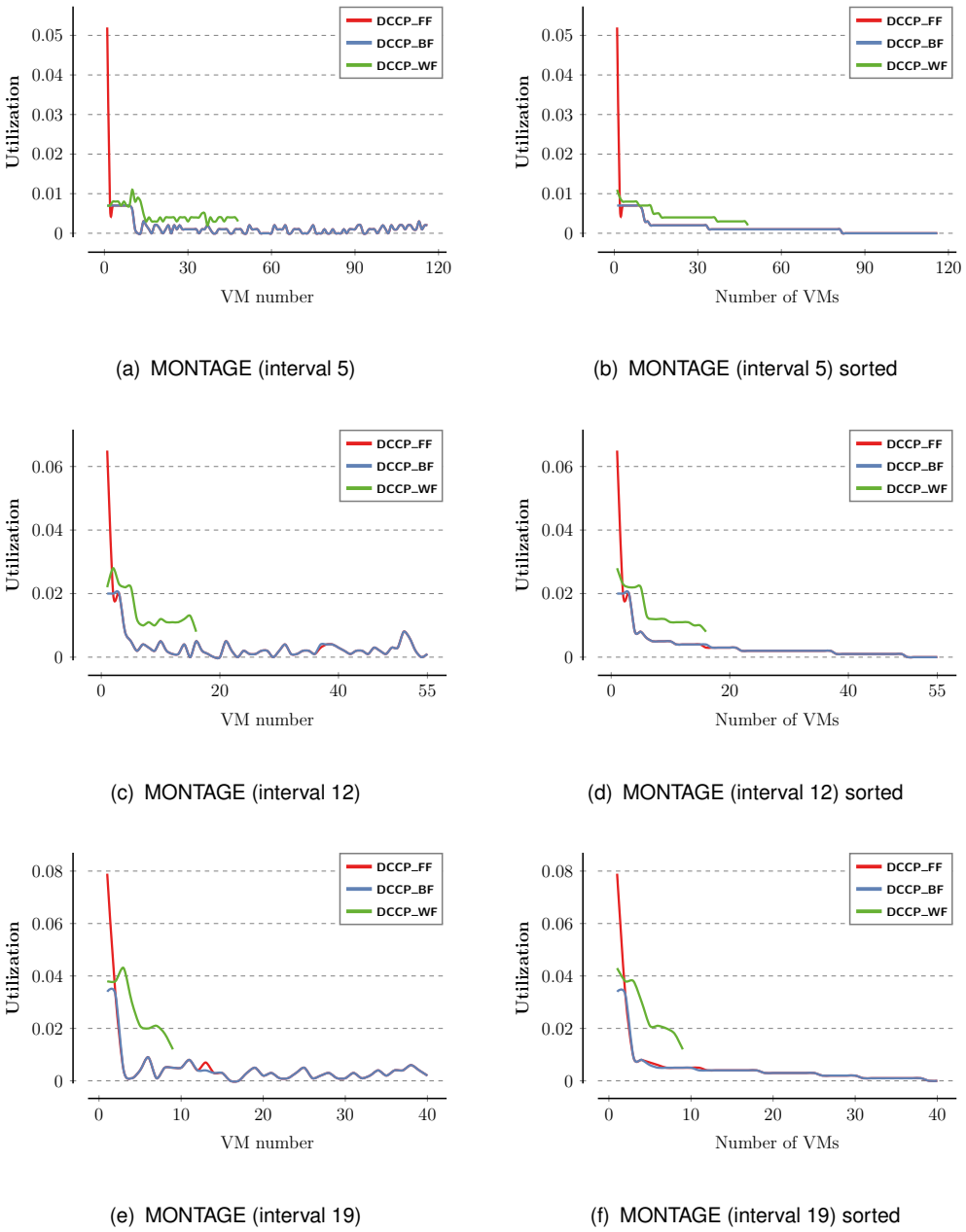
(d) LIGO (interval 12) sorted



(e) LIGO (interval 19)

(f) LIGO (interval 19) sorted

Figure 4.9: VM utilization for three different deadline intervals with Backfilling policies for LIGO.



(a) MONTAGE (interval 5)

(b) MONTAGE (interval 5) sorted

(c) MONTAGE (interval 12)

(d) MONTAGE (interval 12) sorted

(e) MONTAGE (interval 19)

(f) MONTAGE (interval 19) sorted

Figure 4.10: VM utilization for three different deadline intervals with backfilling policies for MONTAGE.

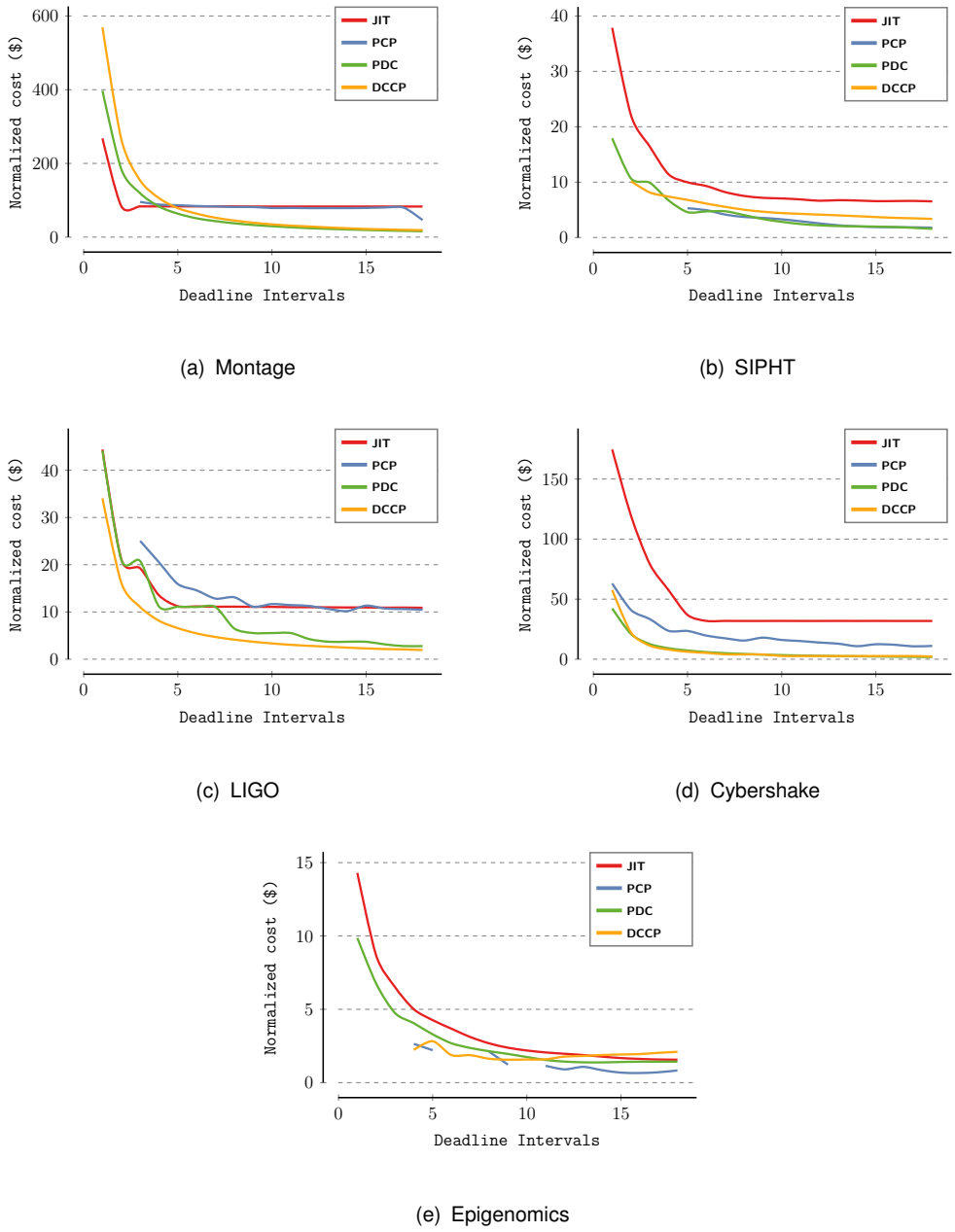


Figure 4.11: Normalized Cost vs. deadline for five different datasets.

4.3.5 Success Rate Analysis

Figure 4.12 shows the relative Success Rate (SR) of each algorithm as the deadline factor, α, is increased from 1 to 20.

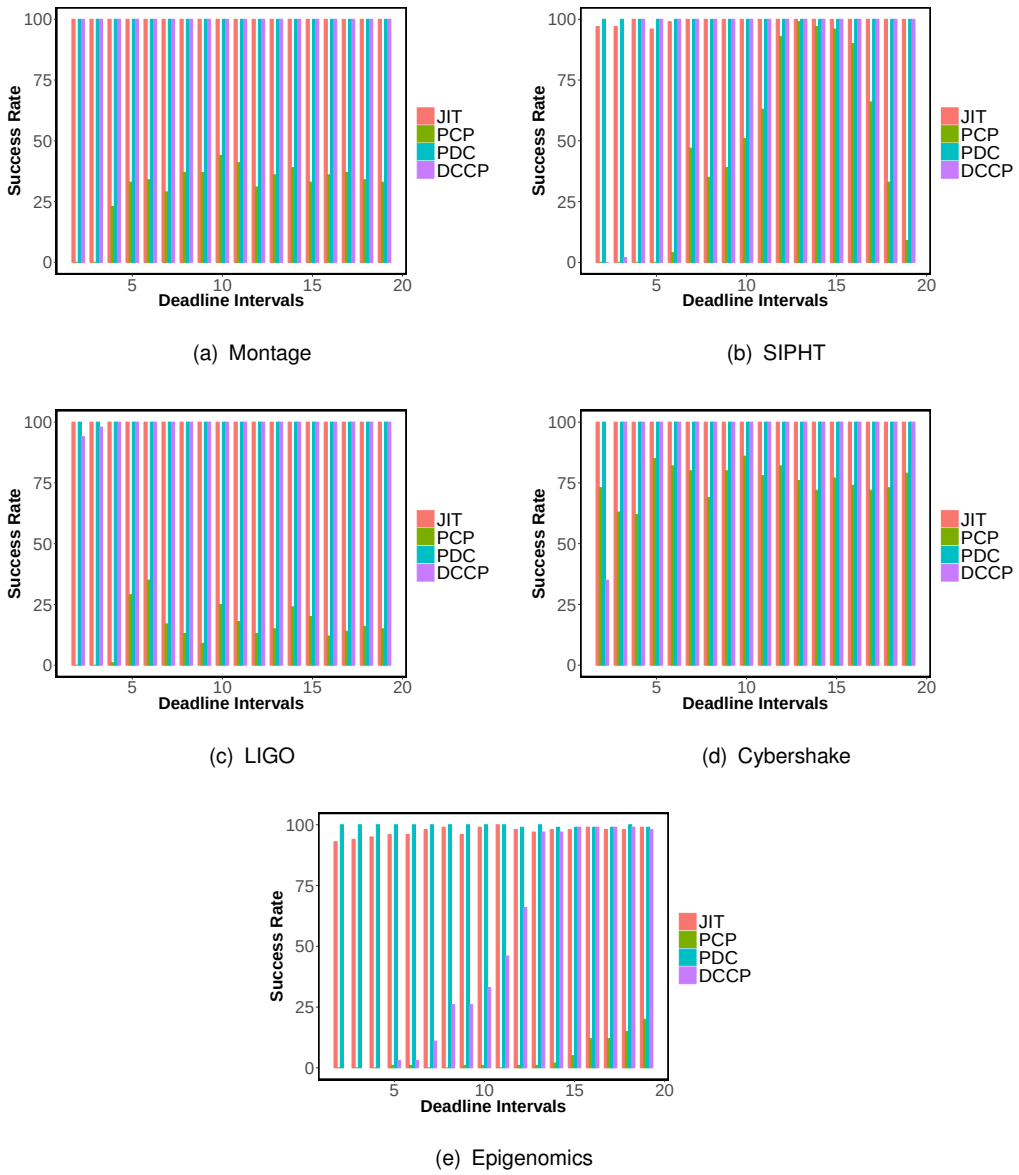


Figure 4.12: Success Rate for five different datasets.

A low success rate indicates that the algorithm cannot find a makespan that meets the deadline (in most datasets). The best overall performers are the PDC and JIT algorithms, which exhibit a success rate of 100% for most deadlines. The relaxing of the deadline causes the success rate of each algorithm to increase except for IC-PCP. Although fewer failures were expected when the deadline is relaxed, the behavior of IC-PCP in different intervals is contrary to these expectations. The highest failure occurs with a deadline factor of $\alpha < 4$ resulting in 100% failure, except in Cybershake. IC-PCP also has the worst performance in Epigenomics, below a deadline factor of 16. IC-PCP can find a schedule for less than 2% of the datasets before its deadline is reached.

The best performance of IC-PCP belongs to Cybershake, which has a success rate of above 60% in all intervals. The DCCP algorithm in all scientific workflows has 100% success when the deadline is more relaxed. The maximum failure in DCCP happens in Epigenomics when deadline is tight. No significant differences were found between the PDC and JIT whereas both are able to finish workflows for more than 95% of the deadlines. Although JIT can find a solution in most of the tested deadlines, JIT generates expensive schedules, as discussed in 4.3.4.

4.3.6 Throughput Analysis

The throughput of each algorithm is displayed in Figure 4.13. The X-axis in Figure 4.13 is cost based on the deadline intervals. The Y-axis is the number of MFLOPS in billions.

In Figure 4.13, the top left corner indicates better performance at a lower cost. Clearly the throughput is dependent on the success rate – and therefore over all intervals and workflows the throughput of the best algorithms, PDC and JIT, are essentially equal. However, the cost difference between PDC and JIT is significant, as shown in the graph. The best performance of DCCP is in MONTAGE and LIGO (both are data intensive workflows), in which DCCP is close to PDC with a similar cost. IC-PCP has the worst performance in almost

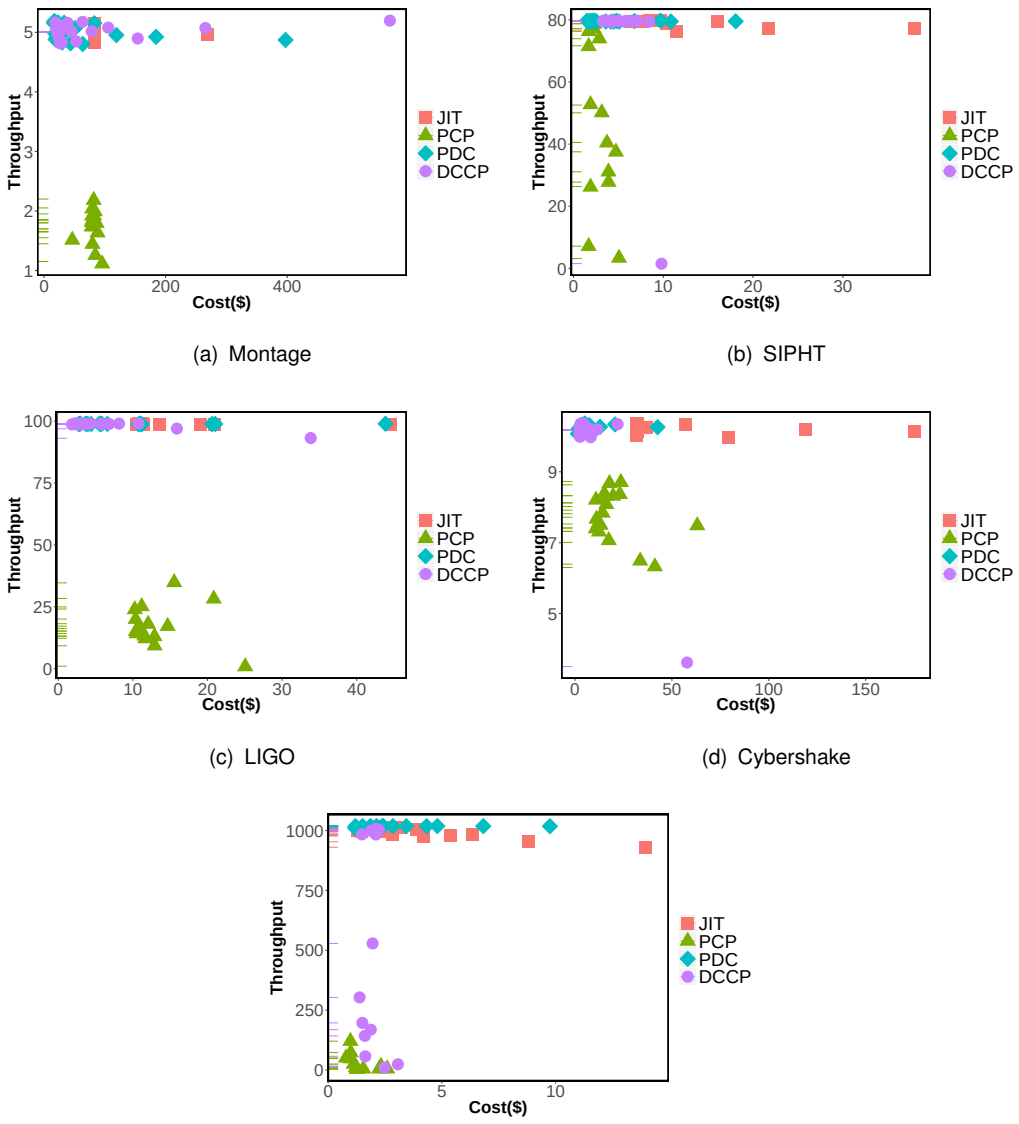


Figure 4.13: Throughput for five different datasets.

all workflows, which is directly related to the low success rate of this algorithm.

4.4 Summary

In this chapter, I introduced new algorithms, Proportional Deadline Constrained (PDC) and Deadline Constrained Critical Path (DCCP). Both algorithms belong to SPDP category (see section 2.4). The PDC operates by maximizing the parallelism in a workflow by separating it into logical levels and then proportionally subdividing the overall workflow deadline over them. The DCCP algorithm is similar to PDC, the main difference is that it also determines the constrained critical path through the workflow in order to co-locate tasks that communicate on the same instance. These algorithms were evaluated (via CloudSim simulation) against two previously published algorithms (IC-PCP and JIT), using a variety of metrics – including success rate, normalized cost, and throughput. I also investigated the influence of the task selection step in the PDC algorithm and backfilling strategies in DCCP. The simulations were conducted using five scientific workflows, Montage, SIPHT, LIGO, Cybershake and Epigenomics, and these were generated by the Pegasus workflow generator.

In terms of cost performance, overall the PDC and DCCP algorithms returned the lowest compute cost, over all workflows and instance configurations. Of particular note, for the Cybershake workflow, PDC returned costs approximately 10% of those incurred by JIT. In terms of success rate and throughput, the best overall performers were the PDC and JIT algorithms, although the JIT algorithm was many times more expensive in terms of cost. I also investigated the effect of eight different policies to evaluate task selection order on scheduling performance in PDC. In the next chapter, I will explore different possible strategies for distributing budget and deadline over a workflow and investigate its impact on the overall cost of the resulting schedule.

The DCCP strategically backfills residuals in provisioned instances. This approach is most effective in data-intensive workflows due to the reduction in data movement. Worst fit resulted in higher utilisation on fewer instances than

either first fit or best fit. Worst fit reduces further fragmentation of the residuals by leaving larger allocatable blocks.

Overall, both algorithms are able to achieve a consistently high success rates and throughput, while in most cases presenting the lowest overall pay-per-use cost.

Chapter 5

Distribution Strategies for Scientific Workflow Scheduling

5.1 Introduction

One critical element is the provisioning of pay-per-use instances, and the subsequent scheduling of workflow tasks over them – in essence we need to complete execution on time and within budget. Scientific workflows vary in size from a couple of tasks to thousands or million of tasks, and these need to be scheduled in parallel and dependency order over, potentially, many instances. This is a workflow scheduling problem – and is inherently *NP*-complete. There are two significant phases to solving such a workflow scheduling problem: selecting the task to be scheduled, and selecting the instance to be provisioned. The choices made in these phases naturally have a significant impact on the overall cost of the resulting schedule and whether it can meet its defined constraint. In this chapter, looking at distributing budget and deadline based on the dependency structure embedded in the workflow is presented. Essentially I transform the workflow into internally dependency free "bags of tasks" (called levels [43]) and I then distribute the workflow budget and deadline over these levels using different strategies.

5.2 Budget Distribution Strategies

One problem in scheduling budget constrained workflow is how to spend that budget for the best performance. This is essentially a budget assignment problem (BAP), and in existing workflow scheduling approaches [15,54] it is shared proportionally based on a subset of the execution characteristic(s) of the task (or cluster of data related tasks) being scheduled, such as execution time, CPU requirements or memory requirements.

I distribute the workflow budget over these levels using six strategies. Three of these strategies are designed explicitly for my means of budget distribution and therefore also represent novel work. I ensure that any budget share that is unused by the level to which it is allocated is trickled down to the next level. For the remainder of this chapter, I will refer to my approach as Budget Distribution with Trickling (BDT).

Based on my results I suggest two hypotheses worthy of further consideration:

Hypothesis 1 *The earliest tasks in the workflow are the most critical when constructing a schedule.*

Hypothesis 2 *Assigning a higher budget to the earliest tasks in a workflow generally leads to a lower makespan.*

5.2.1 The Budget-Aware Scheduling Algorithm

In this section, I describe my budget-aware scheduling algorithm, Budget Distribution with Trickling (BDT). The algorithm is divided into four main phases (each of which relates to a following subsection: 5.2.2– 5.2.5):

(A) Workflow partitioning: The workflow is partitioned into dependency free bags of tasks, called levels.

(B) Budget Distribution: The user-defined budget is then allocated to each defined level using one of six different strategies.

(C) Task Selection: A task is selected based on its priority in the ready list for execution.

(D) Instance Selection: The instances are chosen to meet the available budget.

The focus of this chapter is on budget distribution, the other phases are included for completeness.

5.2.2 Workflow Partitioning

I aim to maximize task parallelism by arranging tasks in levels, where within each level no tasks have dependencies on another in the same level. Each level can therefore be thought of as a bag of tasks (BoT) containing a set of independent tasks.

There are two main algorithms for allocating tasks to different levels: Deadline Bottom Level (DBL) [43] and Deadline Top Level (DTL) [44]. DBL and DBT categorize tasks in bottom-top direction and top-bottom direction, respectively. In this thesis, I use the DBL algorithm to partition tasks into different levels.

I describe the level of task t_i as an integer representing the maximum number of edges in the paths from task t_i to the exit task (see Fig. 5.1). The level number (denoted by N_L) associates a task to a BoT. For the exit task, the level number is always 1, and for the other tasks it is determined by:

$$N_L(t_i) = \max_{t_j \in succ(t_i)} \{N_L(t_j) + 1\} \tag{5.1}$$

where $succ(t_i)$ denotes the set of immediate successors of task t_i. All tasks are then grouped into Task Level Sets (TLS) based on their levels.

$$\mathrm{TLS}(\ell) = \{t_i | N_L(t_i) = \ell\} \tag{5.2}$$

where ℓ is an integer denoting the level in $[1 \ldots N_L(t_{entry})]$.

5.2.3 Budget Distribution

As the principle of distributing budget based on the dependency structure of a workflow (levels) is new, I need to evaluate the performance of a variety of strategies to gauge the value of the approach. I start with the most basic strategies, random and uniform, to provide a baseline comparison. I then explore more complex strategies – width, which is an analogue of prior work on proportional schemes, and then the strategies designed specifically for the BDT approach – height, area and "All in".

The most significant differences in each strategy lie in the calculation of the sub-budget. In some strategies, I have a Budget Factor (BF) that determines a share of the budget for each level. Each sub-budget assigned to a level is termed the level budget.

1. Random: The budget is allocated randomly over the levels in the workflow

2. Uniform: Each level gets a $1/L$ share of the budget, where L is the total number of levels.

3. Height Proportional: Each level gets a share of the user budget proportional to its distance from the entry node. The smaller the distance the greater the share.

4. Width Proportional: Each level gets a share of the user budget proportional to the number of tasks within that level.

5. Area Proportional: Combines width and height strategies to set the budget for each level.

6. All in: Places the entire budget on the entry level and any remainders are trickled down to later levels. This is a refined version of Height proportional, which was formulated as an extreme test of hypotheses 1 and 2, rather than a realistic suggestion. I did not expect this strategy to work

in the general case, as I anticipated a reduced success rate. However, counter-intuitively it returned the best overall performance.

An example is presented to show how the budget is distributed among different levels. Random needs no further explanation, so the example will only detail uniform through "All in" strategies. Figure 5.1 shows the structure of a sample workflow with 10 tasks and their dependencies. In this figure, the left column shows level numbers calculated by equation 5.1. The right column is obtained by counting tasks in each level starts from the exit task. In this example N_{max}=5, which is the maximum level in the workflow. Also, a budget of 165 is assumed.

Each strategy distributes the user budget based on the following basis (see Table 5.1 for the complete set of budget shares):

- Uniform Proportional: Each level gets 165/5 share of budget as the workflow has five levels.

- Height Proportional: Each level is assigned a weighted share of budget relative to its height in the workflow. This is calculated by:

$$L_{weight} = \sum_{k=1}^{N_{max}=5} k = 15.$$

The Budget Factor (BF) is calculated by:

$$BF = \frac{budget}{L_{weight}} = \frac{165}{15} = 11.$$

For instance, level 4, consisting task B and C, is assigned a share of the budget equal to $4 \times BF = 4 \times 11 = 44$.

- Width Proportional: Each level gets a share of budget, depending the number of tasks in corresponding level:

$$BF = \frac{budget}{tasknumbers} = \frac{165}{10} = 16.5.$$

For instance, the budget share assigned to level 4 with two tasks is $2 \times BF = 2 \times 16.5 = 33$.

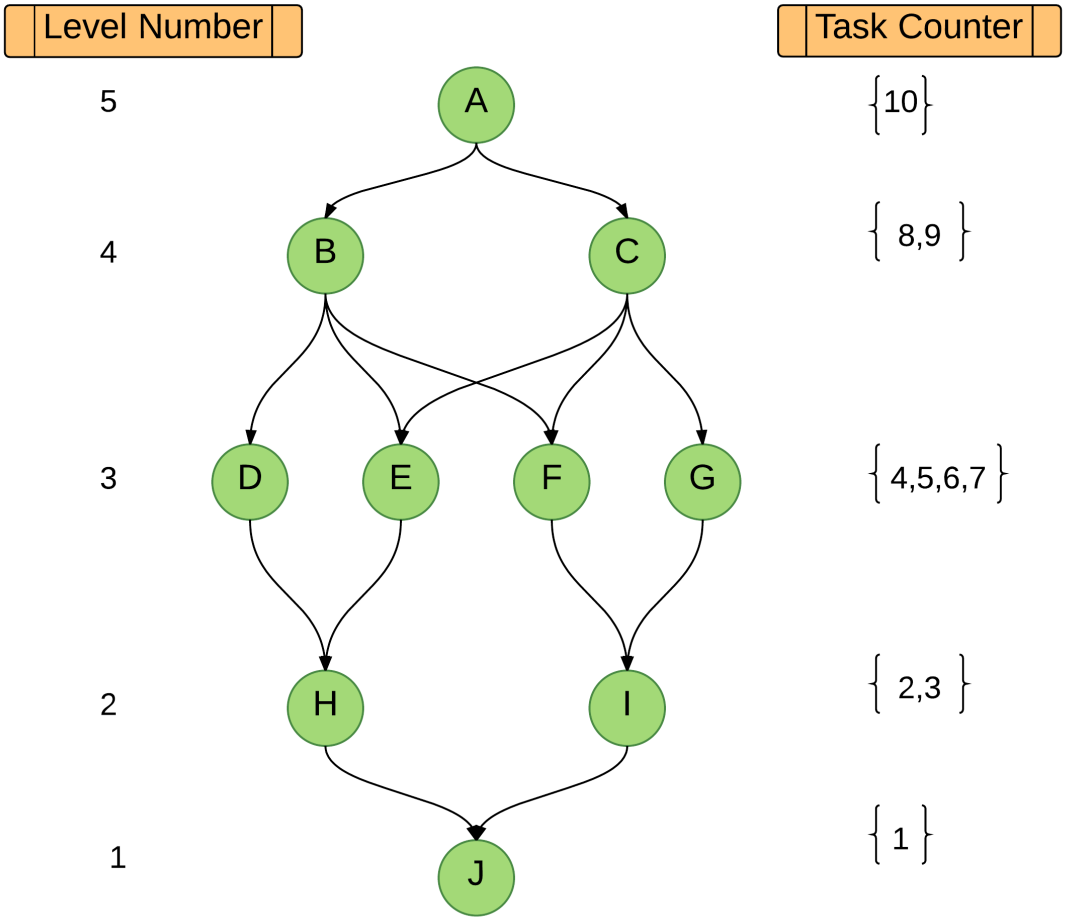


Figure 5.1: A Sample Workflow with 10 tasks.

- Area Proportional: In this strategy, the budget share is allocated to each level is a combination of height and width strategies. Calculated by:

$$L_{weight} = \sum_{k=1}^{10} k = 55.$$

The Budget Factor (BF) is calculated by:

$$BF = \frac{budget}{L_{weight}} = \frac{165}{55} = 3$$

The budget then is distributed based on the sum of numbers in the right column in Fig. 5.1. For example, level 3 is allocated the share $(4 + 5 + 6 + 7) \times BF = 22 \times 3 = 66$.

- All in: The total budget is assigned to level 5. After scheduling all tasks in this level, any spare budget is trickled to the next level.

5.2.4 Task Selection

In BDT, tasks are executed level by level, which means a task can start execution once all tasks in previous levels have been scheduled. There are no dependencies between tasks that are at the same level. Therefore, all of them are ready to execute and are put to the task ready list. To select a task, at first all tasks in the ready list should be prioritized. In this thesis, tasks are prioritized based on their Earliest Start Time (EST).

It could be argued that it is pointless to prioritize task as all ready tasks are independent and their parent tasks have been already scheduled, but this is not the case. Let me trace the execution of a sample graph shown in Fig. 5.2. The number above each edge shows the data transfer time between tasks. Upon executing tasks A, B and C, all their children are placed in the ready list for execution. Note that due to execution of task A and C on the same instance, data are local, and therefore the transfer time is essentially zero. However, task B must wait to receive data from their parent (shown in blue intervals). Task

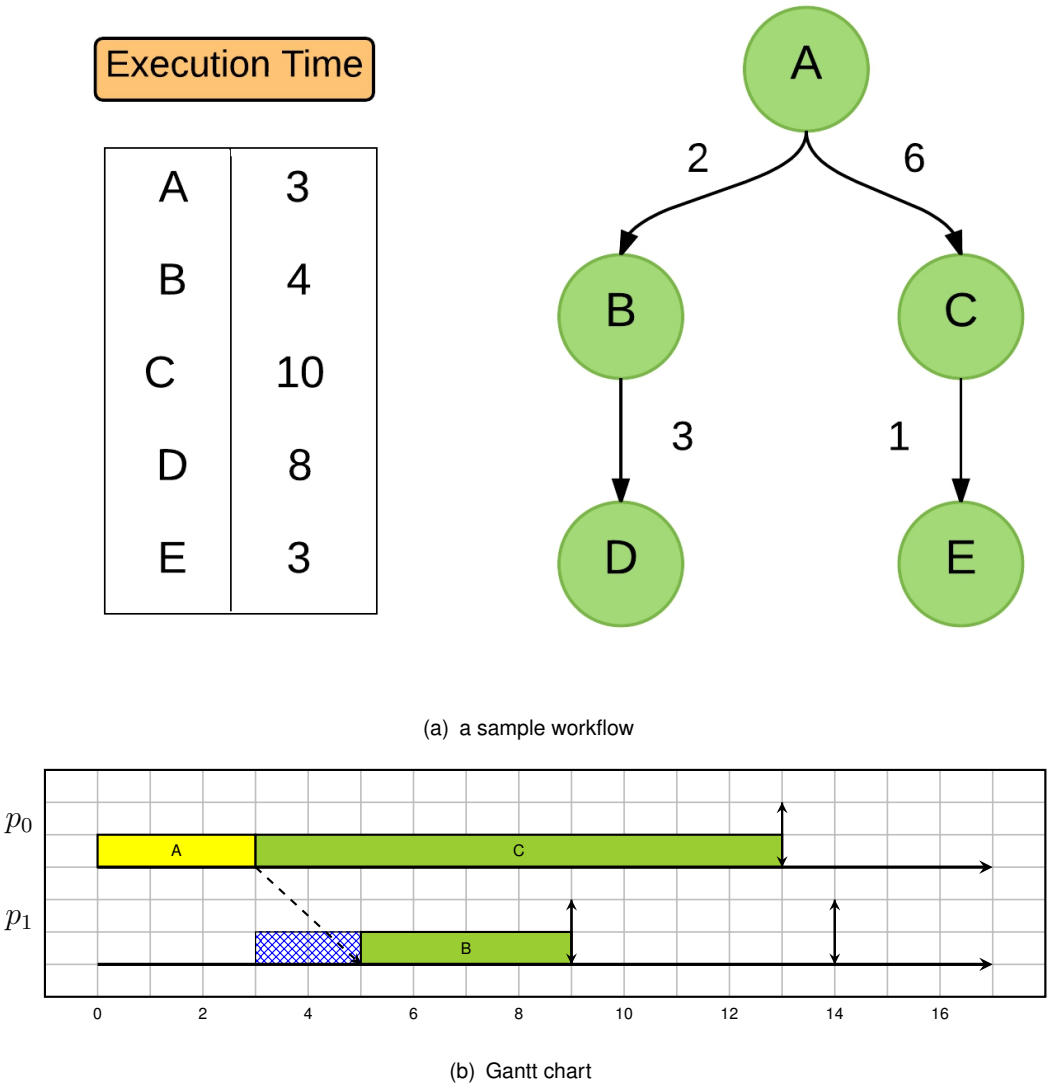


Figure 5.2: Task Selection Example

Table 5.1: Budget distribution for each strategy over each level for a total budget of 165 in Figure 5.1.

	Budget Distribution Strategy				
	Uniform	Height	Width	Area	"All in"
Level 5	$\frac{165}{5} = 33$	$5 \times BF = 55$	$1 \times BF = 16.5$	$10 \times BF = 30$	165
Level 4	$\frac{165}{5} = 33$	$4 \times BF = 44$	$2 \times BF = 33$	$17 \times BF = 51$	0
Level 3	$\frac{165}{5} = 33$	$3 \times BF = 33$	$4 \times BF = 66$	$22 \times BF = 66$	0
Level 2	$\frac{165}{5} = 33$	$2 \times BF = 22$	$2 \times BF = 33$	$5 \times BF = 15$	0
Level 1	$\frac{165}{5} = 33$	$1 \times BF = 11$	$1 \times BF = 16.5$	$1 \times BF = 3$	0

D can start its execution on instance p_0 and p_1 at time 13 and 9, respectively (shown by $\updownarrow$). The earliest time that task E can start on instance p_0 is 13, and on instance p_1 is 14. Therefore, I give a higher priority to task D.

The Earliest Start Time (EST) of a task t_i is calculated on the instance with the shortest execution time and defined as:

$$EST(t_i) = \begin{cases} 0 & , t_i = t_{entry} \\ \max\limits_{t_j \in pred(t_i)} \left\{ EST(t_j) + w_{t_j} + C_{i,j} \right\} & , \text{otherwise}, \end{cases} \tag{5.3}$$

where w_{t_j} is the execution time of task t_j on the fastest instance type. The amount of data transferred from task t_i to task t_j is called communication time (denoted by $C_{i,j}$).

Task selection starts from the level that consists of t_{entry}. After executing the first level, all tasks in the next level are put in the ready list to be scheduled.

5.2.5 Instance Selection

The BDT algorithm attempts to minimize the execution time while meeting the budget. Each level receives a computed budget share, which is the maximum that is able to be spent for the tasks within this level. I start by calculating both the time and the cost of executing each task on each instance type, given by equations 5.4 and 5.5, forming two sets of Cost and Time.

In the equation 5.4, $subBudget$ is a share of the budget that assigned to a level. The cost of scheduling for the current task, t_i, on the instance p_j, is shown by C_i. The minimum cost of executing current task among all instances is C_{best}.

$$\mathrm{Cost}_{t_i}^{p_j} = \frac{subBudget - C_i}{subBudget - C_{best}}. \tag{5.4}$$

In the Time set, the required time for the current task on instance p_j is a function of $\mathrm{ECT}(t_i, p_j)$ and is expressed in equation 5.5. The maximum and minimum completion times of executing the task t_i among all instances are $\mathrm{ECT}(max)$ and $\mathrm{ECT}(min)$, respectively.

$$\mathrm{Time}\ {}_{t_i}^{p_j} = \frac{\mathrm{ECT}(max) - \mathrm{ECT}(t_i, p_j)}{\mathrm{ECT}(max) - \mathrm{ECT}(min)}. \tag{5.5}$$

To find the best instance, I use the Time Cost Trade-off Factor (TCTF) in equation 5.6.

$$\mathrm{TCTF}\ {}_{t_i}^{p_j} = \frac{\mathrm{Time}\ {}_{t_i}^{p_j}}{\mathrm{Cost}\ {}_{t_i}^{p_j}}. \tag{5.6}$$

There is a possibility that the total assigned budget for the level ℓ has already been spent ($subBudget$=0) while there are still some unscheduled tasks. If this condition is true, it makes equation 5.4 zero. Therefore, I cannot launch a new instance as there is no budget left. Note that the value of equation 5.6 becomes zero as well.

When an instance is provisioned, the user is charged for the entire billing interval, even if the task completes before the end of the interval. One way to reduce the cost of executing tasks is by using leftover capacity (residuals) in

provisioned instances that have been already paid for. Therefore, if other tasks can execute on an existing instance with a residual, their execution costs can be considered zero. Moreover, the utilization of cloud resources depends on how tasks are placed together. Instance fragmentation and resource wastage occurs if tasks are not packed efficiently. The BDT algorithm utilizes these residuals for executing ready tasks, which reduces makespan at no additional cost.

An important concept in my algorithm is trickling down unused budget, and this is expressed by equation 5.7. I define Spare Budget (SB) as the amount of money remaining after allocating all tasks in the level ℓ. I then add the left-over to the next level ($\ell + 1$).

$$SB = subBudget_{\ell} - \sum_{t_i \in \mathrm{TLS}(\ell)} C_i. \tag{5.7}$$

5.2.6 Evaluation

Five common scientific workflows were used: Cybershake, Montage, LIGO, Epigenomics and SIPHT to evaluate the performance of my algorithms under realistic load, the characteristics of which have been analyzed in [21]. The CloudSim [85] was used, and configured with one data-center and six different instance types. The characteristics of these instance types are based on the EC2 instance configurations presented in Table 4.6. The average bandwidth between instances was fixed to 20 MBps, based on the average bandwidth provided by AWS [86]. The processing capacity of an EC2 unit was estimated at one Million Floating Point Operations Per Second (MFLOPS) [87]. The pricing model and other characteristics were explained in section 4.3.1.

To evaluate the budget sensitivity of the BDT algorithm and associated strategies, different budget ranges were considered for the scientific datasets from lowest possible through to sufficient. The lowest possible budget to schedule a workflow is given by equation 5.8.

$$Lowest_{cost} = \sum_{\forall t_i \in G} Cost_{t_i}^{p_j}, \tag{5.8}$$

where p_j is the cheapest instance. To achieve this, all tasks are executed on the instance with the lowest cost (the cheapest instance). This assignment gives us the lowest possible cost required for executing a workflow, irrespective of finishing time. Using this lowest cost, the minimal budget is calculated as follows:

$$budget = \alpha * Lowest_{cost} \quad 1 < \alpha < 10. \tag{5.9}$$

The budget range starts from 1.5 to consider minimum budget with increasing step length of 0.5. The EC2 instances charge hourly basis from the time of provisioning, even if the instance is only used for a fraction of that period. The simulations were run with lease times of 15, 30, 45 and 60 minutes to evaluate the sensitivity of the algorithm to the length of the lease.

In order to compare performance in respect to workflow size, workflows with 1000 tasks were considered. The Pegasus workflow generator [21] was used to create representative synthetic workflows with the same structure as real world scientific workflows (Cybershake, Montage, LIGO and SIPHT). Again, for each workflow structure, and each budget range, 100 distinct Pegasus generated workflows were scheduled in CloudSim, and these results are detailed in the following section.

5.2.7 Analysis of LIGO

Table 5.2: Example of computed budget distribution for each strategy over each level of a LIGO for budget range=5 and budget=7.035.

Budget Distribution Strategy											
		Uniform		Height		Width		Area		"All in"	
	Task Numbers	sub Budget	Spare Budget	sub Budget	Spare Budget	sub Budget	Spare Budget	sub Budget	Spare Budget	sub Budget	Spare Budget
Level 6	229	1.173	0.01	2.01	0.15	1.61	0.01	2.85	0.01	7.035	0.001
Level 5	229	1.173	0.03	1.675	0.01	1.61	0.03	2.11	0.03	0.001	0.001
Level 4	21	1.173	0.05	1.34	0.03	0.14	0.006	0.15	0.01	0.001	0.001
Level 3	250	1.173	0.01	1	0.02	1.75	0.01	1.39	0.02	0.001	0.001
Level 2	250	1.173	0.03	0.67	0.009	1.75	0.02	0.51	0.06	0.001	0.001
Level 1	21	1.173	0.05	0.335	0.02	0.14	0.02	0.003	0.05	0.001	0.001
Achieved Makespan		938.97		798.71		798.61		682.6		603.93	

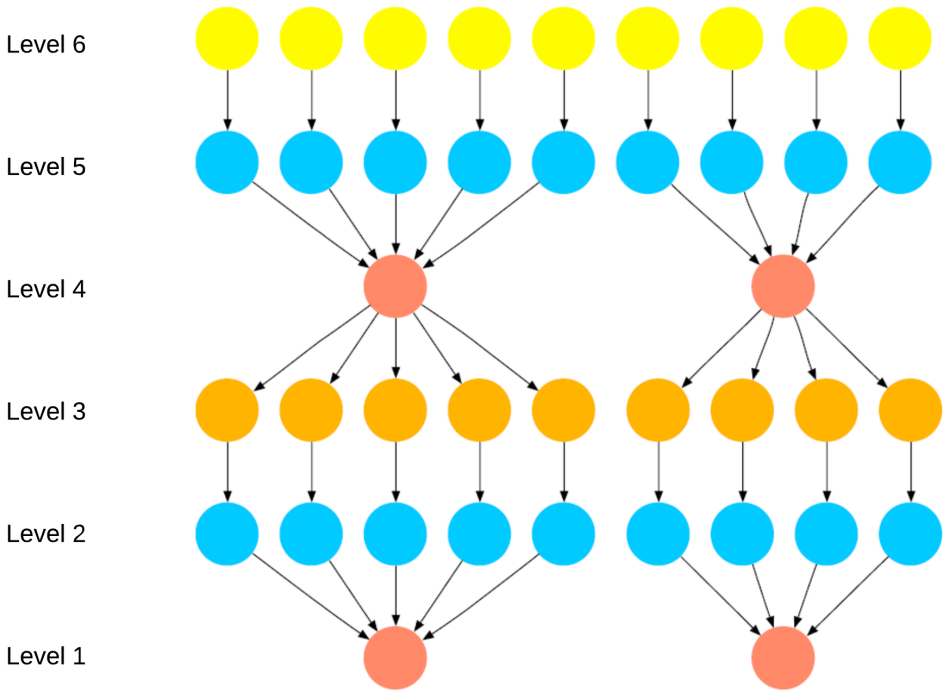


Figure 5.3: A simple structure of LIGO with six levels

Table 5.3: VM requested types by different strategies based on Table 5.2.

Budget Distribution Strategy					
Vm Type	Uniform	Height	Width	Area	"All in"
m3.medium	1	1	6	2	0
c4.large	6	3	2	2	1
c3.xlarge	6	3	2	3	1
m4.2xlarge	0	2	2	2	0
c4.4xlarge	6	2	2	2	0
c3.8xlarge	0	2	2	2	4
#VMs	19	13	16	13	6
Total Cost	6.985	7.006	7.026	6.968	7.035

The Laser Interferometer Gravitational Wave Observatory (LIGO) attempts to detect gravitational waves produced by various events in the universe as per Einstein's theory of general relativity [21]. A simple LIGO-like workflow with six levels is shown in Figure 5.3.

For a LIGO workflow with 1000 tasks, Table 5.2 gives the levels with the corresponding task's numbers. The budget range of 5 with the corresponding budget of 7.035 is selected to evaluate the budget distribution strategies. The share of budget that each level gets and the spare budget are given in Table 5.2. For instance, after scheduling of all tasks in the first level by using the Height Proportional strategy, the spare budget of 0.15 (indicated in red) is added to assigned budget of the next level (shown in blue), resulting in the total budget of 1.69 for that level (the budget trickling concept). The last row in Table 5.2 shows the makespan that was achieved by each strategy.

It is also interesting to look at which instance types and how many of each were provisioned by each strategy; this is given in Table 5.3. The most significant observation is that the "All in" strategy allocates a small number of powerful instances, reducing the data transfer costs and achieving the lowest makespan as given in Table 5.2. From users' perspective, finding a schedule with a lower makespan for a given budget is the main concern, while from the providers' perspective, maximizing utilization is the main concern. Very few re-

search papers report the total number of instances that a workflow needs when provisioning. Although all BDT strategies meet the budget (only possible due to trickle down) and find a schedule, the type and the amount of requested VMs are very different, as shown in Table 5.3. Launching too many instances does not lower makespan. Instead, it causes high scheduling overhead and low instance utilization. Therefore, the budget distribution strategy has a direct impact on resource utilization – which is significantly important.

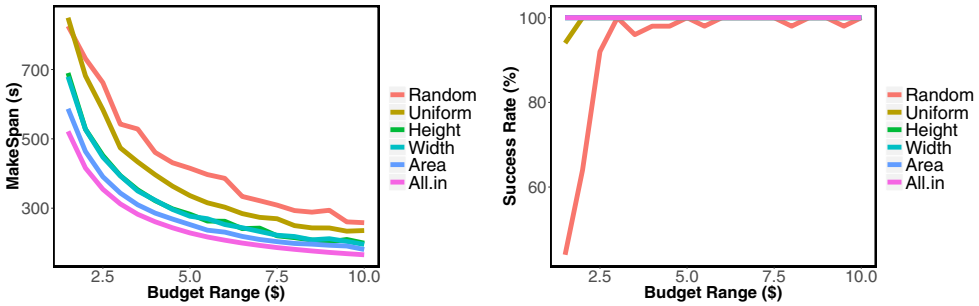
The makespan and success rate of LIGO for four specified lease times are shown in Fig 5.4. The "All in" strategy again performs significantly better than other strategies for each of the defined budget factors from low to high values. The random strategy has the worst performance for both makespan and success rate. Another observation is that the general trend of all strategies for different instance lease times is largely similar.

5.2.8 Other workflows

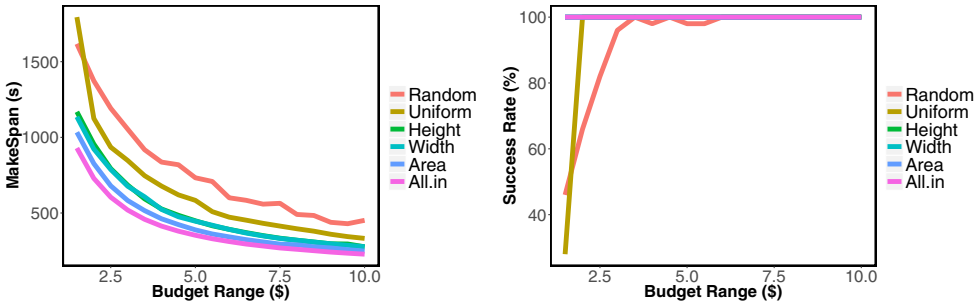
The makespan and success rates of other workflows are presented in Fig. 5.5. The general trend is that by increasing the budget, a scheduler can launch more costly services, which in turn leads to a lower makespan.

In terms of makespan, for almost all workflows, the "All in" strategy has the best performance, including lowest makespan and highest success rates. An interesting observation is that the structure and type of workflow appear to have a significant effect on success rate. For instance, in CYBERSHAKE, the success rates of most of the strategies are generally poor; this is likely due to them being a data intensive workflow - the "All in" strategy of few, high power instances minimized the cost of data movement and produced the best makespan and success rate.

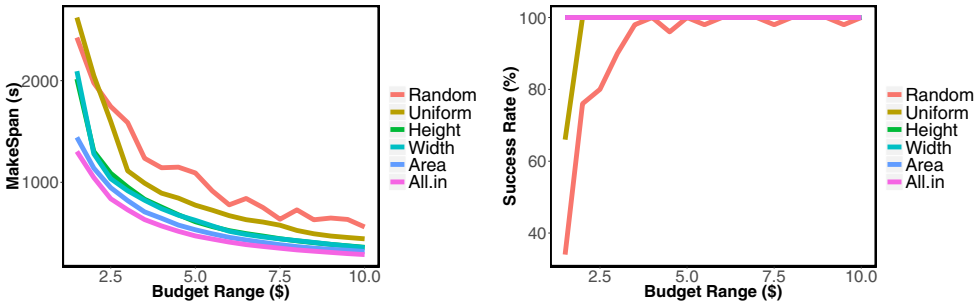
Overall, a budget bias towards the early tasks (and levels) of a workflow appears to produce better overall performance.



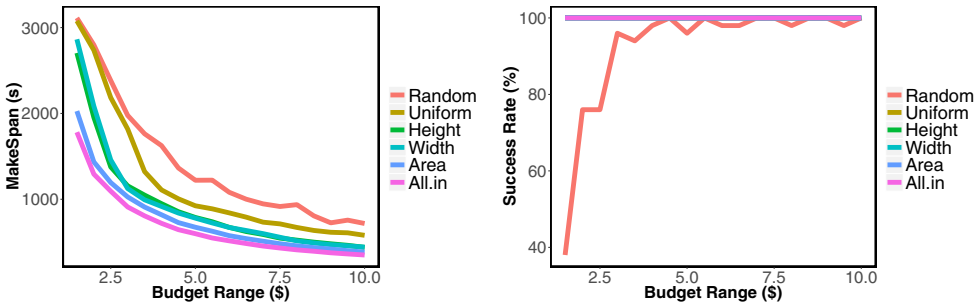
(a) Lease Time=15



(b) Lease Time=30

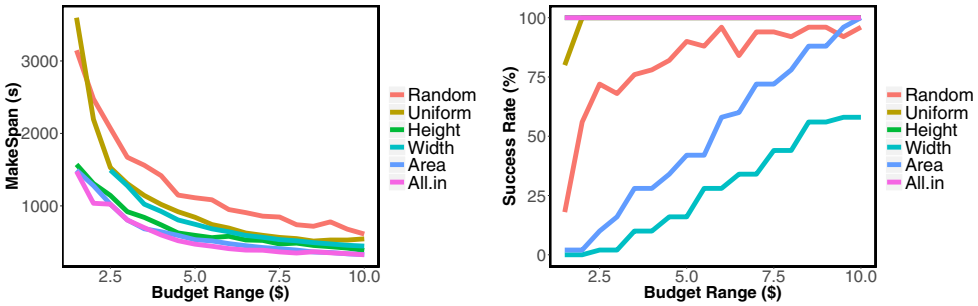


(c) Lease Time=45

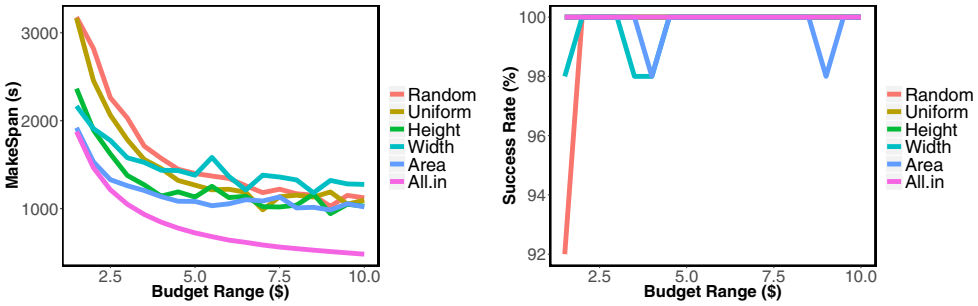


(d) Lease Time=60

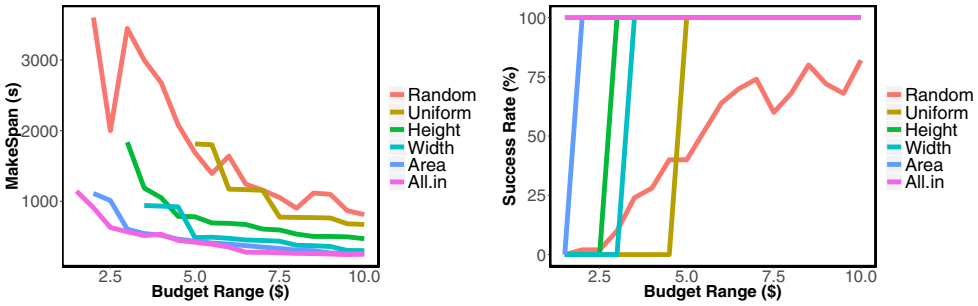
Figure 5.4: Makespan and Success rate performance executing LIGO for all strategies for lease time of 15, 30, 45 and 60.



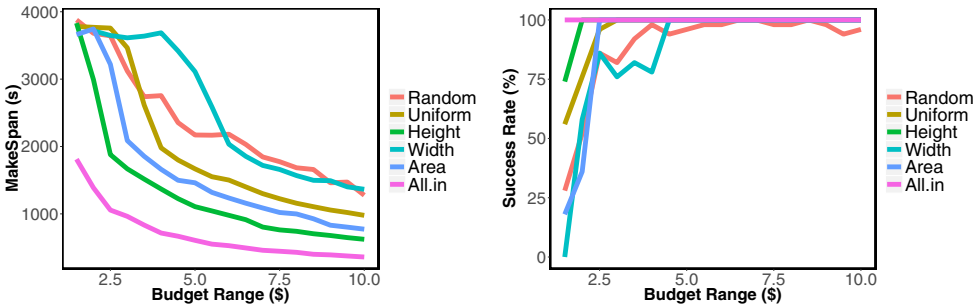
(a) CYBERSHAKE



(b) EPIGENOMICS



(c) MONTAGE



(d) SIPHT

Figure 5.5: Makespan and Success rate performance executing of workflows for all strategies for lease time 60.

5.3 Deadline Distribution Strategies

One approach to distribute deadline is to divide or distribute the deadline over the workflow as sub-deadlines to ensure a more manageable constraint satisfaction problem, and then to provision the instances to meet these sub-deadlines [89, 90]. This leaves us with two fundamental questions:

- What are the different possible ways to distribute a deadline over a workflow?
- How do these different strategies affect cost?

To answer these questions in this chapter, I explore a range of strategies for distributing a deadline over a workflow.

5.3.1 The DDR algorithm

In this section I outline my new Deadline Distribution Ratio (DDR) algorithm. In addition to the two phases of task selection and instance selection, when I perform deadline distribution, I also need to introduce two additional phases, giving:

(A) Workflow partitioning: The workflow is partitioned into dependency-free bags of tasks, called levels.

(B) Deadline Distribution: The user-defined deadline (T_D) is divided and distributed between levels. Each level gets its own level deadline. All tasks in the same level have the same level-deadline.

(C) Task Selection: A task is selected based on its priority in the ready list for execution.

(D) Instance Selection: The instances are chosen to meet the available deadline.

Critically, this section also introduces six base and 14 combined distribution strategies that are evaluated in this thesis as part of the overall DDR algorithm.

5.3.2 Workflow partitioning

Each task is categorized in a level by analyzing its synchronization requirements to maximize the achievable parallelism from the workflow. This categorizing is in the same way as I described in 5.2.2.

5.3.3 Deadline Distribution

5.3.3.1 Initial Estimation

The initial estimated deadline for each level (ℓ) is calculated as:

$$InitialT_{sd}(\ell) = \max_{t_i \in \mathrm{TLS}(\ell)} \{\mathrm{ECT}(t_i)\}, \tag{5.10}$$

where $\mathrm{ECT}(t_i)$ denotes the Earliest Completion Time (ECT) of task t_i over all instances, and the ECT is defined as

$$\mathrm{ECT}(t_i) = \max_{\ell \in pred(t_i)} \{InitialT_{sd}(\ell), EST\,(t_i)\} + w_{t_i}, \tag{5.11}$$

where $EST\,(t_i)$ is defined in equation. 3.7, $pred\,(t_i)$ denotes the set of predecessors of task t_i; w_{t_i} denotes the minimum execution duration for task t_i, and ℓ indicates the parent level t_i. The task, t_{entry}, has no predecessors; its ECT is equal to zero. In equation 5.10, the maximum ECT of all tasks in a level is used as the overall estimate for that level. This duration is effectively the absolute minimum time that is required for all tasks in a level to complete execution in parallel.

5.3.3.2 Deadline Distribution Strategies

The main idea of deadline distribution is simple, distribute deadline among different levels and try to complete its execution before any assigned sub-deadline so that the global deadline can be met.

The baseline deadline distribution strategies are:

- Random (R): The deadline is allocated randomly over the levels in the workflow.

- Uniform (U): Each level gets a $1/L$ share of the deadline, where L is the total number of levels.
- Height Proportional (H): Each level gets a share of the user deadline proportional to its distance from the entry node.
- Width Proportional (W): Each level gets a share of the user deadline proportional to the number of tasks within that level.
- Area Proportional (A): Combines width and height strategies to set the deadline for each level.
- Length Proportional (L): Each level gets a share of the deadline non-uniformly based on the proportion of its length. Levels with longer tasks gain a larger share of the user deadline.

I now present an example to show how the deadline is distributed among different levels. Random needs no further explanation, so the example will only detail the rest of baseline strategies. Figure 5.1 shows the structure of a sample workflow with 10 tasks and their dependencies. In this figure, the left column shows level numbers calculated by equation 5.1. The right column is obtained by counting tasks in each level, starting from the exit task. In this example N_{max}=5, which is the maximum level in the workflow. Also, a deadline of 165 is assumed. This number is arbitrary and serves as an example.

Each strategy distributes the user deadline based on the following basis (see Table 5.4 for the complete set of deadline shares and assigned sub-deadline to each level):

- Uniform: Each level gets 165/5 share of deadline as the workflow has five levels.
- Height Proportional: Each level is assigned a weighted share of deadline relative to its height in the workflow. This is calculated by:

$$L_{weight} = \sum_{k=1}^{N_{max}=5} k = 15.$$

The Deadline Factor (DF) is calculated by:

$$DF = \frac{deadline}{L_{weight}} = \frac{165}{15} = 11.$$

For instance, level 4, consisting task B and C, is assigned a share of the deadline equal to $4 \times DF = 4 \times 11 = 44$.

- Width Proportional: Each level gets a share of deadline, depending the number of tasks in corresponding level:

$$DF = \frac{deadline}{tasknumbers} = \frac{165}{10} = 16.5.$$

For instance, the deadline share assigned to level 4 with two tasks is $2 \times DF = 2 \times 16.5 = 33$.

- Area Proportional: In this strategy, the deadline share allocated to each level is a combination of height and width strategies. It is calculated by:

$$L_{weight} = \sum_{k=1}^{10} k = 55.$$

The Deadline Factor (DF) is calculated by:

$$DF = \frac{deadline}{L_{weight}} = \frac{165}{55} = 3$$

The deadline is then distributed based on the sum of numbers in the right column in Fig. 5.1. For example, level 3 is allocated the share $(4 + 5 + 6 + 7) \times DF = 22 \times 3 = 66$.

- Length Proportional: After calculating the estimated deadline value for all levels, I distribute the user deadline among all tasks non-uniformly, based on a deadline proportion denoted by $\propto_{deadline}$ in equation 5.12:

$$\propto_{deadline} = \frac{T_D - InitialT_{sd}(1)}{InitialT_{sd}(1)}, \tag{5.12}$$

where $InitialT_{sd}(1)$ is the level that contains the exit task.

I then compute the length of each level deadline as a function of this deadline proportion to each level as follows:

$$T_{sd}(\ell) = InitialT_{sd}(\ell) + (\propto_{deadline} \times |InitialT_{sd}(\ell)\ |)\ . \tag{5.13}$$

Intuitively, the levels with longer executing tasks gain a larger share of the user deadline.

In Table 5.4, two rows are specified for each level. The first row indicates the share of deadline that each level gets. The second row is the final value of assigned sub-deadline to each level that is calculated based on the cumulative value with previous levels. Clearly, the sub-deadline value for level 1 should be equal to the total deadline. For instance, in Height strategy in level 4, the share of deadline is 44 (indicated in red) and the assigned sub-deadline is 99 (shown in blue).

I also combine the baseline strategies in order to produce different variant strategies. For example, combination of the three strategies, Initial Estimation(E), Width(W) and Length Proportional (L), gives me a new strategy to distribute deadline, which is named EWL. In this strategy, first, estimated deadline for all levels are calculated. Then, the leftover deadline is distributed based on the combination of Width and Length strategies. This gives a total of 14 different strategies that are evaluated in the experiments. Some of the generated strategies are shown as exemplars in Figure 5.6.

5.3.4 Task Selection

In each step of my algorithm, those tasks that are ready to execute are put in the task ready list. A task is ready when all of its parents have been executed and all its required data have been provided. Therefore, there are no dependencies between tasks that are at the same level. In order to select a proper task for execution, all tasks in the ready list are prioritized with their Earliest Start Time (EST). The EST is the soonest possible time that a task can start its execution,

Deadline Distribution Strategy					
		Uniform	Height	Width	Area
Level 5	share of deadline	$\frac{165}{5} = 33$	$5 \times DF = 55$	$1 \times DF = 16.5$	$10 \times DF = 30$
	sub-deadline	33	55	16.5	30
Level 4	share of deadline	$\frac{165}{5} = 33$	$4 \times DF = 44$	$2 \times DF = 33$	$17 \times DF = 51$
	sub-deadline	66	99	49.5	81
Level 3	share of deadline	$\frac{165}{5} = 33$	$3 \times DF = 33$	$4 \times DF = 66$	$22 \times DF = 66$
	sub-deadline	99	132	115.5	147
Level 2	share of deadline	$\frac{165}{5} = 33$	$2 \times DF = 22$	$2 \times DF = 33$	$5 \times DF = 15$
	sub-deadline	132	154	148.5	162
Level 1	share of deadline	$\frac{165}{5} = 33$	$1 \times DF = 11$	$1 \times DF = 16.5$	$1 \times DF = 3$
	sub-deadline	165	165	165	165

Table 5.4: Deadline distribution for each strategy over each level for a total deadline of 165 in Figure 5.1.

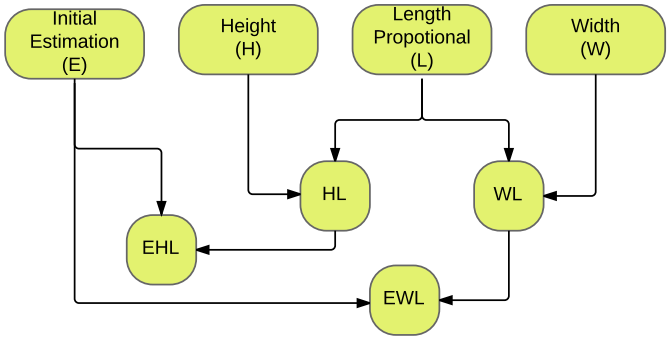


Figure 5.6: Producing different strategies based on baseline strategies.

which depends on the finish time of its parent. The Earliest Start Time (EST) of a task t_i is calculated on the instance with the shortest execution time and defined as:

$$EST(i) = \begin{cases} 0 & , t_i = t_{entry} \\ \max\limits_{t_j \in pred(t_i)} \left\{ EST(t_j) + w_{t_j} + C_{i,j} \right\} & , \text{Otherwise,} \end{cases} \tag{5.14}$$

where w_{t_j} is the execution time of task t_j on the fastest instance type. The amount of data transferred from task t_i to task t_j is called communication time (denoted by $C_{i,j}$). The EST on all VMs is calculated for each task. The task that starts first will be the best candidate for execution.

5.3.5 Instance Selection

The Instance Selection phase aims to identify the most suitable instance to execute tasks. The Instance Selection decision for each task aims to minimize the total cost of workflow execution while also attempting to meet the task's sub-deadline.

I introduce an objective function referred to as Instance Comparative Ratio

(ICR).

$$ICR_{t_i}^{p_j} = \frac{Level_{deadline}^{\ell_{t_i}} - \mathrm{ECT}(t_i, p_j)}{TaskCost_{t_i}^{p_j}}, \tag{5.15}$$

where $Level_{deadline}^{\ell_{t_i}}$ is the deadline that is assigned to the level that contains the task t_i. The time needed for the current task, (t_i), to execute on the instance p_j is calculated by $\mathrm{ECT}(t_i, p_j)$. The ECT is the earliest time that a task can complete execution on an instance (as defined in equation 5.11). The value in the numerator of equation 5.15 assesses the differences between the sub-deadline and earliest completion time of the current task on the instance p_j. The denominator is the cost of current tasks, which is defined in 3.8.

Most cloud providers like Amazon Web Services (AWS) Elastic Compute Cloud (EC2) charge users based on 60 minute intervals. When an instance is provisioned, the user is billed for the entire billing interval even if the task completes before the end of the interval. Therefore, if other tasks can execute on the same instance within the remaining interval, their execution cost can be considered zero. Thus, when allocating instances I prioritize selecting instances with remaining idle billing intervals. The first step of the algorithm explicitly considers instances that have no cost to execute the current task as well as ensuring that the earliest completion time does not exceed the level deadline. The instance with minimum ECT is then selected (the fastest one).

If no instances can be found in the previous step, the algorithm provisions a new instance based on the highest ICR value. In tight deadlines, there is a possibility that cheaper instances cannot meet the task level's sub-deadline. Therefore, the value of ICR is negative as its numerator is negative. If this condition is met, more expensive instances are candidates for the current task. In fact, the ICR value is trying to adjust the cost and time for current task among all instances. For cost minimization purpose, most of the proposed algorithms like [91] try to schedule a task on a cheapest available instance (slower) while still meeting its assigned sub-deadline. However, with this strategy, tasks can take a much longer time if the resources are slower and this leads to some delay in the EST of its children. To avoid this, the key concept of introducing ICR in

5.15 is to make a trade-off between time and cost.

5.3.6 Evaluation

Public cloud provides instance types containing various amounts of CPU, memory, storage and network bandwidth at different prices. A resource model is used based on the Amazon Elastic Compute cloud, where instances are provisioned on demand. The pricing model and other characteristics are explained in section 4.3.1.

The fastest schedule (denoted by FS) is calculated as a baseline schedule. Effectively, this baseline is the fastest possible execution – ignoring costs and is computed as:

$$FS = \sum_{t_i \in CP} (w_i^j), \tag{5.16}$$

where w_i^j is the computation cost of task t_i on the fastest instance p_j. A Critical Path (CP) is the longest path from the entry to exit node of a task. If all tasks on the CP of a workflow are executed on the fastest instance type, the fastest schedule will be reached.

The deadline is defined as a function of the fastest schedule, and this deadline is expressed in equation 5.17 in which the deadline varies from tight to moderate to relaxed:

$$\text{deadline} = \alpha * FS, \quad 0 < \alpha < 20. \tag{5.17}$$

The deadline factor α starts from 1 to consider very tight deadlines (typically approaches the fastest schedule) and is increased by one up to a value of 20, which results in a very relaxed deadline.

5.3.7 Experimental Results

In this section, the performance comparison of 14 deadline distribution strategies in my algorithm is presented. Then, the evaluation of the presented algo-

rithm with other state-of-the-art algorithms [46,49,89] is presented in 5.3.9. The main metrics evaluated in my comparison are the cost and success rate (SR).

To compare the monetary cost between the algorithms, the cost of failure in meeting a deadline is considered. For this purpose, a weight is assigned to average cost returned by each algorithm. Let k denote the set of a simulation runs that successfully meets the scheduling deadline, thus the weighted cost is calculated as:

$$Cost_w = \frac{\sum_k Cost_o(k)}{SR}, \tag{5.18}$$

where $Cost_o(k)$ is the cost for the experiments that meet the deadline and SR denotes the success rate.

5.3.8 Cost comparison for distribution strategies

Legend	Algorithm Name
R	Random
U	Uniform
H	Height
W	Width
A	Area
L	Length
EU	Estimation & Uniform
EH	Estimation & Height
EW	Estimation & Width
EA	Estimation & Area
EL	Estimation & Length
EHL	Estimation & Height & Length
EWL	Estimation & Width & Length
EAL	Estimation & Area & Length

Table 5.5: Definition of legends in Fig. 5.7.

In order to observe the precise behavior of each strategy, for each dataset, ranges for deadline from 5 to 10 were selected. This range was chosen because it gives us more detail about the performance of all strategies in simplifying interpretation of results. Table 5.5 lists the legends notation in Fig. 5.7.

The first observation is that different strategies have unstable trends in tested datasets. For example, the Width (W) strategy has the worst performance in CYBERSHAKE while gaining the almost lowest cost in MONTAGE. Similarly, the EL strategy resembles the same trend in MONTAGE and SIPHT. This is attributed to the fact that workflows differ remarkably in their characteristics, including structure, size, computation and communication requirements.

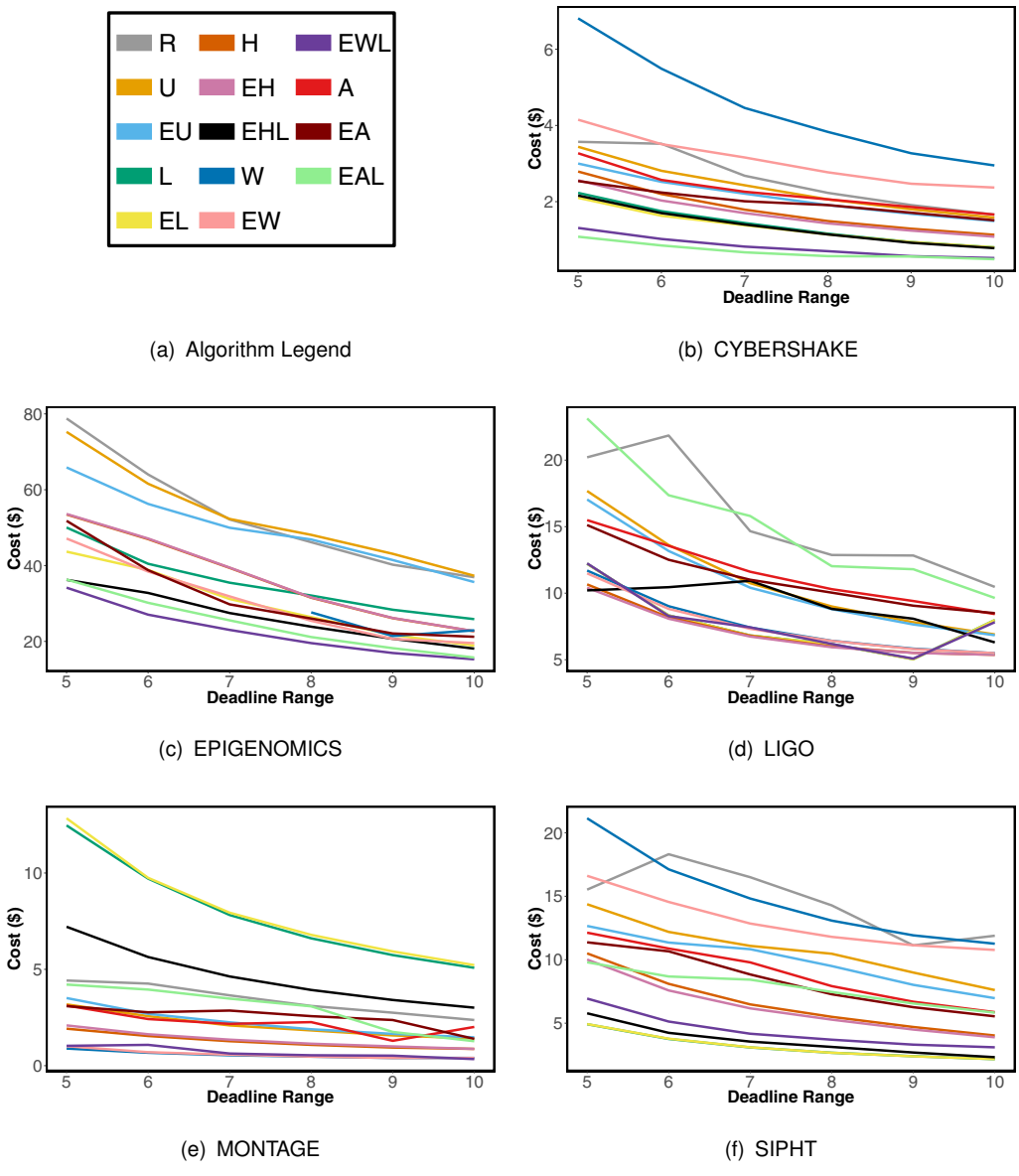
Workflows consists of various components, including process, pipeline, data distribution, data aggregation and data redistribution [75]. The behavior of the EWL strategy indicates a good performance with different datasets. This strategy considers the number of tasks in a level and length of a level, simultaneously. In the next section, this strategy is considered in the distribution of the deadline in my algorithm.

While cost differences may seen negligible between some of the strategies, for executing big datasets, the differencing could be significant. This shows that the deadline distribution strategy could play a key role in minimizing the cost.

5.3.9 Cost Comparison with other algorithms

Three state-of-the-art algorithms, IC-PCP [46], JIT [49] and PDC [89], were selected in order to compare with the DDR algorithm. The cost and success rates of five scientific workflows in different deadline intervals are presented in Fig. 5.8.

The general results show that the JIT underperforms others in terms of cost in all cases. For almost all workflows, the DDR algorithm has the best performance, including lowest cost and highest success rate. There are instances where their performance is poorer, but these are few and far between. For example, in CYBERSHAKE and EPIGENOMICS with very tight deadlines, the



(a) Algorithm Legend

(b) CYBERSHAKE

(c) EPIGENOMICS

(d) LIGO

(e) MONTAGE

(f) SIPHT

Figure 5.7: Cost vs. deadline for different deadline distribution strategies.

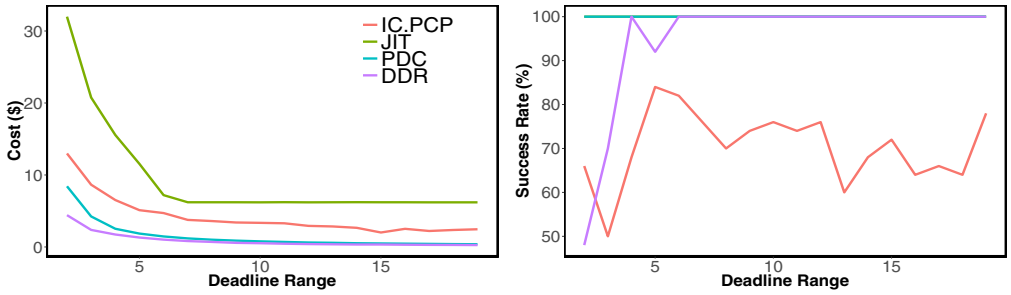
DDR algorithm is unable to schedule some workflows.

The IC-PCP algorithm has the worst success rate, especially in tight deadlines. The relaxing of the deadline should lead increasing the success rate of each algorithm. However, the behavior of IC-PCP in different intervals is contrary to expectations. The highest failure occurs in EPIGENOMICS, while even in relaxed deadline, IC-PCP can find a schedule for less than 25% before its deadline is reached. JIT performs very well in most of the deadline intervals with nearly 100% success rate. However, it is the most expensive algorithm for finding a schedule over all workflows and instance configurations. The main reason for this behavior in JIT is how instances are selected [49].

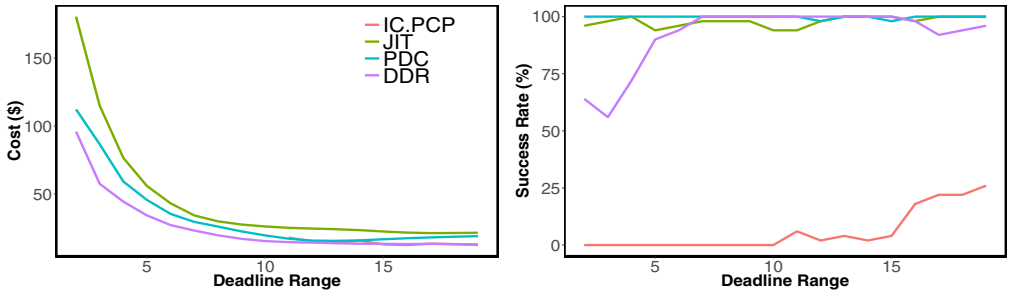
5.4 Summary

In this chapter, I introduced the Budget Distribution with Trickling (BDT) and Deadline Distribution Ratio (DDR) algorithms. The time complexity of the two presented algorithms is of the order $\mathcal{O}(n^2 p)$ which is calculated similar in principle to section 4.2.5.

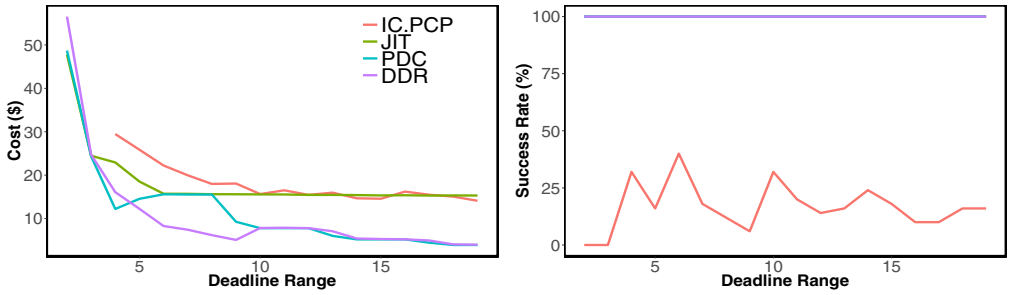
In BDT, the makespan and success rate of six strategies were evaluated using five real-world workflows with different lease times on instances. The width and uniform strategies are largely analogous to existing budget distribution approaches, and these are outperformed in makespan and success rate by Area and "All in" BDT strategies. Overall, the strategy that performed the best for all workflows in terms of both success rate and makespan was "All in". This strategy was proposed largely to test the hypothesis that biasing the budget distribution to the earliest levels was beneficial in terms of reducing makespan. "All in" took this to an extreme by assigning the entire budget to the first level and relying wholly on the trickle-down mechanism to distribute budget to later levels. Counter-intuitively, "All in" gave the best success rates – and the best performance for data-intensive workflows due to the resulting data locality – from fewer more powerful instances. The main finding of my research is the importance of biasing budget distribution to early levels in a workflow. This



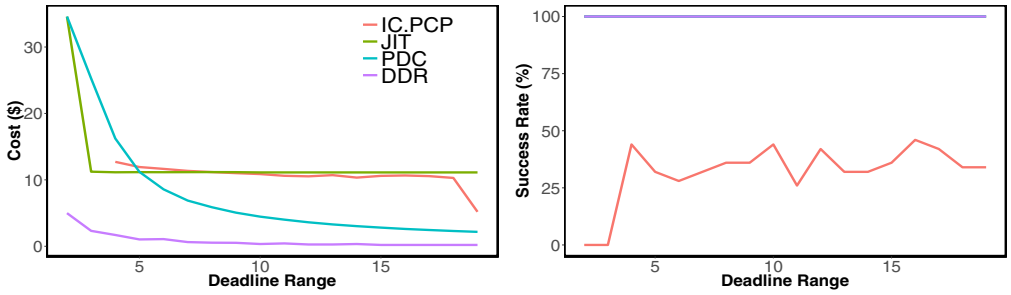
(a) CYBERSHAKE



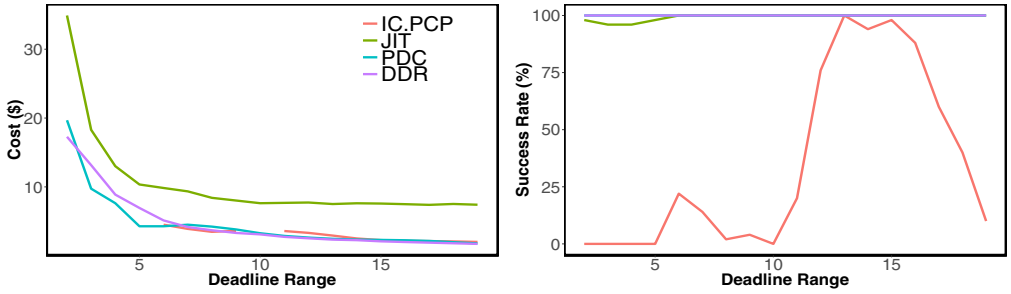
(b) EPIGENOMICS



(c) LIGO



(d) MONTAGE



(e) SIPHT

Figure 5.8: Cost vs. deadline for five different datasets.

leads to finding a schedule with a lower makespan. From this I have extrapolated two hypotheses that need further investigation. In addition, these results suggest that gaining a better understanding of workflow types (compute and data intensive) and workflow structure can lead to better budget distribution and likely scheduling in general.

The DDR focuses on addressing different ways of deadline distribution in scientific workflow. For that purpose, I introduced new strategies for deadline distribution and assessed the effectiveness of these strategies in terms of cost and success rate. Some strategies exhibit performance that is strongly dependent on the workflow size and structure, including process, pipeline, data distribution, data aggregation and data redistribution. In general, the strategy which takes into consideration the execution time of each level as well as number of tasks in the level, yields the lowest cost.

In summary, this chapter presents new notions and strategies for distributing budget and deadline based on the dependency structure inherent in workflows – levels. When multiple constraints are requested by users, the problem of workflow scheduling becomes even more challenging. In the next chapter, I will introduce a new algorithm for scientific workflow scheduling to consider budget and deadline as constraints, simultaneously. The importance of these constraints was discussed in 3.1.4. Moreover, the strategies that performed the best in this chapter in terms of distributing budget and deadline will be applied in the next chapter.

Chapter 6

Budget Deadline Constrained Workflow Scheduling

6.1 Introduction

A major challenge of the cloud paradigm for e-Science lies in limiting or minimising [9] costs while maintaining or even accelerating throughput. In fact, scheduling workflows and provisioning cloud resources naïvely can have a significant financial penalty – especially in dynamic markets such as the Amazon spot market [11]. The majority of research into scheduling and provisioning has focused on one of cost or time, as the fundamental workflow scheduling problem is *NP*-complete and optimising multiple constraints, such as cost and time, over a non-uniform set of unlimited resources is nontrivial. Indeed, this complexity leads to long computation times in order to create a reasonable schedule – hence I advocate that a heuristic scheduling approach is needed.

To address this set of problems, I present a new heuristic scheduling algorithm – Budget and Deadline Aware Scheduling (BDAS) for scheduling workflows constrained by both budget and deadline. The BDAS algorithm uses a novel tradeoff factor between time and cost to determine the most viable schedule, and uses this to determine the most appropriate type of instance to provision.

6.2 The BDAS Algorithm

In this section, I describe the Budget-Deadline Aware Scheduling (BDAS) algorithm that attempts to meet budget and deadline constraints. The BDAS algorithm is divided into five main phases that are indexed in Table 6.1.

	Phase	Description
6.2.1	Workflow partitioning	The workflow is partitioned into dependency-free bags of tasks, called levels.
6.2.2	Budget Distribution	The user-defined budget (B) is then allocated to each defined level using "All in" strategy.
6.2.3	Deadline Distribution	The user-defined deadline (D) is divided and distributed between levels. Each level gets its own level deadline. All tasks in the same level have the same level-deadline.
6.2.4	Task Selection	A task is selected based on its priority in the ready list for execution.
6.2.5	Instance Selection	In this phase, I introduce new trade-off between Cost and Time. I find the best combination of cost and time in order to select the most suitable instance.

Table 6.1: Five main phases of BDAS

6.2.1 Workflow Partitioning

I aim to maximize task parallelism by partitioning tasks so that there are no dependencies between tasks in each level. Each level can therefore be thought of

as a bag of tasks (BoT) containing a set of independent tasks. This categorizing is in the same way as I described in 5.2.2.

6.2.2 The "All in" Budget Distribution

The financial cost and total execution time of a workflow depends on the number and types of instances requested during resource provisioning. The cost plays a significant role in a cloud environment as users wish to minimize costs and providers maximize profits. Most cloud providers, for example Amazon, charge users for a minimum period of time – even if the instance is only used for a fraction of the period leased.

The idea of budget distribution is to distribute a global budget (B) among different levels and schedule each task on an instance considering the available sub-budget $(subB(\ell))$ assigned to a level. Essentially, the workflow is transformed into internally dependency free "bag of tasks" and then the workflow budget is distributed over these levels. In section 5.2.3, I presented new notions for distributing budget based on the dependency structure inherent in workflows levels. In addition, I proposed several new strategies for sharing or distributing budget over these levels.

The most significant differences in each strategy lie in the calculation of the sub-budget. In some strategies, a Budget Factor (BF) is used that determines a share of the budget for each level. Each sub-budget assigned to a level is termed the level budget. Overall, the strategy that performed the best for all workflows in terms of both success rate and makespan was "All in". The "All in" strategy assigns the budget to the earliest levels and was shown to be beneficial in terms of reducing makespan. An important concept in my algorithm is trickling down unused budget to later levels, and this is expressed by equation 6.1:

$$Spare_B = subB(\ell) - \sum_{t_i \in \mathrm{TLS}(\ell)} TaskCost_{t_i}, \tag{6.1}$$

where Task Level Set (TLS) is defined in equation (4.2). I define Spare Budget as the amount of money remaining after allocating all tasks in the level ℓ. I then

add the leftover to the next level ($\ell + 1$). The "All in" gave the best success rates – and the best performance for data-intensive workflows due to the resulting data locality, from fewer but more powerful instances.

6.2.3 Deadline Distribution

The main idea of deadline distribution is to distribute deadline among different levels and complete its execution before any assigned sub-deadline so that the global deadline can be met. One approach is to divide or distribute the deadline over the workflow as sub-deadlines to ensure a more manageable constraint satisfaction problem, and then to provision the instances to meet these sub-deadlines.

In section 5.3.3.2, I explored a range of strategies for distributing a deadline over a workflow. I found that the choice of distribution strategy has a significant impact on performance. Some strategies exhibit performance that is strongly dependent on the workflow size and structure, including process, pipeline, data distribution, data aggregation and data redistribution. In general, the strategy that takes into consideration the execution time of each level as well as number of tasks in the level yields the lowest cost.

Once all tasks are assigned to their respective levels, the tasks are proportionally distributed across each level based on the user deadline (D). Each sub-deadline assigned to a level is termed the level deadline $(subD(\ell))$. In order to meet the overall deadline, I attempt to ensure that every task in a level can complete its execution before the assigned sub-deadline. Firstly, the initial estimated deadline for each level (ℓ) is calculated as:

$$subD(\ell) = \max_{t_i \in \mathrm{TLS}(\ell)} \{\mathrm{ECT}(t_i)\}, \tag{6.2}$$

where $\mathrm{ECT}(t_i)$ denotes the Earliest Completion Time (ECT) of task t_i over all instances.

In equation (6.2), the maximum ECT of all tasks in a level is used as the overall estimate for that level. This duration is effectively the absolute minimum

time that is required for all tasks in a level to complete execution in parallel.

After calculating the estimated deadline value for all levels, the user deadline is distributed among all tasks non-uniformly based on a deadline proportion and the number of tasks in each level. For this purpose, I first introduce the proportion unit denoted by $\propto_{deadline}$ in equation (6.3):

$$\propto_{deadline} = \sum_{\ell=1}^{t_{entry}} |subD(\ell)| \times |\,\mathrm{TLS}(\ell)|, \tag{6.3}$$

where $|subD(\ell)|$ is the length of each level deadline calculated by the differentiate of level ℓ and $\ell - 1$. Also, $|\,\mathrm{TLS}(\ell)|$ is the number of tasks on each level.

Now the Deadline Factor (DF) is defined as:

$$DF = \frac{D - subD(1)}{\propto_{deadline}}, \tag{6.4}$$

where $subD(1)$ is the level that contains the exit task. The numerator in equation (6.4) is the leftover deadline subtracting the total deadline (D) and the soonest possible finish time for a workflow ($subD(1)$).

I then update the length of each level deadline as a function of this Deadline Factor (DF) to each level as follows:

$$subD(\ell) = DF \times |subD(\ell)| \times |\,\mathrm{TLS}(\ell)| + subD(\ell). \tag{6.5}$$

Intuitively, the levels with longer executing tasks and higher number of tasks gain a larger share of the user deadline.

6.2.4 Task Selection

In BDAS, tasks are executed level by level, which means a task can start execution once all tasks in previous levels have been scheduled. There are no dependencies between tasks that are at the same level. Therefore, all tasks within a level are readily executable and are put to the task ready list. To select a task, at first all tasks in the ready list should be prioritized. Tasks are prioritized based on their Earliest Start Time (EST).

6.2.5 Instance Selection

At the point the algorithms perform instance selection: (i) each task is already assigned a level, (ii) the budget and deadline for each level is already determined, and (iii) the priority of each ready task is already assigned. During instance selection, a trade-off must be made between execution time and cost. To demonstrate this trade-off, I show the expressions for both the time and the cost of executing each task on each instance type in equations (6.6) and (6.7), forming a pair of expressions for Time and Cost.

Firstly, the time needed for the current task, t_i, on the instance p_j is denoted by $\mathrm{ECT}\,(t_i|p_j)$. The ECT is the earliest time that a task can finish on an instance, which is defined earlier in equation (4.4). Using this observation, I can then compute how much the estimated level deadline of the current task differs from the earliest completion time of task on the instance p_j:

$$\mathrm{Time}\ {}^{p_j}_{t_i} = \frac{subD^{\ell_{t_i}} - \mathrm{ECT}\,(t_i|p_j)}{subD^{\ell_{t_i}} - \mathrm{ECT}\,(min)}. \tag{6.6}$$

In equation (6.6), $subD^{\ell_{t_i}}$ is the deadline that is assigned to the level that contains the task t_i. Also, $\mathrm{ECT}\,(min)$ is the minimum execution time among all instances that keeps the current task on schedule.

The values of Time for task t_i are related to instance types, wherein a lower value of Time means running on a cheaper instance. The reason is that the values of $\mathrm{ECT}\,(t_i|p_j)$ is bigger on an instance with a lower processing capacity. Also, if the value of Time is negative, it means that the current task on the selected instance will exceed the level deadline i.e. $\mathrm{ECT}\,(t_i, p_j) > Level^{\ell_{t_i}}_{deadline}$.

$TaskCost^{p_j}_{t_i}$ refers to the cost of scheduling the current task t_i on instance p_j. The cost of executing task t_i among all instances p_j is defined as:

$$\mathrm{Cost}\ {}^{p_j}_{t_i} = \frac{subB^{\ell_{t_i}} - TaskCost^{p_j}_{t_i}}{subB^{\ell_{t_i}} - TaskCost_{min}}, \tag{6.7}$$

where $subB^{\ell_{t_i}}$ denotes the budget assigned to the level which contains the current task. The best cost (minimum cost) of executing the task t_i among all instances is $TaskCost_{min}$.

When an instance is first provisioned, the instance is billed on an hourly interval until it is terminated. Therefore, the first task assigned to an instance in a particular billing interval incurs the entire cost of that interval. As a consequence, if other tasks can be executed during that paid interval, then there is no additional execution cost for executing them. Ideally, during instance selection, the reuse of such instances leads to lowering the execution cost. This situation comes true when Cost in equation (6.7) is 1 (The value of $TaskCost_{t_i}^{p_j}$ and $TaskCost_{min}$ are zero).

To find the best instance, a Cost Time Trade-off Factor (CTTF) is used in equation (6.8) that considers a trade-off between cost and time. The CTTF value is trying to adjust the cost and time for current task among all instances. For cost minimization purposes, most of the proposed algorithms like [23] schedule tasks on the cheapest available instance (slower) while still meeting its assigned sub-deadline. However, with this strategy, tasks can take much longer time if the resources are slower and leads to some delay in the EST of its children. To avoid this hold up, more expensive instances are better choices to meet deadline, which probably lead to exceeding the budget. Therefore, the key concept of introducing CTTF in (6.8) is to make a trade-off between time and cost.

While calculating (6.7) and (6.6) for a task among all instances, four different combinations of cost and time ranges that are possible are listed in Table 6.2. As I mentioned before, a trade-off between Cost and Time should be considered in order to find the best instance for each task. The trade-off function is calculated as follows:

$$\mathrm{CTTF}_{t_i}^{p_j} = \mathrm{Cost}\ {}_{t_i}^{p_j} + \mathrm{Time}\ {}_{t_i}^{p_j}. \tag{6.8}$$

The instance with highest CTTF value is selected for execution the task t_i.

6.3 Evaluation

This section presents the performance evaluation of the BDAS algorithm, with recent published papers in [65] and [92] that match my goal and conditions.

Cost	Time	Description
> 0	> 0	In the best scenario, there is enough budget from a user to schedule a workflow. Also, the deadline is relaxed and there is no pressure on finishing the scheduling. In this situations, while distributing budget and deadline, each level can get a big share of them. Therefore, both values of Cost and Time are positive as task level's sub-budget and task level's sub-deadline are big enough. Moreover, at the beginning of the scheduling process when no budget has been spent or there is enough time for scheduling the current task t_i on any instances, both values are positive.
≤ 0	> 0	The total assigned budget for the level ℓ has already been spent ($subB^{\ell_{t_i}}$=0). Therefore, a new instance cannot be launched as there is no budget left. Another probable situation is that the cost of current task t_i on the instance p_j is higher than the left-over sub-Budget (in the numerator of equation (6.7)). If either of these conditions is true, equation (6.7) is less than or equal zero.
> 0	≤ 0	In tight deadlines, there is a possibility that cheaper instances cannot meet the task level's sub-deadline. Therefore, the value of ECT in equation (6.6) is greater or equal to task level's sub-deadline. If this condition is met, the value of equation (6.6) becomes negative or zero.
≤ 0	≤ 0	In tight budget and deadline, where current sub-budget and sub-deadline are smaller than values of $TaskCost$ and ECT, respectively, for task t_i on an instance, both equations (6.7) and (6.6) can be negative or zero.

Table 6.2: Different possible ranges for Cost and Time.

The CloudSim [85] simulator was used to implement and compare the performance of all four algorithms. The simulation scenario was configured as a single data-center and six different instance types. The pricing model and other characteristics are explained in section 4.3.1.

In order to evaluate the performance of the algorithms with a realistic load, five common scientific workflows were used: Cybershake, Epigenomics, Montage, LIGO and SIPHT. The characteristics and task composition of these workflows have been analyzed in published works cited in the related work section [21, 75]. To evaluate the performance of these algorithms, different deadlines were chosen from tight to relaxed. For this purpose, the fastest schedule (denoted by FS) is calculated as a baseline schedule. Effectively, this baseline is the fastest possible execution – ignoring costs. The fastest schedule is expressed as:

$$FS = \sum_{t_i \in CP} (w_i^j), \tag{6.9}$$

where w_i^j is the computation cost of task t_i on the fastest instance p_j. Additionally, different budget ranges were considered for the scientific datasets from lowest possible through to sufficient. As a baseline schedule, the lowest budget (denoted by LB) to schedule a workflow is given by equation (6.10):

$$LB = \sum_{\forall t_i \in G} TaskCost_{t_i}^{p_j}, \tag{6.10}$$

where p_j is the cheapest instance. To achieve this, all tasks are executed on the instance with the lowest cost (the cheapest instance). This assignment gives us the lowest possible cost required for executing a workflow, irrespective of finishing time.

Using equations (6.9) and (6.10), a variation for budget and deadline are calculated from tight to moderate to relaxed:

$$\text{deadline} = \alpha * FS, \quad 0 < \alpha < 20, \tag{6.11}$$

$$\text{budget} = \beta * LB, \quad 0 < \beta < 20. \tag{6.12}$$

Both deadline factor α and budget factor β start from 1 to consider very tight values and are increased by one up to a value of 20, which results in a very relaxed deadline and sufficient budget.

6.3.1 Performance Metrics

In order to evaluate the algorithms under test, the following performance metrics were selected: Success Rate (SR), Cost Ratio (CR) and Time Ratio (TR).

- Success Rate (SR): Success rate of each algorithm (SR), calculated as the ratio between the number of simulation runs that successfully met the scheduling of both constraints, deadline and budget, and the total number of simulation runs (denoted by n_{Tot}), defined as:

$$SR = \frac{n\,(k)}{n_{Tot}}, \tag{6.13}$$

  where $n(k)$ is the cardinality of the set k and $n_{Tot} = 100$.

- Cost Ratio (CR): In order to compare the achieved cost between each algorithm, a Cost Ratio is used, which is calculated by dividing overall cost (expressed in equation 3.9) by given budget (B).

$$CR = \frac{Cost_o}{B}. \tag{6.14}$$

  A CR value greater than 1 indicates a cost larger than the budget, which counts as a failure to meet the defined budget. CR value of less than 1 indicates that the scheduled workflow meets the budget.

- Time Ratio (TR): With the same reasoning with CR, the value of Time Ratio metric is considered while smaller than 1 means a user-defined deadline

(D) is met.

$$TR = \frac{Makespan}{D}. \tag{6.15}$$

6.4 Experimental Results

I defined 20 different deadline factors (given by (6.11)) and 20 different budget factors (given by (6.12)). Therefore, for my experiments permuting both factors yields 400 different cases per workflow and a grand total of 40000 test cases for all workflows. Comparative studies were conducted by box plots for cost ratio and time ratio defined in (6.14) and (6.15) respectively. A box plot shows the degree of dispersion of cost ratio and time ratio. The lowest "dash" denotes the minimum, while the upper "dash" denotes the maximum. The median is marked in the box, and the top boundary and bottom boundary denote the upper and lower quartile.

I present the results in two dimensions:

(i) Time Efficiency (Budget Factor vs. Time Ratio): Four different ranges of budget factor were chosen in the experiments giving us more detail about the performance of all algorithms in order to simplify interpretation of results. Then, for each budget factor a box plot is drawn for all achieved makespan by Time Ratio (6.15) factor. With this plot I can observe for a constant budget factor, for example $\beta = 4$, the behavior of each algorithm on meeting/failing and utilizing deadlines. Moreover, corresponding success rates of each budget factor are plotted.

(ii) Cost Efficiency (Deadline Factor vs. Cost Ratio): Four different ranges of deadline factor were chosen, including very tight deadlines ($\alpha = 4$), ending with more relaxed deadline ($\alpha = 16$). The y-axis in this plot is Cost ratio (6.14) indicating meeting/failing and utilizing all ranges of budget for a constant deadline. Also, corresponding success rates of each deadline factor are plotted.

The primary object for any workflow scheduling algorithm is finding a schedule without violating defined constraints. For a user, shortening the execution time with minimum cost is of interest. Therefore, precise behavior of each algorithm on expenditure budget/deadline is of interest. In order to present descriptive statistics, box plots are used, which are divided into two areas named valid schedule and invalid schedule. The valid schedule starts from 0 to 1, indicating an algorithm could meet the define constraint. Therefore, time/cost efficiency (valid schedule) for a given budget/deadline factor not only shows if a schedule without violation is found but also represents the distribution of each constraint.

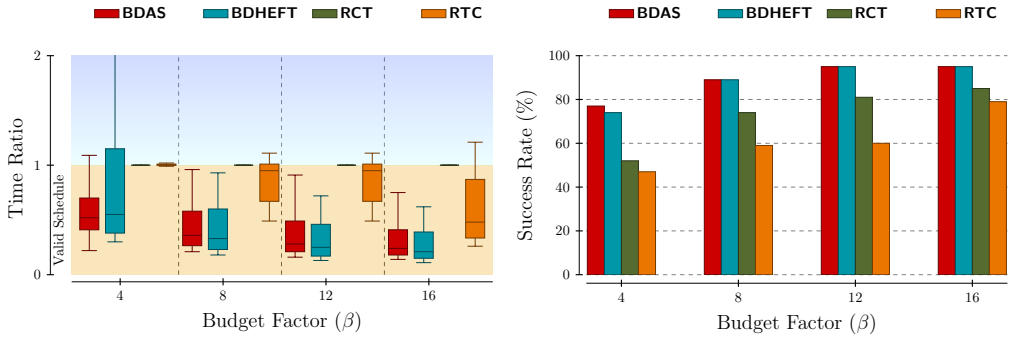
6.4.1 CYBERSHAKE

Figure 6.1 shows the results achieved for CYBERSHAKE. For time efficiency (Figure 6.1(a)), BDAS and BDHEFT have the best performance by achieving a smaller time ratio than others. This means that both algorithms were able to find a schedule ahead of the given deadlines. Moreover, the related success rate shows BDAS and BDHEFT outperform both RCT and RTC.

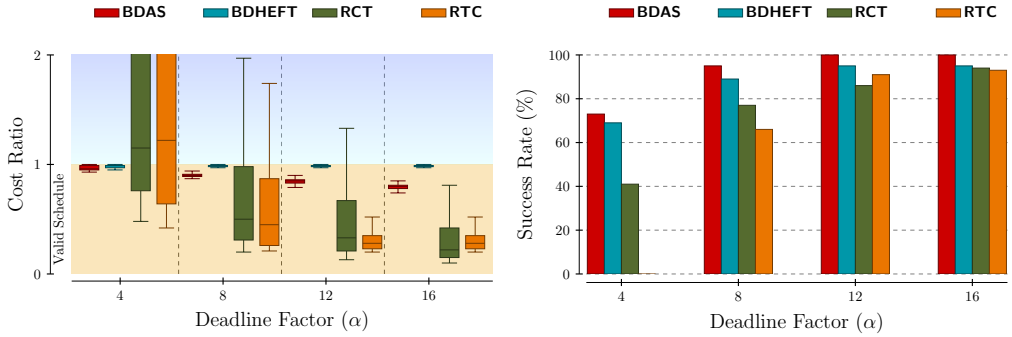
However, in terms of Cost Efficiency, both BDAS and BDHEFT utilized up to 100% of the assigned budget in CYBERSHAKE workflow (Figure 6.1(b)). The RCT and RTC algorithms are better at saving cost compared to BDAS and BDHEFT. The RTC algorithm has the worst success rate among the four, especially in tight deadlines. Indeed, RTC experienced a 100% failure when deadline factor, α, is 4.

6.4.2 EPIGENOMICS

The results obtained for the EPIGENOMICS are shown in Figure 6.2. The BDAS algorithm is capable of meeting constraints of nearly 100%. It almost uses the whole budget while finding a schedule sooner than defined deadline. The BDAS for EPIGENOMICS utilizes the whole budget and finishes sooner than defined deadline leading to users' satisfaction. The behavior of RCT and RTC



(a) Time Efficiency



(b) Cost Efficiency

Figure 6.1: CYBERSHAKE

are opposite of the BDAS algorithm whereby they utilize deadline rather than budget. An interesting feature of RCT in Figure 6.2(a) happens when budget factor, β, is 4. As can be seen, almost all time ratio results yield a valid schedule, while achieving a success rate of 60%. It means 40% of failures in this algorithm happens due to violating budget. The opposite is observed in BDHEFT in 6.2(b), when deadline factor ,α , is 8. Nearly all cost ratio values are inside the valid schedule; however, the success rate is less than 40%. The reason is that most of the failures are caused by exceeding the given deadline.

The highest failure occurs in EPIGENOMICS while deadline factor α is 4, BHEFT,RCT and RTC can find a schedule for less than 7% before they're constraints are reached. However, the BDAS algorithm obtained more than 90% success rate under the same conditions.

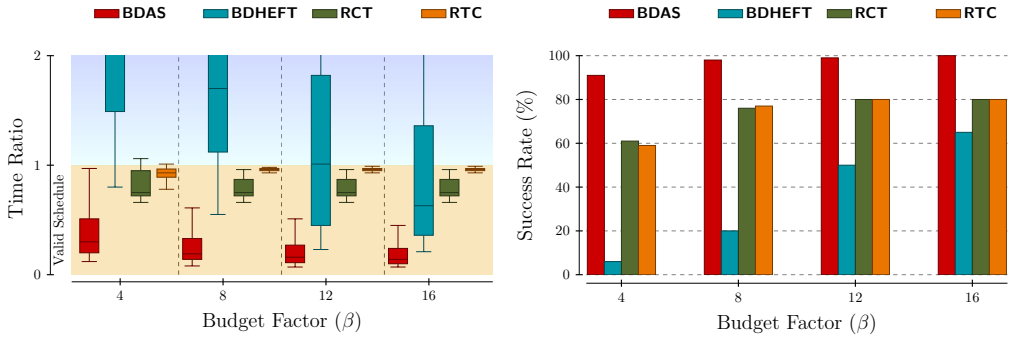
6.4.3 LIGO

Figure 6.3 shows the results achieved for LIGO. In general, the BDAS heuristic has the best performance in terms of meeting given constraints, budget and deadline. The behavior of BDAS is similar to EPIGENOMICS, whereby it utilizes the full budget and completes the execution of workflows sooner than prescribed deadline. On the contrary, the RCT and RTC algorithm spend less budget while exhausting the deadline.

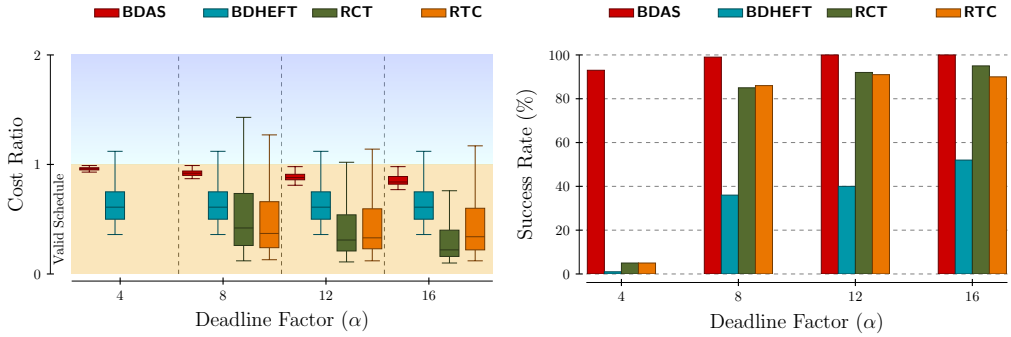
The worst performance is observed with BDHEFT in the first and second defined deadline factor, α, with almost 100% failure. All related cost ratio values for mentioned deadline factors are placed in the valid schedule, as can be seen in 6.3(b). All failures are due to violating the prescribed deadline.

6.4.4 MONTAGE

The results obtained from the simulations for the MONTAGE workflow are shown in Figure 6.4. All algorithms exhibit nearly identical success rate for this application in specified intervals. The figures for BDAS in both plots show similar decrease trends in time ratio and cost ratio. In terms of time efficiency

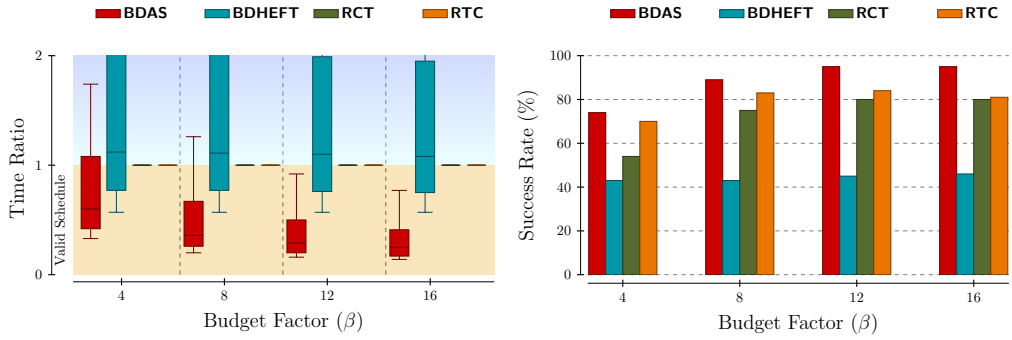


(a) Time Efficiency

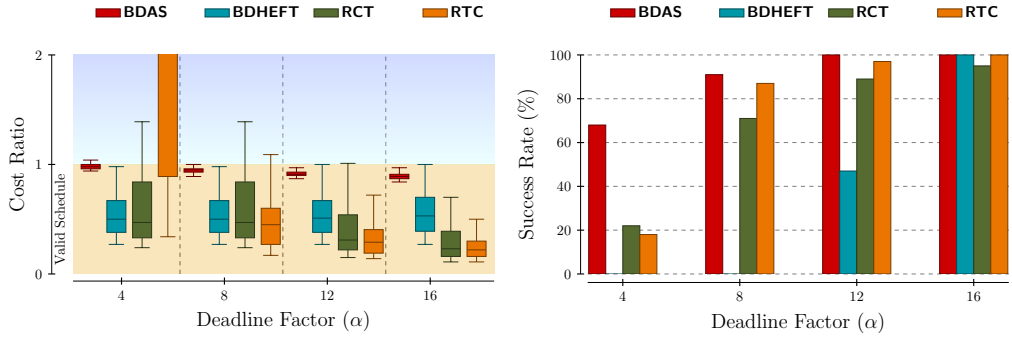


(b) Cost Efficiency

Figure 6.2: EPIGENOMICS



(a) Time Efficiency

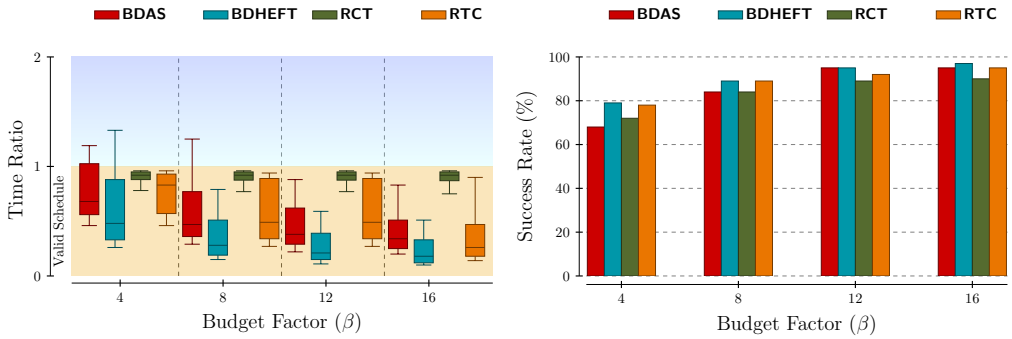


(b) Cost Efficiency

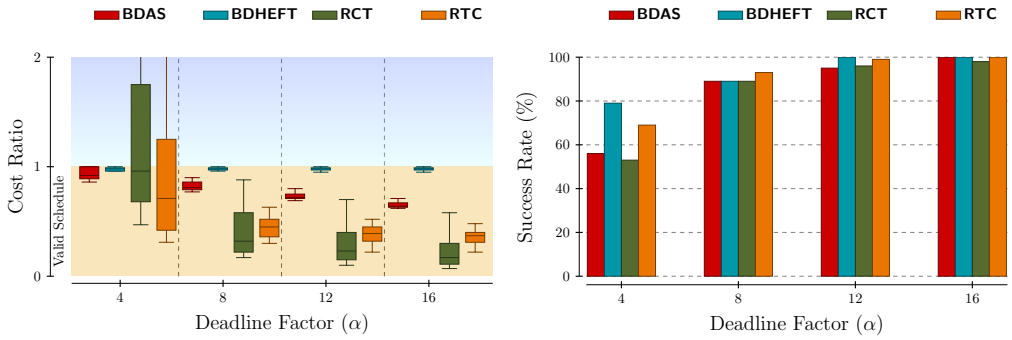
Figure 6.3: LIGO

Figure 6.4(a), the time ratio obtained by BDHEFT is smaller than the ratio obtained by other algorithms in all intervals. The behavior of BDHEFT for different deadline intervals implies that it maintains a constant level of performance by utilizing all the allocated budget.

Apart from the lowest considered budget factor and deadline factor in my experiments, all four algorithms obtained ratios remained under 1 for MONTAGE (valid schedule). In MONTAGE, all algorithms gained the highest total success rate compared to other scientific workflows, as discussed in Section 6.4.6.



(a) Time Efficiency



(b) Cost Efficiency

Figure 6.4: MONTAGE

6.4.5 SIPHT

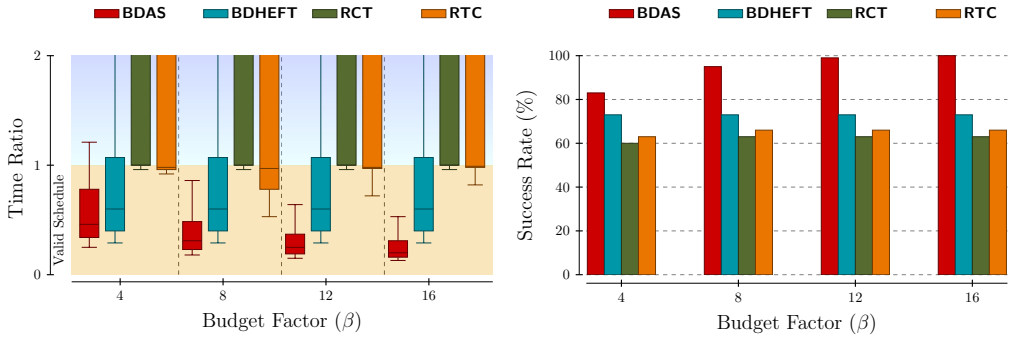
The time ratio and cost ratio obtained by tested algorithms for SIPHT workflows are shown in Figure 6.5(a) and Figure 6.5(b), respectively.

The maximum of box plots for RCT and RTC are not shown because any ratio greater than 1 means a failure to yield a valid schedule, which is not of interest. Therefore, I only show values up to 2 to better illustrate results with valid schedules in more detail. The results are similar for RTC and RTC when almost 100% failure is observable in the tight deadline in 6.5(b). As no plots are shown in this figure, it means the achieved cost by RCT and RTC is at least 2 times greater than defined budget.

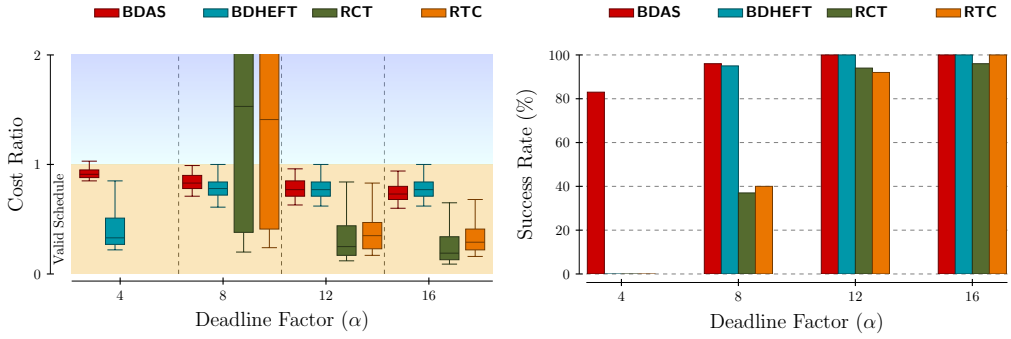
The impact of BDHEFT is a complete contrast while achieving 100% failure due to time violation, which can be supported by looking at 6.5(b). Its clearly obvious that when α is 4, all cost ratios are placed in the valid schedule. However, the corresponding success rate reports almost 100% failure. Therefore, it can be concluded that failures happened due to exceeding deadlines.

6.4.6 Total Success Rate

Table 6.3 shows the total Success Rate of each algorithm, considering all values for budget factor, β, and deadline factor, α. For each workflow, I test 400 different states, which is combination of 20 different budget factors and 20 different deadline factors. I have considered 100 different workflows that leads to 40000 different test cases for each scientific workflow. A low success rate indicates that the algorithm cannot find a schedule that meets the budget and deadline. The best overall performer is BDAS, which exhibits a success of above 84% in all datasets with the best performance of 96% in Epigenomics. The highest failure and worst performance occurs in BDHEFT algorithm, which fails around 60% of different test cases in EPIGENOMICS and LIGO. RCT and RTC present almost the same results with the highest success rates with 82% and 87% in MONTAGE, respectively. It is worth noticing that MONTAGE workflow is the most successful application to be scheduled with all algorithms, based on re-



(a) Time Efficiency



(b) Cost Efficiency

Figure 6.5: SIPHT

	BDAS	BDHEFT	RCT	RTC
CYBERSHAKE	88%	84%	72%	64%
EPIGENOMICS	96%	38%	72%	71%
LIGO	86%	44%	69%	78%
MONTAGE	84%	90%	82%	87%
SIPHT	93%	72%	59%	62%
Mean	89.4%	65.6%	70.8%	72.4%

Table 6.3: Total Success Rate for five different scientific workflows. The mean Total Success Rate for BDAS is 17.0%–23.8% higher than other algorithms.

ported success rate.

In general, my approach is more consistent to generate acceptable schedules under defined constraints.

6.4.7 A summary of the performance of scheduling algorithms

In order to have a better insight on the performance of the proposed BDAS algorithm, a new metric is used to measure the behavior of BDAS based on budget and deadline. The time-cost relationship shows a more accurate representation of BDAS on spending defined budget and deadline. This relationship is defined as:

$$f = \sum_{i \in \alpha} \sum_{j \in \beta} \frac{T(i,j)}{C(i,j)}, \tag{6.16}$$

where T is defined as:

$$T = \frac{makeSpan}{Deadline} \tag{6.17}$$

and C is defined as:

$$C = \frac{AchievedCost}{Budget} \tag{6.18}$$

	BDAS	BDHEFT	RCT	RTC
CYBERSHAKE	0.4	0.75	2.7	1.88
EPIGENOMICS	0.35	4.06	2.66	3.06
LIGO	3.09	3.32	3.26	3.45
MONTAGE	0.71	0.54	3.81	1.19
SIPHT	0.72	2.24	4.69	3.28

Table 6.4: Time-Cost relationship for five different scientific workflows.

Deadline factor, α is defined in (6.11), and budget factor, β, is defined in (6.12). The results for five scientific workflows are given in Table 6.4 based on 20 different deadline and budget factors.

In general, the behaviour of scheduling algorithms in this chapter can be summarized as follows:

- In general, BDAS utilized budget without considerable failures.
- The BDAS outperformed all algorithms in terms of success rate in specified tight budget/deadline factors.
- In contrast to BDAS, the performance analysis shows the both RCT and RTC utilize deadline while trying to find a cheaper schedule. In tight deadlines, most of the failures in RCT and RTC happen due to violating the budget constraint. This is the case in 6.1(b), 6.2(b) and 6.5(b).
- BDHEFT performs worse than other algorithms. It not only suffers from a large number of failures and constraint violation; I also see the wide range on its cost and time ratios.
- Most of the failures in BDHEFT are due to violating deadline. For example, in SIPHT, when deadline factor is 4, the cost ratio obtained by BDHEFT is placed in the valid schedule. However, it achieved almost 100%

failure due to violating the prescribed deadline. The same pattern is observable in EPIGENOMICS workflow.

6.4.8 Sensitivity Analysis

The design of a scheduling strategy depends on what objectives are more important to users. In order to check how sensitive an algorithm is, in terms of assigning priority to budget and deadline, a sensitivity analysis is performed in this thesis. Sensitivity analysis may be carried out to evaluate the stability and robustness of scheduling algorithms, allowing users to consider the preferred choice.

To achieve customizable services based on user requirements, I consider a weight parameter in my algorithm. With this mechanism, the sensitivity experiment consists of analyzing how different priorities on cost and time would affect my algorithm. The parameter ω takes values between 0 and 1; the smaller the value the more priority on time, while values close to 1 correspond to assigning more priority on cost. The weight modifies the CTTF in 6.8 as follows:

$$\mathrm{CTTF}_{t_i}^{p_j} = (\omega) * \mathrm{Cost}\ {}_{t_i}^{p_j} + (1-\omega) * \mathrm{Time}\ {}_{t_i}^{p_j}. \tag{6.19}$$

The results in Figure 6.6 are shown up to their maximum possible number of success rates in total, no matter what the budget factor and deadline factor are. The results of this set of experiments suggest that the structure and type of workflows can significantly impact on the user priorities in scheduling workflows. The analysis provides insights into the performance of my scheduling algorithm with data intensive vs. compute intensive workflows, of which the most important ones are:

- Data-Intensive: Cybershake, LIGO and MONTAGE, which are three data-intensive workflows that deal with large amounts of data in the range of megabytes to petabytes while spending most of their time performing I/O activities. As reported in [21], a MONTAGE workflow with 10000 tasks reads 146 GB of input data, needs 4.93 CPU hours, and writes 49 GB

of output data. In these data-intensive applications, data transfer time between resources usually takes a significant portion of the execution time. Therefore, the most effective approach in data-intensive workflows is reducing in data movement. Assigning low priority to Cost in the trade-off equation introduced in 6.8 usually leads to plenty of failure. When the weight parameter, ω, is 0.1, almost 80% of experiments in Cybershake fail to meet the defined constraints. In the same condition, LIGO and MONTAGE experienced 60% and 55% failures, respectively.

In order to have a better insight, I look to the Time and Cost equations given earlier in 6.6 and 6.7, respectively. The value for Time is large for an instance that can finish its execution sooner, which is probably an expensive instance. Because the expression in equation 6.19 has a higher weight on Time, more expensive resource are selected. This explanation is confirmed by the results while all the failures in the tested data-intensive workflows happened due to exceeding budget.

Another interesting observation in data-intensive applications is the contrary situation, when the highest priority is assigned to Cost (ω=0.9). Highest failure (60%) happened in MONTAGE while CYBERSHAKE experienced 40% failures. Before analysis, I need to study the interaction between Time and Cost in this scenario. The highest value for Cost in 6.7 is attained when a task is executed during paid intervals that have no cost. Because the Cost factor in equation 6.19 has higher weight, which directly affects the selection of the most suitable instance, an instance with zero cost usually has a higher chance of being selected. However, it may fail to meet a task's sub-deadline. At the end, continuing the same pattern leads a failure by exceeding the deadline.

- Compute-Intensive: Large-scale applications mainly consisting of complex tasks requiring high performance computing instances are called compute intensive workflows. In these applications, the majority of task's execution time deals with computation rather than I/O. The EPIGENOMICS

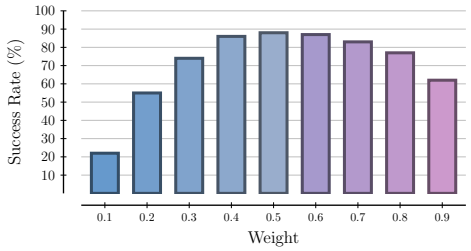
workflow can be considered as CPU intensive application due to spending almost 99% of its execution in the CPU, and performs comparatively low I/O compared to Montage [21]. Higher priority given to time rather than cost caused nearly 50% failure in SIPHT and 40% failure in EPIGENOMICS. This is a result of more expensive resources being more likely to be selected, resulting in exceed budget. The experimental results in EPIGENOMICS show that the likelihood of meeting constraints is enhanced by increasing the value ω. As can be seen in Figure 6.6, the general trend is that by assigning higher priority on cost, a scheduler yields fewer failures. As I mentioned earlier, compute-intensive workflows allocate most of their time to performing computation. Therefor, a budget is more important factor leading launch more instances.

In summary, the results of this set of experiments suggest that the different priorities have different impacts in finding a valid schedule for each workflow type.

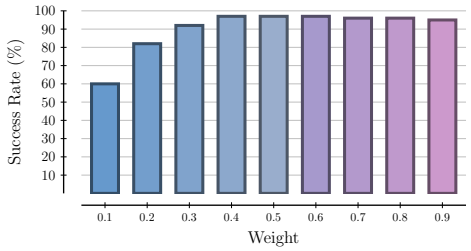
6.4.9 Decreasing billing cycle

In this section we investigate the Time Ratio and Cost Ratio performance of the four scheduling algorithms across different billing cycle (from 15 minutes to 1 hour with 15 minute increments). For the Time Ratio evaluations, we use a budget factor of $\beta = 4$ and $\beta = 16$, while Cost Ratio evaluations are performed with deadline factors fixed at $\alpha = 4$ and $\alpha = 16$.

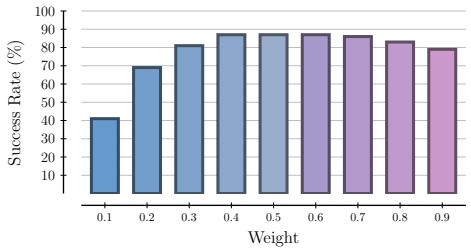
Figures 6.7–6.11 show the Time Ratio and Cost Ratio performance for CYBERSHAKE, LIGO, MONTAGE and SIPHT respectively. For BDAS and BDHEFT, the medians for Time Ratio marginally decrease with decreasing billing cycle while the range for Time Ratio also shrinks with decreasing billing cycle (from 60 minutes to 15 minutes). This observation for BDAS Time ratio holds across all four workflows for both budget factors $\beta = 4$ and $\beta = 16$. By decreasing the billing cycle, the number of idle time slots between scheduled tasks are decreased leaving more budget available for launching new VMs, thus resulting



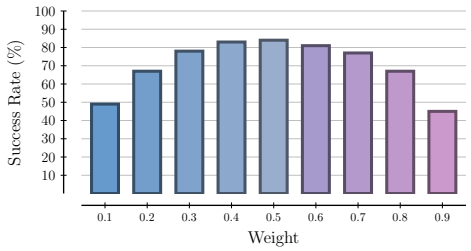
(a) CYBERSHAKE



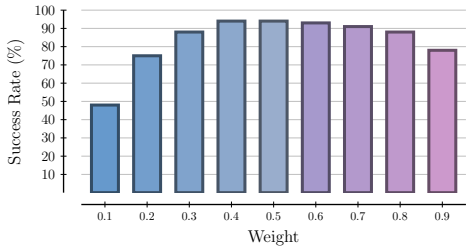
(b) EPIGENOMICS



(c) LIGO



(d) MONTAGE



(e) SIPHT

Figure 6.6: Sensitivity Analysis for five different data set.

in a reduction of Time Ratio.

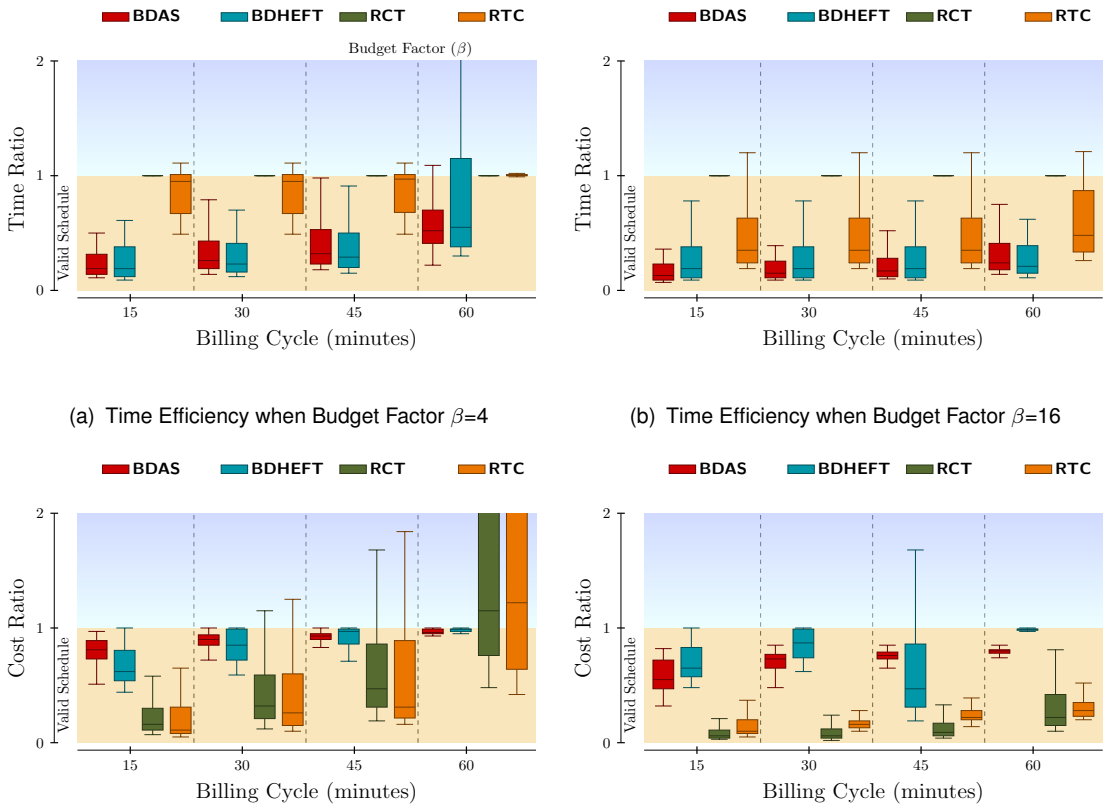
The Cost Ratio medians for BDAS and BDHEFT also decreases with decreasing billing cycle, however, the range for Cost Ratio slightly increases (converse of Time Ratio) suggesting that shorter billing cycles introduce more cost volatility. A larger range for Cost Ratio is observed in the CYBERSHAKE Fig. 6.7 and EPIGENOMICS Fig. 6.8, and to a lesser extent in LIGO Fig. 6.9, MONTAGE Fig. 6.10 and SIPHT Fig. 6.11. A shorter billing cycle means more granular updates on incurred cost resulting in fewer idle instances, this helps reduce cost (as seen in the median trend). However, the more granular updates also increase the Cost ratio variance as observed in Figures 6.7–6.11. Both RTC and RCT do not show significant variation in Time Ratio and Cost Ratio performance with decreasing billing cycle.

Overall, the different billing cycles have marginal influence of the Cost Ratio and Time Ratio; this finding concurs with another research by Rodrigues and Buyya [93].

6.5 Summary

This chapter presented a novel algorithm to address the problem of scientific workflow scheduling in dynamically provisioned commercial cloud environments. This approach focuses on addressing the unique characteristics of workflow execution on cloud platforms, such as on-demand provisioning and instance heterogeneity while simultaneously meeting budget and deadline constraints.

The time complexity of BDAS is of the order $\mathcal{O}\left(n^2p\right)$ which is calculated similar in principle to section 4.2.5. I evaluated the BDAS algorithm and compared it with three previously published algorithms (BDHEFT, RCT, RTC), using a range of metrics. In terms of success rate, the best overall performer is the BDAS algorithm, which exhibits a success of above 84% in all datasets while the worst performance occurs in BDHEFT algorithm, which fails around 60% of different test cases in EPIGENOMICS and LIGO. I also investigated the sensitivity of my



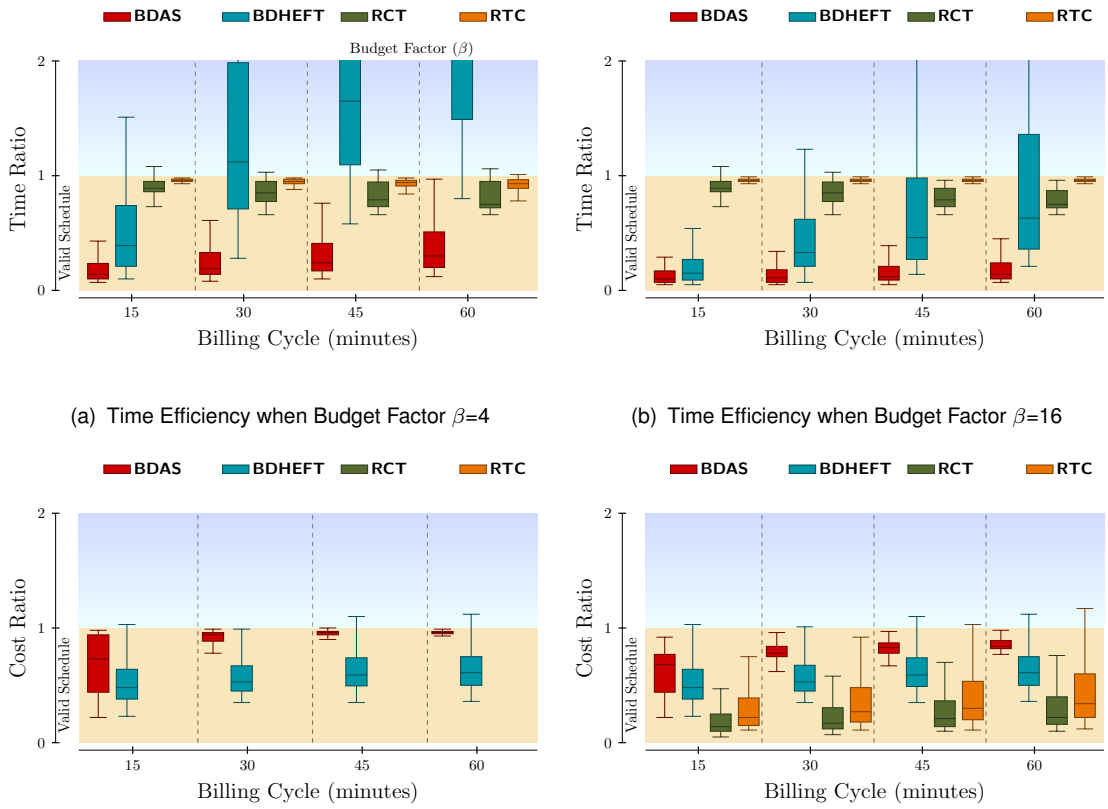
(a) Time Efficiency when Budget Factor β=4

(b) Time Efficiency when Budget Factor β=16

(c) Cost Efficiency when Deadline Factor α=4

(d) Cost Efficiency when Deadline Factor α=16

Figure 6.7: CYBERSHAKE



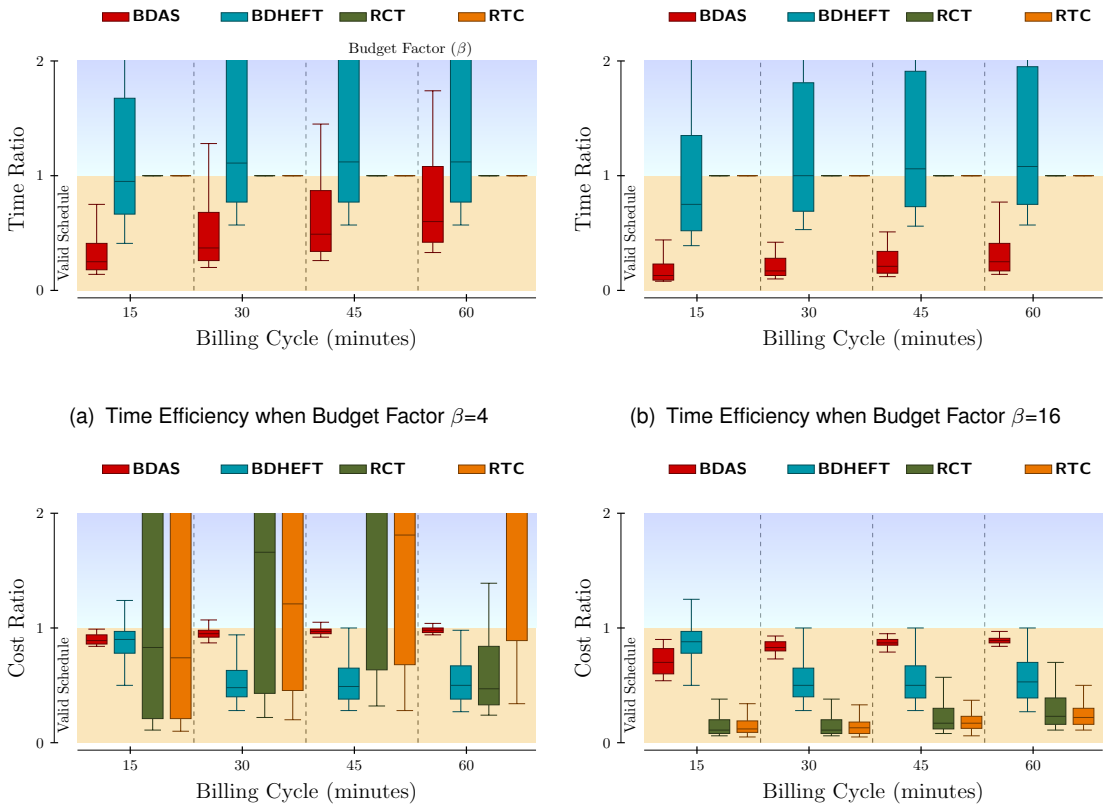
(a) Time Efficiency when Budget Factor β=4

(b) Time Efficiency when Budget Factor β=16

(c) Cost Efficiency when Deadline Factor α=4

(d) Cost Efficiency when Deadline Factor α=16

Figure 6.8: EPIGENOMICS



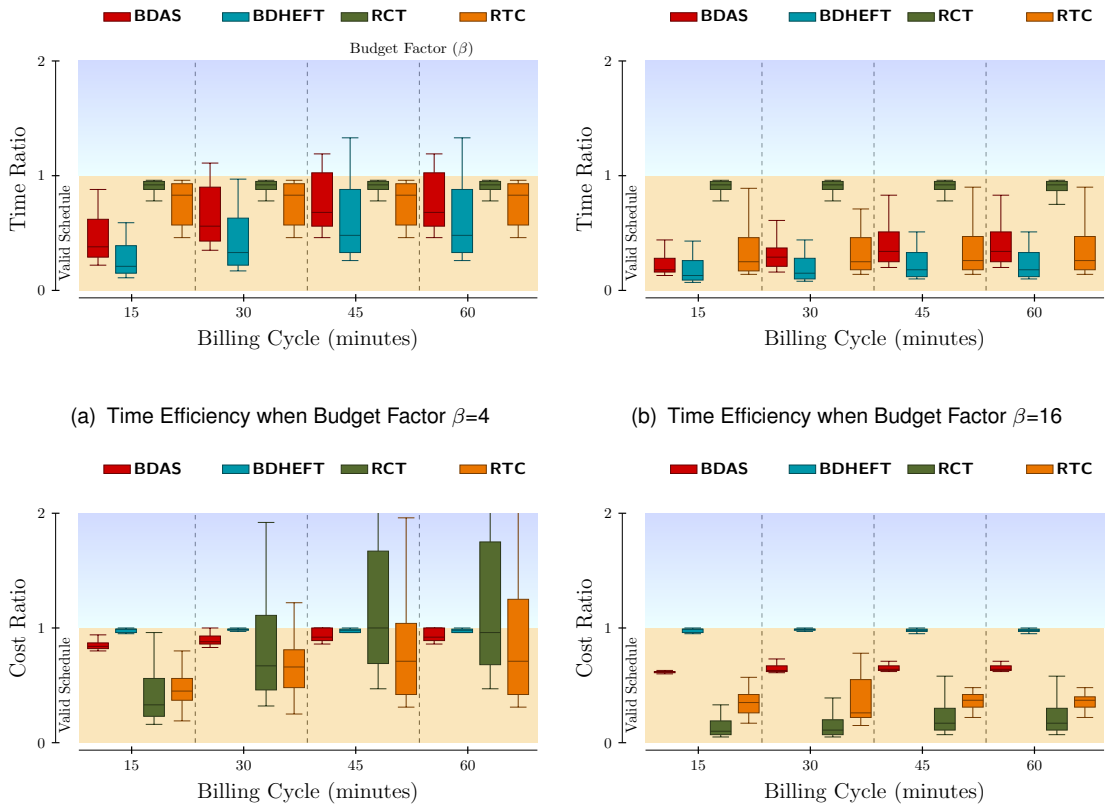
(a) Time Efficiency when Budget Factor β=4

(b) Time Efficiency when Budget Factor β=16

(c) Cost Efficiency when Deadline Factor α=4

(d) Cost Efficiency when Deadline Factor α=16

Figure 6.9: LIGO



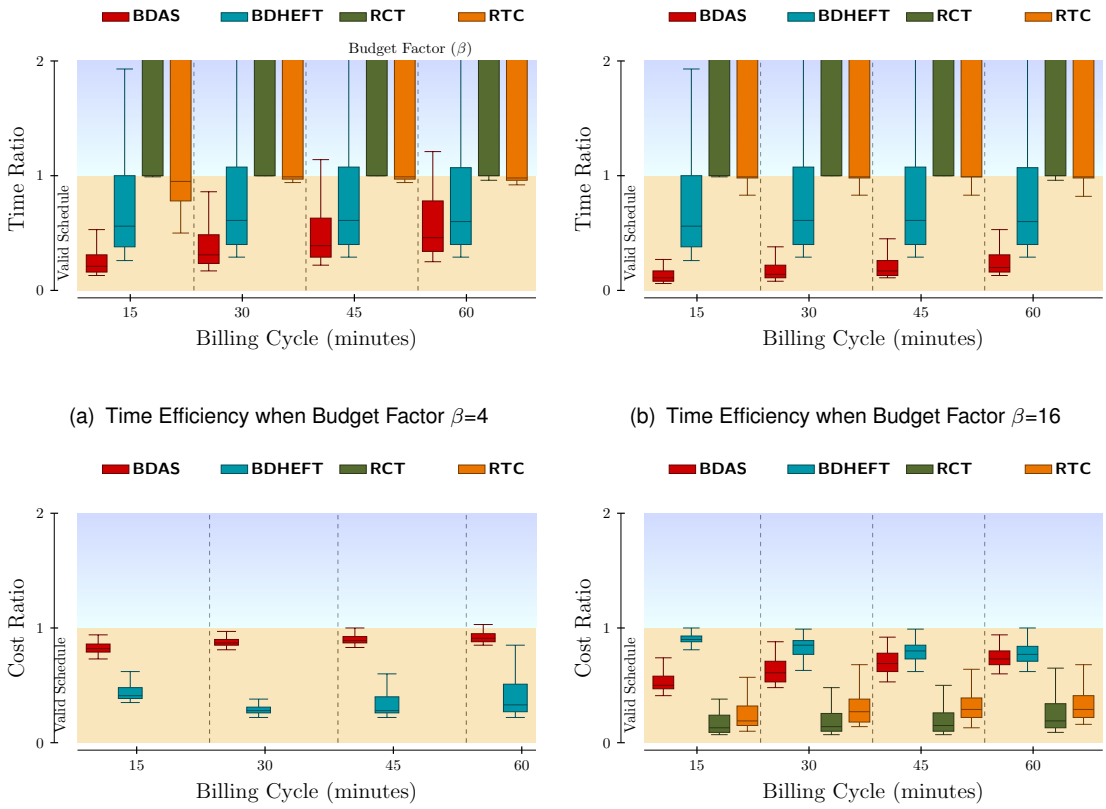
(a) Time Efficiency when Budget Factor β=4

(b) Time Efficiency when Budget Factor β=16

(c) Cost Efficiency when Deadline Factor α=4

(d) Cost Efficiency when Deadline Factor α=16

Figure 6.10: MONTAGE



(a) Time Efficiency when Budget Factor β=4

(b) Time Efficiency when Budget Factor β=16

(c) Cost Efficiency when Deadline Factor α=4

(d) Cost Efficiency when Deadline Factor α=16

Figure 6.11: SIPHT

algorithm on different workflow types over a range priorities for budget and deadline. Increasing budget on compute-intensive workflows tends to achieve higher success rate, which implies that budget is more important than deadline for such applications.

A primary advantage of BDAS (and by extension, heuristic algorithms) is its low overhead, or computational tractability. The scheduling performance achieved by the BDAS algorithm is promising, especially in light of the time required to compute a schedule. However, the presented algorithms in previous chapters belong to SPDP category (see section 2.4), when the number of workflows are known in advance and all are submitted at the same time. In practice, a scheduler may have to schedule an unpredictable stream of workflows. As a result, in the next chapter, I intend to explore the use of a heuristic algorithm in near-realtime, multiple workflow and dynamic scheduling situations.

Chapter 7

Dynamic Workflow Scheduling

7.1 Introduction

Once a scientific experiment is defined in a workflow, it needs to be executed, which requires resources to be requested and provisioned. Such computational resources are unlikely to be dedicated to the execution of a single workflow, and as such, multiple overlapping workflows, which occur at various non periodic times, need to be scheduled and resourced.

As a motivating scenario, consider the example from [94] in which the authors describe data-driven workflows that are executed on-demand, in response to data produced by beam-line experiments. These data need to be processed in near real-time and may occur at undefined intervals. The authors state "The ability to analyze data produced by detectors in near-real-time can enable optimized data collection schemes, on-the-fly adjustments to experimental parameters, early detection of and response to errors (saving both beam time and scientist time), and ultimately improved productivity…". It is clear that in experiments involving physical sciences or real world measurements, scheduling such workloads with static schedulers that require all execution characteristics and all workloads to be known in advance is untenable.

Despite considerable research on cloud scheduling, cost challenges and execution performance of workloads are still significant and unsolved issues. In

fact, scheduling workflows and provisioning cloud resources naïvely can have a significant financial penalty – especially in dynamic markets such as the Amazon spot market [11]. Indeed, the most attractive elements of the cloud itself, which are its elasticity, heterogeneity, and on-demand cost model, further compound the already *NP*-hard [12] workflow scheduling problem. Unfortunately for the actualization of science as outlined above, the most common simplifying assumption for the vast majority of prior workflow scheduling research is that the workload is static and all characteristics and parameters are known and accurate *a-priori*.

This chapter addresses this problem by focusing on dynamic scheduling and presents a novel heuristic algorithm, the Dynamic Workflow Scheduler (DWS), for dynamically scheduling workflows on the cloud from an incoming workload of non-periodic workflows. The presented algorithm schedules workflows with a mixture of different deadline runtime constraints – providing a solution to the Workflow as a Service (WaaS) scheduling problem [72].

7.2 System Architecture

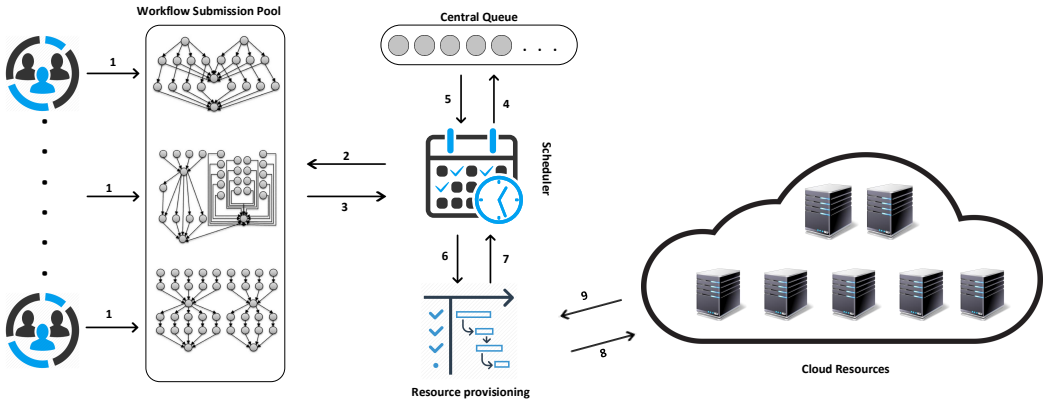


Figure 7.1: Architecture of presented system

The high level architecture of presented Workflow as a Service (WaaS) in this

chapter is shown in Figure 7.1 – and consists of a submission pool, scheduler, resource provisioning unit and central queue.

Users submit workflows at different times (label 1, Figure 7.1), forming a stream of workflows that arrive at unpredictable times in the submission pool. An arrival to the submission pool triggers the scheduler to pick all available workflows from the pool (labels 2 and 3, Figure 7.1). The scheduler then finds all the ready tasks from the workflows and adds these tasks to the central queue (label 4). A task is ready when all of its parents have been executed and all its required data have been provided. Therefore, the central queue contains tasks from different workflows that are ready to execute and exhibit no dependencies. The scheduler's role is then to find the most suitable task for execution from the queue and does so by prioritizing them (label 5). The next step for the scheduler is to initiate execution of the selected tasks based on the current status of the systems resources, and does so by coordinating with the resource provisioning block. Specifically, the resource provisioning block within the WaaS responds to the queries received by scheduler regarding resource availability (labels 6 and 7). The resource provisioning block is responsible for managing all instances and performs actions such as deploy, migrate, suspend, resume and shut down. Multiple instances are launched dynamically to fulfill service requests, with the ability to run concurrent applications on various operating systems on a single physical machine (labels 8 and 9). After scheduling a task on a resource, the central queue is updated by the scheduler, which may add additional ready tasks.

7.3 Workload Model

A workload is defined as a set of all workflows in the workflow submission pool. Let G_k denote a workflow indexed by k, therefore the workload ($\mathbb{G}$) is expressed as

$$\mathbb{G} = \cup_{k=1}^{N} G_k. \tag{7.1}$$

A Directed Acyclic Graph (DAG) is the most common representation of a workflow [28]. A workflow is defined as a graph $G_k = (T, E)$ where $T = \{t_0, t_1, ..., t_n\}$ is a set of tasks represented by vertices and $E = \{e_{i,j} \mid t_i, t_j \in T\}$ is a set directed edges denoting data or control dependencies between tasks t_i and t_j. The only difference with defined notations in section 3.3.1 is calculating the total cost for scheduling multiple workflows.

The cost of executing task t_i on instance p_j is calculated as:

$$TaskCost_{t_i}^{p_j} = \left\lceil \frac{w_{t_i}^{p_j}}{N_t} \right\rceil * c_j, \tag{7.2}$$

where c_j is the cost of instance p_j for one time interval and N_t is the time of an interval, and $w_{t_i}^{p_j}$ is the execution time of task t_j on the instance p_j. The cost of executing all tasks in a DAG is defined as:

$$Cost_G = \sum_{t_i \in G} TaskCost_{t_i}^{p_j}, \tag{7.3}$$

where G is the DAG. Finally, the overall cost of executing all DAGs in a workload is defined as:

$$Cost_o = \sum_{G \in \mathbb{G}} Cost_G, \tag{7.4}$$

where $\mathbb{G}$ is the workload defined in equation (7.1).

7.4 The DWS algorithm

In this section, I describe the Dynamic Workload Scheduler (DWS) algorithm – I start with the pseudo code given in algorithm 1. In the remainder of this section, I dig deeper and explore in detail the various different phases of the algorithm for a given workflow G_k from the workload $\mathbb{G}$ as defined in equation 7.1.

7.4.1 Workflow Partitioning

In this step, the tasks are partitioned into different levels based on their respective dependencies. Subsequently, the user-defined deadline (D) and user-

Algorithm 1 Main DWS Algorithm

```
1: procedure DWS
2:     while Central Queue is not empty do
3:         if a new DAG has arrived then
4:             call Algorithm 2
5:         end if
6:         t_i ⟵ select the task with the highest priority
           based on EDF (explained in section 7.4.3)
7:
8:         for all instances p_j ∈ P do
9:             calculate the Time set as defined in equation (7.5)
10:            calculate the Cost set as defined in equation (7.6)
11:            calculate the CTTF as defined in equation (7.7)
12:        end for
13:        BestInstance ⟵ choose the instance that
           has the highest CTTF value
14:
15:        Schedule task t_i on BestInstance
16:
17:        Update the Central Queue
18:    end while
19: end procedure
```

defined budget (B) are distributed over the levels established in the preprocessing step. Each level gets its own level deadline and budget and all tasks in the same level have the same level-deadline and level-budget. This partitioning is in the same way as I described in 5.2.2.

7.4.2 Deadline Distribution

In previous section 5.3.3.2, I explored a range of strategies for distributing a deadline over a workflow. I found that the choice of distribution strategy has a significant impact on performance. Some strategies exhibit performance that is strongly dependent on the workflow size and structure, including process, pipeline, data distribution, data aggregation and data redistribution. In general, the strategy that takes into consideration the execution time of each level as well as number of tasks in the level yields the lowest cost. The explanation of deadline distribution and its notations is outlined in section 6.2.3.

Algorithm 2 Deadline Distribution

```
1: procedure DISTRIBUTE DEADLINE(DAG, D)
2:     for all task t_i ∈ DAG do
3:         calculate the L(t_i) as defined in equation (4.1)
4:     end for
5:     categorize tasks on tasks level set as defined in equation (4.2)
6:     for all levels in DAG do
7:         calculate the level deadline as defined in equation (6.5)
8:     end for
9:     put t_entry on Central Queue
10: end procedure
```

7.4.3 Task Selection

In DWS, tasks are selected from the global queue containing all ready tasks from different workflows. A task is ready when all of its parents have been executed

and all its required data are readily accessible. As a consequence, there are no dependencies between tasks waiting in the ready list.

To select a task, at first all tasks in the ready list need to be prioritized. In this chapter, tasks are prioritized based on the Earliest Deadline First (EDF) strategy. EDF gives priority to the tasks having the nearest deadline. Therefore, the task with the earliest deadline has the highest priority and it is the best candidate for execution.

7.4.4 Instance Selection

At the point the algorithm performs instance selection, each task of received workflows is already assigned to a level, the deadline for each level is already determined, and the priority of each ready task is assigned. The Instance selection phase aims to identify the most appropriate instance to execute the current task. I therefore need to trade off execution time and cost. Firstly, the time needed for the current task, t_i, on the instance p_j is denoted by $\mathrm{ECT}\,(t_i|p_j)$. The ECT is the earliest time that a task can finish on an instance, which is defined earlier in equation (4.4). Using this observation, I can then compute how much the estimated level deadline of the current task differs from the earliest completion time of the task on the instance p_j:

$$\mathrm{Time}\ {}^{p_j}_{t_i} = \frac{subD^{\ell_{t_i}} - \mathrm{ECT}\,(t_i|p_j)}{subD^{\ell_{t_i}} - \mathrm{ECT}\,(min)}. \tag{7.5}$$

In equation (7.5), $subD^{\ell_{t_i}}$ is the deadline that is assigned to the level that contains the task t_i. Also, $\mathrm{ECT}\,(min)$ is the minimum execution time among all instances that keeps the current task on schedule.

The values of Time for task t_i are related to instance types, wherein a lower value of Time means running on a cheaper instance. The reason is that the value of $\mathrm{ECT}\,(t_i|p_j)$ is bigger on an instance with a lower processing capacity. Also, if the value of Time is negative, it means that the current task on the selected instance will exceed the level deadline i.e. $\mathrm{ECT}\,(t_i, p_j) > Level^{\ell_{t_i}}_{deadline}$.

In the Cost set given earlier in (7.2), $TaskCost_{t_i}$ refers to the cost of scheduling the current task t_i on instance p_j. In equation, (7.6), the worse cost (maximum cost) and best cost (minimum cost) of executing the task t_i among all instances are C_{worse} and C_{best}, respectively.

$$\mathrm{Cost}\ {}^{p_j}_{t_i} = \frac{TaskCost_{worse} - TaskCost_i}{TaskCost_{worse} - TaskCost_{best}} \tag{7.6}$$

To find the best instance, a Cost Time Trade-off Factor (CTTF) in equation (7.7) is used that considers a trade-off between cost and time.

$$\mathrm{CTTF}\ {}^{p_j}_{t_i} = \frac{\mathrm{Cost}\ {}^{p_j}_{t_i}}{\mathrm{Time}\ {}^{p_j}_{t_i}} \tag{7.7}$$

When an instance is first provisioned, the instance is billed on an hourly interval until it is terminated. Therefore the first task assigned to an instance in a particular billing interval is assigned the entire cost of that interval. As a consequence, if other tasks can be executed during that paid interval, once the first task has been completed, then their individual execution cost is effectively zero. Therefore, during instance selection, I first prioritize the reuse of such instances (when Cost in equation (7.6) is 1), providing that the level deadline is not exceeded (when Time in equation (7.5) is positive). If there are more than one paid instances, the DWS selects the one that has the minimum execution time (faster instances). If no such instances are available, I will attempt to use a provisioned but as yet unused (in this interval) instance or, as a last resort, provision an entirely new instance.

7.5 Evaluation

The CloudSim simulator was used to compare the performance of both algorithms. The simulation scenario was configured as a single data-center and six different instance types. The pricing model and other characteristics were explained in section 4.3.1.

In order to evaluate the performance of the algorithms with a realistic load, three different workloads with 1000, 2000 and 4000 workflows were considered. Each workload was a combination of common scientific workflows: Cybershake, Epigenomics, Montage, LIGO and SIPHT. Each workflow could have different sizes, which are 100, 200, 500 and 1000 tasks. The arrival rate for each workload was modeled based on a Poisson distribution. The characteristics and task composition of these workflows have been analyzed in published works cited in the related work section [21, 75]. In order to evaluate the performance of these algorithms, different deadlines were chosen from tight to relaxed. For this purpose, the fastest schedule (denoted by FS) was calculated as a baseline schedule. Effectively, this baseline is the fastest possible execution – ignoring costs. The fastest schedule is expressed as:

$$FS = \sum_{t_i \in CP} (w_i^j), \tag{7.8}$$

where w_i^j is the computation cost of task t_i on the fastest instance p_j.

Using equation (7.8), a variation for deadline is calculated from tight to moderate to relaxed:

$$\text{deadline} = \alpha * FS, \quad 0 < \alpha < 20, \tag{7.9}$$

The deadline factor α starts from 1 to consider very tight values and is increased by one up to a value of 20, which results in a very relaxed deadline. To assign a deadline for each workflow, the deadline factor α is randomly chosen based on a uniform distribution.

7.5.1 Performance Metrics

To evaluate the algorithms under test, the following performance metrics were selected: Success Rate (SR), Cost Ratio (CR) and Time Ratio (TR).

- Success Rate (SR): Success rate of each algorithm (SR), calculated as the ratio between the number of simulation runs that successfully met the

scheduling deadline and the total number of simulation runs (denoted by n_{Tot}), defined as:

$$SR = \frac{n\,(k)}{n_{Tot}}, \tag{7.10}$$

where $n(k)$ is the cardinality of the set k and $n_{Tot} = 100$.

- Workload Cost (WC): To compare the cost between the algorithms, equation (7.4) is used, which calculates the achieved cost of each algorithm for executing a workload.

- Time Ratio (TR): To compare the achieved makespan between each algorithm, a Time Ratio is used, which calculated by dividing makespan of each workflow by given deadline (D).

$$TR = \frac{Makespan}{D}. \tag{7.11}$$

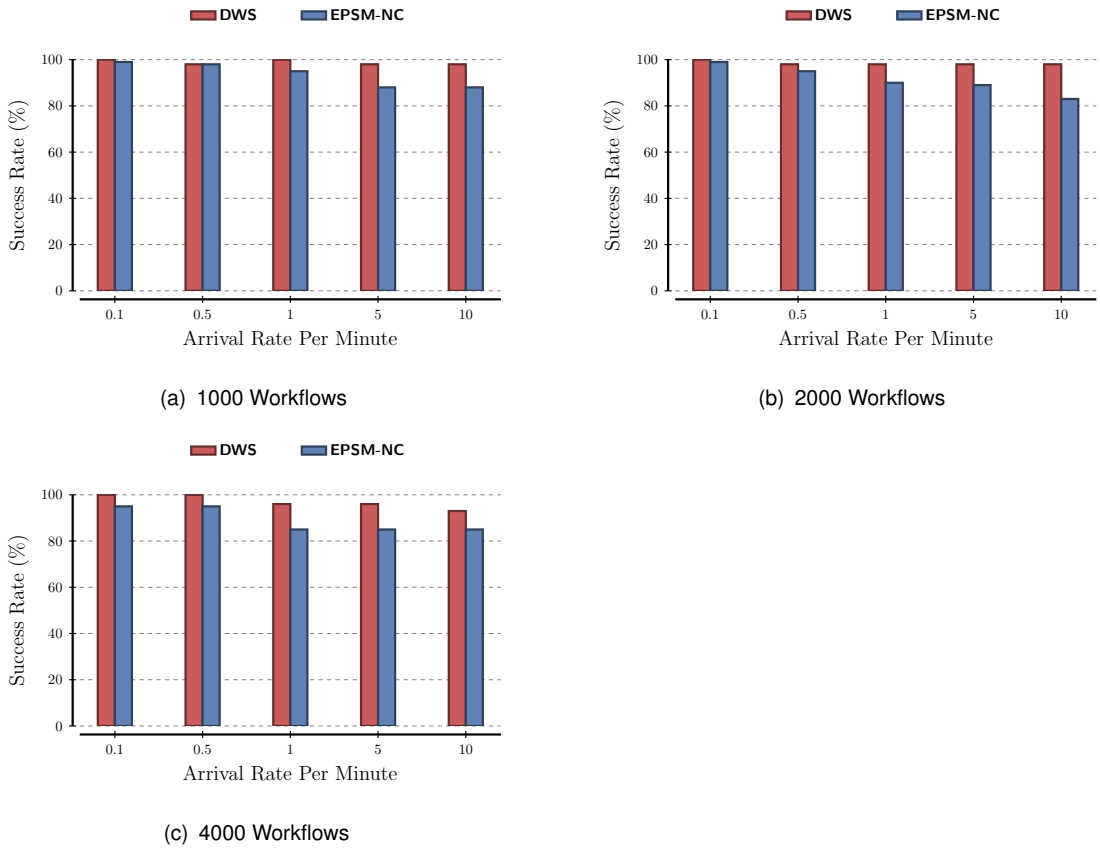
A TR value greater than 1 indicates a makespan larger than the deadline, which counts as a failure to meet the defined deadline. TR value of less than 1 indicates that the scheduled workflow meets the deadline.

7.6 Experimental Results

This section presents the performance evaluation of the DWS algorithm, with the EPSM-NC algorithm from [73], which most closely aligns with my goals and design.

7.6.1 Success Rate Analysis

The primary object for any workflow scheduling algorithm is finding a schedule without violating its constraints – in my case, the constraint is deadline. Figure 7.2 shows the relative Success Rate (SR) of each algorithm as the arrival rate is increased from 0.1 to 10. A low success rate indicates that the algorithm

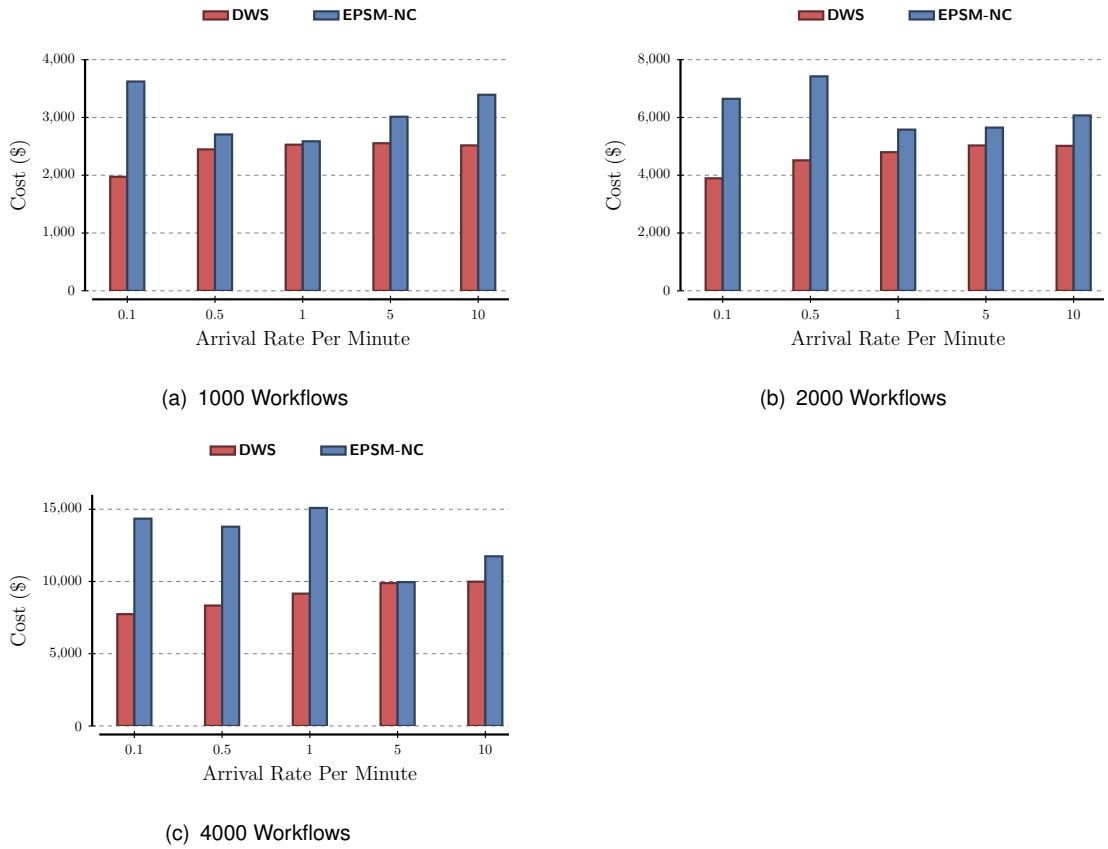


(a) 1000 Workflows

(b) 2000 Workflows

(c) 4000 Workflows

Figure 7.2: Success Rate

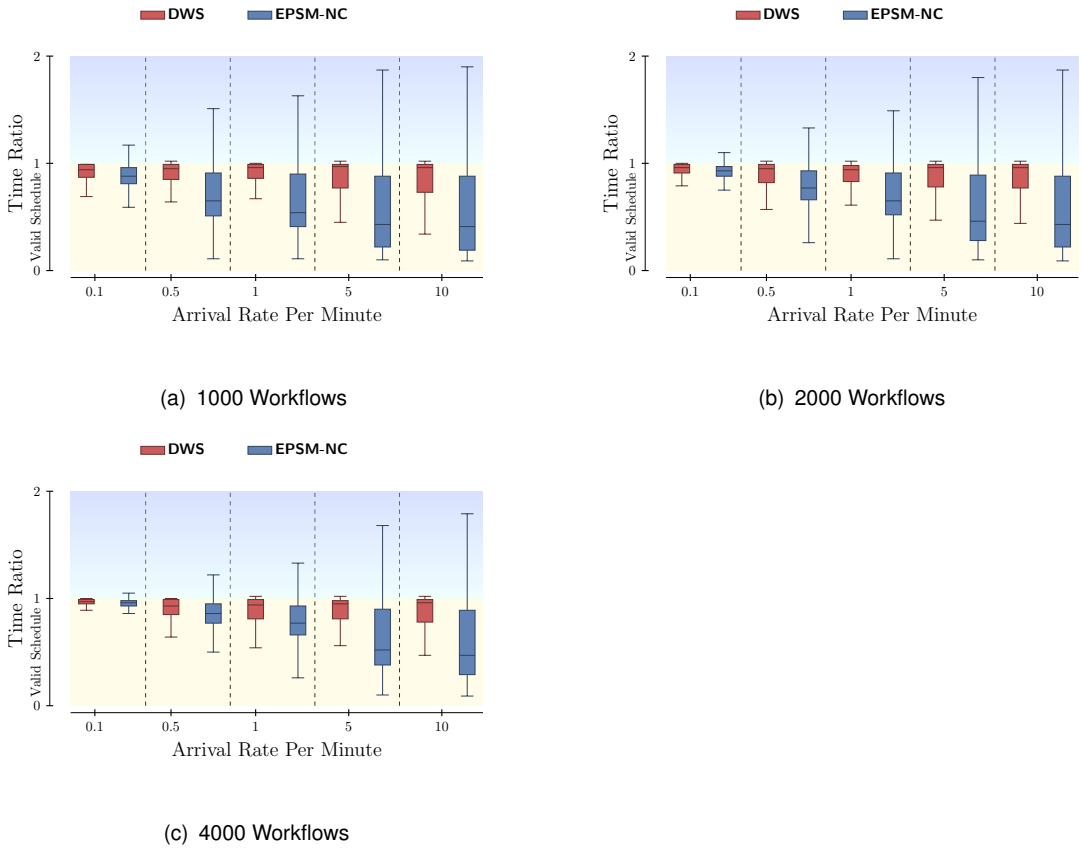


(a) 1000 Workflows

(b) 2000 Workflows

(c) 4000 Workflows

Figure 7.3: Workload Cost



(a) 1000 Workflows

(b) 2000 Workflows

(c) 4000 Workflows

Figure 7.4: MakeSpan Utilization

cannot find a makespan that meets the deadline for a workflow. No significant differences were found between the DWS and EPSM-NC whereas both exhibit nearly identical success rate for all workloads when the arrival rates were less than 1. The best overall performer is DWS, which exhibits a success rate of 96% for most deadlines in different arrival rates. As the rate of workflows per minute increases, the success rate of EPSM-NC decreases most rapidly. When arrival rate is 12 workflows per minute, almost 15%-18% of workflows fail to meet the deadline when scheduled by EPSM-NC. For the same experimental conditions, DWS experiences a 4%–7% failure rate.

7.6.2 Cost Comparison Analysis

In this section, the cost of each algorithm is evaluated using six different instance types with different characteristics as described in Table 4.6. In general, DWS experiences a monotonic increase in cost in respect to an increase in the arrival rate of workflows. However, this is not the case for EPSM-NC, which exhibits an inconsistent non-linear relationship. It would be reasonable to expect a linear increase as each algorithm would be expected to lease more VMs to meet the deadline for each workflow.

Both algorithms utilize any idle time slots formed between scheduled tasks on each resource to maximize the overall instance utilization without a need to request a new instance. Therefore, this difference is due to the trade-off between time and cost (see equation 7.7) in the DWS algorithm. Overall, the DWS outperforms the EPSM-NC algorithm, achieving the lowest overall cost over all workflows in a workload. In all cases, the DWS outperforms the EPSM-NC algorithm, achieving the lowest overall cost over all workflows in a workload.

The EPSM-NC experiences considerably higher cost when arrival rate is low (0.1 workflow per minute). In section 4.2.2.1, I explored the importance of, and sensitivity to, the prioritization of tasks in the task selection phase. In the published EPSM-NC algorithm [73], tasks are selected without any prioritization and this is the likely major source of the cost differences between the algorithms.

As a final observation, from Figure 7.3, the cost of scheduling a workload is close for both algorithms when arrival rate is high. For example, in the workload of 4000, when arrival rate is 5, DWS and EPSM-NC achieves almost the same cost. The number of instances provisioned depends on the arrival rate of workflows. When arrival rate is high, additional instances are requested to meet the deadline constraints. At the same time, there is a opportunity to better use idle times, leading to increase in VM utilization.

Overall, DWS is shown to achieve considerably lower costs by an average 23% when compared to EPSM-NC for all the workloads.

7.6.3 Deadline Utilization

The behavior of each algorithm on utilizing deadline, which is defined as Time Ratio (TR) in equation (7.11), is shown in Figure 7.4. Box plots are used, which are divided into two areas named valid schedule and invalid schedule. The valid schedule starts from 0 to 1, indicating an algorithm could meet the define constraint. Therefore, time efficiency (valid schedule) for a given deadline factor not only shows if a schedule without violation is found but also represents the distribution of deadline. Any ratio greater than 1 means a failure to generate a valid schedule – therefore, only values up to 2 are shown to better focus on the results with valid schedules.

In terms of deadline consumption, DWS utilized up to 100% of assigned deadline in all workloads as shown in Figure 7.4. Indeed, my algorithm uses almost the entire deadline (nearly all time ratio values are inside the valid schedule) while finding a schedule with lower cost as discussed in previous section. The EPSM-NC is able to often complete the execution of workflows sooner than the deadline (smaller values in the valid box) – but suffers from a greater variation and more failure. Time ratios greater that 1 means failure that is observable in Figure 7.4. These failures are corresponding to the discussed success rates in Figure 7.2.

7.7 Summary

This chapter presented a novel algorithm to address the problem of multiple workflow scheduling in dynamically provisioned commercial cloud environments. My approach addresses dynamic workflow scheduling within the unique characteristics of cloud platforms, such as on-demand provisioning and instance heterogeneity, while simultaneously meeting deadline as a constraint. The presented algorithm exhibits a success of above 93% for all workloads considering different arrival rates. Indeed, the results demonstrate that the proposed DWS algorithm utilizes deadline while finding a schedule without considerable failures and lower cost.

Chapter 8

Conclusions and Future Directions

8.1 Summary of Contributions

Elastic, on-demand cloud computing enables significant computational leverage to be applied to real world problems, be they medical, commercial, industrial or scientific. From a scheduling point of view, the most interesting subset of these problems is of those that can be represented as workflows. Workflows are being extensively used by scientists to model and manage complex, compute and data-intensive experiments, and more of these workflows are being progressively moved onto commercial clouds.

In this regard, this thesis addresses the problem of scientific workflow scheduling in cloud considering user-requirements. The algorithms in this thesis address the challenges of scheduling, on-demand resource provisioning and meeting user requirements such as budget and deadline.

To find the important characteristics and challenges of scientific workflow scheduling in cloud, chapter 2 presented a detailed taxonomy, including the background and relevant aspects for workflow scheduling problem. In addition, it discussed the related work and summarized the research topics of workflow scheduling in cloud. This chapter helps researchers to understand and identify research directions in this area.

Chapter 3 described the challenges that need to be considered followed by

problem definition of workflow scheduling. The significance of user requirements such as meeting deadline, which is one of the objectives in this thesis, was discussed in section 3.1.4. To address this objective, chapter 4 presented two cost-effective and deadline constrained heuristic algorithms for scheduling scientific workflows. Both algorithms belong to the class of list-based scheduling algorithms consisting of a task prioritization phase and a task assignment phase. In the PDC, the influence of the task selection step was investigated. In the DCCP algorithm, backfilling leftover capacity (residuals) was considered in provisioned instances and applied to several different backfilling strategies. This approach was most effective in data-intensive workflows due to the reduction in data movement. Overall, both algorithms achieved a consistently high success rates and throughput, while in most cases presenting the lowest overall pay-per-use cost.

One problem in workflow scheduling is how to spend/distribute the defined budget/deadline for the best performance. For this purpose, chapter 5 explored different possible ways to distribute budget and deadline based on the dependency structure inherent in workflows levels. In terms of budget distribution, two hypotheses were investigated. Overall, the strategy that performed the best for all workflows in terms of both success rate and makespan was "All in". This strategy was proposed largely to test the hypothesis that biasing the budget distribution to the earliest levels was beneficial in terms of reducing makespan. "All in" took this to an extreme by assigning the entire budget to the first level and relying on the trickle-down mechanism to distribute budget to later levels. In terms of deadline distribution, a range of strategies for distributing a deadline over a workflow were presented. In all strategies, the deadline was divided and distributed over the workflow as a set of sub-deadline. Overall, the strategy that takes into consideration the execution time of each level as well as the number of tasks in the level yields the lowest cost.

With additional constraints imposed by users, the problem of workflow scheduling becomes even more challenging. Chapter 6 presented a new heuristic for scheduling workflows constrained by both budget and deadline. The heuris-

tic used a novel trade-off factor between time and cost to determine the most viable schedule, and used this to determine the most appropriate type of instance to provision. A sensitivity analysis was performed on this algorithm and we concluded that increasing budget on compute-intensive workflows tends to achieve higher success rate, which implies that budget was more important than deadline for such applications.

The algorithms presented in chapters 4, 5 and 6 belong to SPDP category (see section 2.4) when the number of workflows is known in advance and an unlimited number of instance types can be provisioned at runtime. To explore the use of heuristic algorithms in near-realtime situations, when scheduling of an unpredictable stream of workflows is needed, chapter 7 focused on dynamic scheduling. A novel algorithm was introduced to solve the multiple workflow scheduling constrained by deadline on the cloud. This algorithm is placed in DPDP category, when an incoming workload of non-periodic workflows can be arrived at any time. The results demonstrate that the proposed algorithm utilizes deadline while finding a schedule without considerable failures and lower cost.

8.2 Future Work Directions

The current research and results lay the ground work towards scientific workflow scheduling in cloud. In future, various extensions to the current research are possible to further enrich the problem domain. This final section of the thesis outlines some future research directions:

- **Workflow structure**

  Chapter 3 discussed eight different policies (summarized in Table 4.1) in order to show how the order of execution can influence the scheduling results, particularly the cost. The results of the experiments suggest that the structure of workflows can significantly impact on the ranking and scheduling cost. Therefore, it is crucial to develop a better understand-

ing of workflow structure and its impact on scheduling performance as a critical prelude to the design of new heuristic scheduling algorithms.

- **WaaS provider**

  Dynamic workflow scheduling is an open and difficult problem, and a workable solution will benefit science and research across many fields by decreasing cost and increasing experimental throughput. Therefore, focus on Workflow as a Service (WaaS) provider is much required in order to host many scientific applications. WaaS providers rent cloud instance from IaaS and charge users to execute their applications. In such a dynamic environment, multiple workflows can be submitted to a provider at different time intervals. WaaS should be able to dynamically schedule a workload while considering different user's requirements. The next step for advancing WaaS is to decide the right number of instances required to execute a workload in favor of minimize the cost of leased resources from the users' perspective, and simultaneously maximize the utilization from the providers' perspective.

- **Multiple Constraints** Another important consideration is the extension of the algorithms to consider more objectives such as robustness, security and privacy. For example, some scientific applications in bioinformatics and medical research domain analyze and produce sensitive files that need to be kept secure. Therefore, scheduling algorithms for such applications need to consider security and locality aspects to protect user's data.

- **Energy Efficient Scheduling**

  Data centers are expensive for hosting scientific applications and consume huge amount of energy. Based on the recent report in [95], data centers in U.S. consumed about 1.8% of total power consumption, and registered a 4% increase from 2010 to 2014. Therefore, more studies in workflow scheduling in cloud considering energy efficiency are much required.

- **New cost model**

Some cloud providers, such as Amazon, charge users on an hourly interval from the time of provisioning, even if the instance is only used for a fraction of that period. However, at the final stage of writing this thesis, Amazon instance prices have been changed [1]. Instances that are launched in On-Demand, Reserved, and Spot form will be billed in one-second increments. This raises some new challenges in terms of vm utilization, profit maximization, performance variation of instances, provisioning and releasing delays of instances, that need to be considered as future directions.

[1] $https://aws.amazon.com/blogs/aws/new-per-second-billing-for-ec2-instances-and-ebs-volumes/$

Bibliography

[1] J. Towns, T. Cockerill, M. Dahan, I. Foster, K. Gaither, A. Grimshaw, V. Hazlewood, S. Lathrop, D. Lifka, G. D. Peterson, R. Roskies, J. R. Scott, N. Wilkins-Diehr, Xsede: Accelerating scientific discovery, Computing in Science Engineering 16 (5) (2014) 62–74.

[2] B. P. Abbott, R. Abbott, et al., Observation of gravitational waves from a binary black hole merger, Phys. Rev. Lett. 116 (2016) 61–102.

[3] V. Mayer-Schonberger, K. Cukier, Big Data: A Revolution That Will Transform How We Live, Work, and Think, Houghton Mifflin Harcourt, Boston, 2013.

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

[5] C. Evangelinos, C. N. Hill, Cloud computing for parallel scientific hpc applications: Feasibility of running coupled atmosphere-ocean climate models on amazons ec2, in: In The 1st Workshop on Cloud Computing and its Applications (CCA), 2008, pp. 2–34.

[6] K. R. Jackson, L. Ramakrishnan, K. Muriki, S. Canon, S. Cholia, J. Shalf, H. J. Wasserman, N. J. Wright, Performance analysis of high performance computing applications on the amazon web services cloud, in: Proc. 2nd IEEE Intl. Conference on Cloud Computing Technology and Science, CloudCom10, 2010, pp. 159–168.

[7] S. C. Park, S. Y. Ryoo, An empirical investigation of end-users switching toward cloud computing: A two factor theory perspective, Computers in Human Behavior 29 (1) (2013) 160–170.

[8] I. T. Foster, 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, Science Cloud '13, ACM, 2013, pp. 1–2.

[9] R. Chard, K. Chard, K. B. andLukasz Lacinski, R. Madduri, I. Foster, Cost-aware cloud provisioning, in: the IEEE 11th International Conference on E-Science, 2015, pp. 136–144.

[10] I. Foster, Service-oriented science: scaling the application and impact of eresearch (abstract), in: First International Conference on e-Science and Grid Computing (e-Science'05), 2005, pp. 1–2.

[11] R. Chard, K. Chard, K. Bubendorfer, L. Lacinski, R. Madduri, I. Foster, Cost-aware elastic cloud provisioning for scientific workloads, in: proceedings of the 8th IEEE International Conference on Cloud Computing (CLOUD), New York, 2015, pp. 136–144.

[12] J. Ullman, Np-complete scheduling problems, Journal of Computer and System Sciences 10 (3) (1975) 384 – 393.

[13] F. Wu, Q. Wu, Y. Tan, Workflow scheduling in cloud: a survey, The Journal of Supercomputing (2015) 1–46.

[14] R. Calheiros, R. Buyya, Meeting deadlines of scientific workflows in public clouds with tasks replication, Parallel and Distributed Systems, IEEE Transactions on 25 (7) (2014) 1787–1796.

[15] R. Sakellariou, H. Zhao, E. Tsiakkouri, M. D. Dikaiakos, Scheduling workflows with budget constraints, in: in Integrated Research in Grid Computing, S. Gorlatch and M. Danelutto, Eds.: CoreGrid series, Springer-Verlag, 2007.

[16] K. Chard, K. Bubendorfer, P. Komisarczuk, High occupancy resource allocation for grid and cloud systems, a study with drive, in: Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing, HPDC '10, ACM, New York, NY, USA, 2010, pp. 73–84.

[17] R. Moore, T. A. Prince, M. Ellisman, Data-intensive computing and digital libraries, Commun. ACM 41 (11) (1998) 56–62.

[18] F. A. da Silva, H. Senger, Improving scalability of bag-of-tasks applications running on masterslave platforms, Parallel Computing 35 (2) (2009) 57 – 71.

[19] T. D. Braun, H. J. Siegel, N. Beck, L. L. Blni, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys, B. Yao, D. Hensgen, R. F. Freund, A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems, Journal of Parallel and Distributed Computing 61 (6) (2001) 810 – 837.

[20] D. P. da Silva, W. Cirne, F. V. Brasileiro, Trading Cycles for Information: Using Replication to Schedule Bag-of-Tasks Applications on Computational Grids, Springer Berlin Heidelberg, Berlin, Heidelberg, 2003, pp. 169–180.

[21] G. Juve, A. Chervenak, E. Deelman, S. Bharathi, G. Mehta, K. Vahi, Characterizing and profiling scientific workflows, Future Generation Computer Systems 29 (3) (2013) 682 – 692.

[22] G. Berriman, A. Laity, J. Good, J. Jacob, D. Katz, E. Deelman, G. Singh, M. Su, T. Prince, Montage: The architecture and scientific applications of a national virtual observatory service for computing astronomical image mosaics, in: Proceedings of Earth Sciences Technology Conference, 2006.

[23] M. Malawski, G. Juve, E. Deelman, J. Nabrzyski, Algorithms for cost- and deadline-constrained provisioning for scientific workflow ensembles in iaas clouds, Future Generation Computer Systems 48 (2015) 1 – 18.

[24] R. Graves, T. H. Jordan, S. Callaghan, E. Deelman, E. Field, G. Juve, C. Kesselman, P. Maechling, G. Mehta, K. Milner, D. Okaya, P. Small, K. Vahi, Cybershake: A physics-based seismic hazard model for southern california, Pure and Applied Geophysics 168 (3) (2011) 367–381.

[25] G. Brumfiel, High-energy physics: Down the petabyte highway, Nature News 469 (7330) (2011) 282–283.

[26] I. Foster, The Grid: A New Infrastructure for 21st Century Science, John Wiley and Sons, Ltd, 2003, pp. 51–63.

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

[28] I. J. Taylor, E. Deelman, D. B. Gannon, M. Shields, Workflows for e-Science: Scientific Workflows for Grids, Springer Publishing Company, Incorporated, 2014.

[29] P. Mell, T. Grance, Tech. rep., National Institute of Standards and Technology (NIST), Gaithersburg, MD.

[30] Q. Zhang, L. Cheng, R. Boutaba, Cloud computing: state-of-the-art and research challenges, Journal of Internet Services and Applications 1 (1) (2010) 718.

[31] W. Wang, D. Niu, B. Li, B. Liang, Dynamic cloud resource reservation via cloud brokerage, in: 2013 IEEE 33rd International Conference on Distributed Computing Systems, 2013, pp. 400–409.

[32] Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility, Future Generation Computer Systems 25 (6) (2009) 599 – 616.

[33] N. Kokash, An introduction to heuristic algorithms, Department of Informatics and Telecommunications (2005) 1–8.

[34] K. Miettinen, F. Ruiz, A. P. Wierzbicki, Introduction to multiobjective optimization: interactive approaches, in: Multiobjective Optimization, Springer, 2008, pp. 27–57.

[35] C. R. Reeves, Modern heuristic techniques for combinatorial problems, John Wiley & Sons, Inc., 1993.

[36] Y.-K. Kwok, I. Ahmad, Static scheduling algorithms for allocating directed task graphs to multiprocessors, ACM Computing Surveys (CSUR) 31 (4) (1999) 406–471.

[37] M. Wang, W. Zeng, A comparison of four popular heuristics for task scheduling problem in computational grid, in: 2010 6th International Conference on Wireless Communications Networking and Mobile Computing (WiCOM), 2010, pp. 1–4.

[38] H. Liu, A. Abraham, A. E. Hassanien, Scheduling jobs on computational grids using a fuzzy particle swarm optimization algorithm, Future Generation Computer Systems 26 (8) (2010) 1336 – 1343.

[39] Z. Beheshti, S. M. H. Shamsuddin, A review of population-based meta-heuristic algorithms, Int. J. Adv. Soft Comput. Appl 5 (1) (2013) 1–35.

[40] C. W. Tsai, W. C. Huang, M. H. Chiang, M. C. Chiang, C. S. Yang, A hyper-heuristic scheduling algorithm for cloud, IEEE Transactions on Cloud Computing 2 (2) (2014) 236–250.

[41] J. Yu, R. Buyya, K. Ramamohanarao, Workflow scheduling algorithms for grid computing, in: Metaheuristics for scheduling in distributed computing environments, Springer, 2008, pp. 173–214.

[42] M. Maheswaran, S. Ali, H. J. Siegel, D. Hensgen, R. F. Freund, Dynamic mapping of a class of independent tasks onto heterogeneous computing systems, Journal of Parallel and Distributed Computing 59 (2) (1999) 107 – 131.

[43] Y. Yuan, X. Li, Q. Wang, Y. Zhang, Bottom level based heuristic for workflow scheduling in grids, Chinese Journal of Computers-Chinese Edition-31 (2) (2008) 282–290.

[44] J. Yu, R. Buyya, C. K. Tham, Cost-based scheduling of scientific workflow applications on utility grids, in: First International Conference on e-Science and Grid Computing., 2005, pp. 140–147.

[45] Y. Yuan, X. Li, Q. Wang, X. Zhu, Deadline division-based heuristic for cost optimization in workflow scheduling, Information Sciences 179 (15) (2009) 2562–2575.

[46] S. Abrishami, M. Naghibzadeh, D. H. Epema, Deadline-constrained workflow scheduling algorithms for infrastructure as a service clouds, Future Generation Computer Systems 29 (1) (2013) 158 – 169.

[47] L. Bittencourt, E. Madeira, HCOC: a cost optimization algorithm for workflow scheduling in hybrid clouds, Journal of Internet Services and Applications 2 (3) (2011) 207–227.

[48] E.-K. Byun, Y.-S. Kee, J.-S. Kim, S. Maeng, Cost optimized provisioning of elastic resources for application workflows, Future Generation Computer Systems 27 (8) (2011) 1011 – 1026.

[49] J. Sahni, D. Vidyarthi, A cost-effective deadline-constrained dynamic scheduling algorithm for scientific workflows in a cloud environment, Cloud Computing, IEEE Transactions on PP (99) (2015) 1–1.

[50] J. Yu, R. Buyya, A budget constrained scheduling of workflow applications on utility grids using genetic algorithms, in: Workshop on Workflows in Support of Large-Scale Science, IEEE, 2006, pp. 1–10.

[51] H. Topcuoglu, S. Hariri, M.-Y. Wu, Performance-effective and low-complexity task scheduling for heterogeneous computing, Parallel and Distributed Systems, IEEE Transactions on 13 (3) (2002) 260–274.

[52] J. Li, S. Su, X. Cheng, Q. Huang, Z. Zhang, Cost-conscious scheduling for large graph processing in the cloud, in: High Performance Computing and Communications (HPCC), 2011 IEEE 13th International Conference on, 2011, pp. 808–813.

[53] S. Su, J. Li, Q. Huang, X. Huang, K. Shuang, J. Wang, Cost-efficient task scheduling for executing large programs in the cloud, Parallel Computing 39 (45) (2013) 177 – 188.

[54] W. Zheng, R. Sakellariou, Budget-Deadline Constrained Workflow Planning for Admission Control, Journal of Grid Computing 11 (4) (2013) 633651.

[55] W. Zheng, R. Sakellariou, Budget-deadline constrained workflow planning for admission control, Journal of Grid Computing 11 (4) (2013) 633 – 651.

[56] X. Lin, C. Q. Wu, On scientific workflow scheduling in clouds under budget constraint, in: 2013 42nd International Conference on Parallel Processing, 2013, pp. 90–99.

[57] C. Q. Wu, X. Lin, D. Yu, W. Xu, L. Li, End-to-end delay minimization for scientific workflows in clouds under budget constraint, IEEE Transactions on Cloud Computing 3 (2) (2015) 169–181.

[58] L. Zeng, B. Veeravalli, X. Li, Scalestar: Budget conscious scheduling precedence-constrained many-task workflow applications in cloud, in: Advanced Information Networking and Applications (AINA), 2012 IEEE 26th International Conference on, 2012, pp. 534–541.

[59] M. Mao, M. Humphrey, Scaling and scheduling to maximize application performance within budget constraints in cloud workflows, in: Parallel Distributed Processing (IPDPS), 2013 IEEE 27th International Symposium on, 2013, pp. 67–78.

[60] A. M. Oprescu, T. Kielmann, Bag-of-tasks scheduling under budget constraints, in: Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on, 2010, pp. 351–359.

[61] J. Yu, M. Kirley, R. Buyya, Multi-objective planning for workflow execution on grids, in: Proceedings of the 8th IEEE/ACM International Conference on Grid Computing, GRID '07, IEEE Computer Society, Washington, DC, USA, 2007, pp. 10–17.

[62] J. Yu, R. Buyya, Scheduling scientific workflow applications with deadline and budget constraints using genetic algorithms, Scientific Programming 14 (3-4) (2006) 217–230.

[63] W.-N. Chen, J. Zhang, An ant colony optimization approach to a grid workflow scheduling problem with various qos requirements, Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on 39 (1) (2009) 29–43.

[64] A. Verma, S. Kaushal, Bi-criteria priority based particle swarm optimization workflow scheduling algorithm for cloud, in: 2014 Recent Advances in Engineering and Computational Sciences (RAECS), 2014, pp. 1–6.

[65] D. Poola, S. Garg, R. Buyya, Y. Yang, K. Ramamohanarao, Robust scheduling of scientific workflows with deadline and budget constraints in clouds, in: Advanced Information Networking and Applications (AINA), 2014 IEEE 28th International Conference on, 2014, pp. 858–865.

[66] K. Liu, H. Jin, J. Chen, X. Liu, D. Yuan, Y. Yang, A compromised-time-cost scheduling algorithm in swindew-c for instance-intensive cost-constrained workflows on a cloud computing platform, Int. J. High Perform. Comput. Appl. 24 (4) (2010) 445–456.

[67] L. F. Bittencourt, E. R. M. Madeira, Towards the scheduling of multiple workflows on computational grids, Journal of Grid Computing 8 (3) (2010) 419–441.

[68] H. Zhao, R. Sakellariou, Scheduling multiple dags onto heterogeneous systems, in: Proceedings 20th IEEE International Parallel Distributed Processing Symposium, 2006, pp. 1–14.

[69] U. Hönig, W. Schiffmann, A meta-algorithm for scheduling multiple dags in homogeneous system environments, in: Proceedings of the eighteenth IASTED International Conference on Parallel and Distributed Computing and Systems (PDCS06), 2006.

[70] W. Chen, Y. C. Lee, A. Fekete, A. Y. Zomaya, Adaptive multiple-workflow scheduling with task rearrangement, The Journal of Supercomputing 71 (4) (2015) 1297–1317.

[71] H.-J. Jiang, K.-C. Huang, H.-Y. Chang, D.-S. Gu, P.-J. Shih, Scheduling Concurrent Workflows in HPC Cloud through Exploiting Schedule Gaps, Springer Berlin Heidelberg, 2011, pp. 282–293.

[72] J. Wang, P. Korambath, I. Altintas, J. Davis, D. Crawl, Workflow as a service in the cloud: Architecture and scheduling algorithms, Procedia Computer Science 29 (2014) 546 – 556, 2014 International Conference on Computational Science.

[73] M. A. Rodriguez, R. Buyya, Scheduling dynamic workloads in multi-tenant scientific workflow as a service platforms, Future Generation Computer Systems.

[74] G. Juve, E. Deelman, Resource provisioning options for large-scale scientific workflows, in: eScience, 2008. eScience '08. IEEE Fourth International Conference on, 2008, pp. 608–613.

[75] S. Bharathi, A. Chervenak, E. Deelman, G. Mehta, M.-H. Su, K. Vahi, Characterization of scientific workflows, in: Third Workshop on Workflows in Support of Large-Scale Science (WORKS)., 2008, pp. 1–10.

[76] S. M. Park, M. Humphrey, Predictable high-performance computing using feedback control and admission control, IEEE Transactions on Parallel and Distributed Systems 22 (3) (2011) 396–411.

[77] X. Liu, J. Chen, Y. Yang, Temporal QoS management in scientific cloud workflow systems, Elsevier, 2012.

[78] J. Livny, H. Teonadi, M. Livny, M. K. Waldor, High-throughput, kingdom-wide prediction and annotation of bacterial non-coding rnas, PloS one 3 (9) (2008) e3197.

[79] A. Abramovici, W. E. Althouse, R. W. Drever, Y. Gürsel, S. Kawamura, F. J. Raab, D. Shoemaker, L. Sievers, R. E. Spero, K. S. Thorne, et al., Ligo: The laser interferometer gravitational-wave observatory, science (1992) 325–333.

[80] Usc molecular genomics core, `http://epigenome.usc.edu`.

[81] Y.-K. Kwok, L. Ahmad, Dynamic critical-path scheduling: An effective technique for allocating task graphs to multiprocessors, Parallel and Distributed Systems, IEEE Transactions on 7 (5) (1996) 506–521.

[82] G. Sih, E. Lee, A compile-time scheduling heuristic for interconnection-constrained heterogeneous processor architectures, Parallel and Distributed Systems, IEEE Transactions on 4 (2) (1993) 175–187.

[83] M. A. Khan, Scheduling for heterogeneous systems using constrained critical paths, Parallel Computing 38 (4) (2012) 175–193.

[84] R. Chard, K. Bubendorfer, B. Ng, Network health and e-science in commercial clouds, Future Generation Computer Systems 56 (2016) 595 – 604.

[85] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. De Rose, R. Buyya, CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms, Software: Practice and Experience 41 (1) (2011) 23–50.

[86] M. R. Palankar, A. Iamnitchi, M. Ripeanu, S. Garfinkel, Amazon S3 for science grids: a viable solution?, in: Proceedings of the 2008 international workshop on Data-aware distributed computing, ACM, 2008, pp. 55–64.

[87] S. Ostermann, A. Iosup, N. Yigitbasi, R. Prodan, T. Fahringer, D. Epema, A performance analysis of ec2 cloud computing services for scientific computing, in: Cloud Computing, Vol. 34, Springer Berlin Heidelberg, 2010, pp. 115–131.

[88] M. Mao, M. Humphrey, A performance study on the vm startup time in the cloud, in: 2012 IEEE Fifth International Conference on Cloud Computing, IEEE Computer Society, Washington, DC, USA, 2012, pp. 423–430.

[89] V. Arabnejad, K. Bubendorfer, Cost effective and deadline constrained scientific workflow scheduling for commercial clouds, in: IEEE 14th International Symposium on Network Computing and Applications, IEEE, Cambridge, MA USA, 2015, pp. 106–113.

[90] V. Arabnejad, K. Bubendorfer, B. Ng, K. Chard, A deadline constrained critical path heuristic for cost-effectively scheduling workflows, in: IEEE/ACM 8th International Conference on Utility and Cloud Computing (UCC), IEEE, Limassol, Cyprus, 2015, pp. 242–250.

[91] M. Malawski, G. Juve, E. Deelman, J. Nabrzyski, Cost- and deadline-constrained provisioning for scientific workflow ensembles in iaas clouds, in: International Conference on High Performance Computing, Networking, Storage and Analysis, SC 12, Los Alamitos, CA, USA, 2012, pp. 1–11.

[92] A. Verma, S. Kaushal, Cost-time efficient scheduling plan for executing workflows in the cloud, Journal of Grid Computing 13 (4) (2015) 495–506.

[93] M. A. Rodriguez, R. Buyya, Budget-Driven Scheduling of Scientific Workflows in IaaS Clouds with Fine-Grained Billing Periods, ACM Trans. Auton. Adapt. Syst. 12 (2) (2017) 5:1–5:22.

[94] Z. Liu, R. Kettimuthu, S. Leyffer, P. Palkar, I. Foster, A mathematical programming and simulation based framework to evaluate cyberinfrastructure design choices, in: the proceedings of the 13th IEEE International Conference on eScience, IEEE, Auckland, New Zealand, 2017.

[95] A. Shehabi, S. Smith, D. Sartor, R. Brown, M. Herrlin, J. Koomey, E. Masanet, N. Horner, I. Azevedo, W. Lintner, United states data center energy usage report.