FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

QoS Based Resource Management for Cloud Environment

Mozhgan Ghasemzadeh

Master in Information Engineering

Supervisor: Jorge Manuel Gomes Barbosa

November 2, 2016

QoS Based Resource Management for Cloud Environment

Mozhgan Ghasemzadeh

Master in Information Engineering

November 2, 2016

Abstract

Cloud computing, often referred to as simply “the cloud,” is the delivery of on-demand computing resources—everything from applications to data centers—over the Internet on a pay-for-use basis. Cloud Computing provides a simple way to access servers, storage, databases and a broad set of application services over the Internet. Cloud Computing providers such as Amazon Web Services own and maintain the network-connected hardware required for these application services, while you use what you need via a web application.

Nowdays, most scientific researchers use cloud computing infrastructure as a potential source of low-cost computing resources that can be provisioned on-demand according to a pay-per-use model. As a result, scientific applications are increasingly adopting cloud computing. To take advantage of this computing platform and maximize the performance in the execution of these applications, often described as workflows, a scheduling scheme to efficiently and effectively handle the application execution is required. Since , each workflow application has its own Quality Of Service requirements, task scheduling has become an essential, but highly demanding, tool. The scheduling problem becomes even more challenging when several QoS parameters are determined by users as objectives. The scheduling problem is well known as NP-complete. Therefore, most researchers in this field try to obtain a good solution by using meta-heuristic or search-based approaches that allow the user to control the quality of the produced solutions. However, these approaches usually impose significantly higher planning costs in terms of the time consumed to produce good results, making them less useful in real platforms that need to obtain map decisions quickly.

This thesis presents novel heuristic approaches for the task scheduling of scientific workflow applications based on the user’s QoS parameters in cloud environments. The main advantages of the proposed strategy is that it features low time complexity, making it more useful in real platforms that need to obtain map decisions on the fly. The proposed strategy uses common pricing models, i.e. hourly billing, offered by most cloud providers in business, tried to minimize the total execution time of the workflow application and, at the same time, reduce the execution cost. We proposed the novel selection policy for selecting the most appropriate VM instance for task assignment. In terms of objectives of the scheduling problem, we consider two relevant and conflicting QoS parameters, namely, time and cost. The proposed heuristic strategy is constrained by deadline and budget values as predefined the user’s QoS parameters. The proposed algorithm is evaluated with prominent real-world workflow applications, and compared with other state-of-the-art algorithms.

Acknowledgements

Foremost, I would like to express my sincere gratitude to my advisor Professor Jorge G. Barbosa for the continuous support of my M.Sc. study and research, for his patience, motivation, enthusiasm, and immense knowledge. His guidance helped me in all the time of research and writing of this thesis.

Besides my advisor, I would also like to thank my husband, Hamid Arabnejad, for his endless love and support. This thesis is dedicated to him.

Mozhgan Ghasemzadeh

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Cloud computing is one of the hottest technical topics today, with broad-ranging effects across IT, Information Architecture, Business, Software Engineering, and Data Storage. The term "Cloud" refers to both the applications delivered as services over the Internet and the hardware and system software in the datacentres that provide services. In recent years, cloud computing environments have been utilized by scientific communities to execute their scientific workflow applications. The growth of scientific workflows has also spurred significant research in the areas of generating, planning and executing such workflows in cloud platforms. Generally, cloud platforms use the pay-as-you-go model, in which computational resources or services have different prices with different performance and Quality of Service (QoS) levels. In this computing model, users consume services and resources when they need them and pay only for what they use. Cost and time have become the two most important user concerns. Thus, the cost/time trade-off problem for scheduling workflow applications has become challenging. Scheduling consists of defining an assignment and mapping to the workflow tasks onto the available resources.

Scheduling, along with many other issues in cloud computing infrastructures, has been extensively studied in the past several decades. Many complex applications in e-science and e-business can be modeled as workflows. A fundamental issue is how the workflow application should be executed on the available resources in the platform in order to satisfy its objective requirements. Task Scheduling is defined as a strategy to decide which (task selection) and where (resource selection) each application task should be executed, and it determines how the input/output data files are exchanged among them.

Motivation for the research

Many algorithms for scheduling workflow applications on cloud platform have been proposed. Most of these approaches consider only a single QoS parameters, such as minimizing total execution time, as their objective of scheduling problem. If we consider multiple QoS parameters, then the problem becomes more challenging. Many algorithms have been proposed for multi-objective scheduling, where meta-heuristic methods or search-based strategies have been used to achieve

good solutions. However, these methods based on meta-heuristics or search-based strategies usually impose significantly higher planning costs in terms of the time consumed to produce good results, which makes them less useful in real platforms that need to obtain map decisions on the fly.

The quality of the scheduling approach is calculated by two main metrics: (a) producing good results and (b) having low time complexity when employed by the scheduler in a realistic scenario. Thus, it is a challenge to develop an efficient scheduling approach to produce good results with low time complexity. These metrics have motivated us to develop QoS-based scheduling approaches to produce good solutions with low time complexity.

Contribution

The value of this research should be perceived as:

- Introducing a heuristic approach for scheduling workflow applications in the cloud computing infrastructure.
- Investigating the state-of-the-art research studies that focus on QoS scheduling algorithm in cloud computing.
- Evaluating and presenting results of the proposed algorithm for real-world scientific workflow applications.
- Simulating based on real cloud platform parameters.

Thesis Organization

This thesis divided in two main parts :

- Chapters 2 and 3 present an overview of cloud computing and the workflow scheduling problem. To be more precise, Chapter 2 . Then, Chapter 3 describes the application model, system model and QoS workflow scheduling problem, followed by a taxonomy of previous research in this area. We classified previous researches based on two QoS parameters, time and cost, which are defined as the objective target of scheduling algorithm in this thesis.
- Chapter 4 presents the proposed heuristic scheduling algorithm, namely Deadline-Budget Workflow Scheduling (DBWS). The DBWS algorithm tries to minimize total execution time and consumed cost in each step of the task assignment phase to meet the predefined constraint values for time and cost QoS parameters. Finally the conclusions and directions for future work are presented in Chapter 5

Chapter 2

Cloud Computing - Background

Introduction to Cloud Computing

The term "cloud computing" consists of two words. The word "Cloud" has been used to refer to the mesh of infrastructures, the combination of software and hardware, which offers a variety of services for end users. In the simplest form, cloud computing means that instead of all the computer hardware and software you're using sitting on your desktop, or somewhere inside your company's network, it's provided for you as a service by another company and accessed over the Internet, usually in a completely seamless way. Exactly where the hardware and software is located and how it all works doesn't matter to you, the user - it's just somewhere up in the nebulous "cloud" that the Internet represents.

Cloud computing provides reliable services through data centers, which are developed on virtualized compute and storage technologies. Users have the ability to access applications and data on-demand from anywhere in the world at any time based on pay-per-use model. In pay-per-use model, users are charged only for their usage. The ability of virtualisation is a major key of cloud computing. Virtualization helps to host multiple operating systems on a single physical machine. Every operating systems are isolated from each others and has their original configuration. For vitualization, Xen [BDF+03, ADC05] is the most used technique. By using this method, the cloud computing resources can be abstracted by creating the interface to the Virtual Machines (VMs). Each VM has its own resources such as CPU, physical memory, network address and link connections. Also, each VM has its own application and operating system.

Several definition of cloud computing are provided. The National Institute of Standards and Technology (NIST) defines cloud computing as :

> *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*[MG11]

In [BYV+09] cloud computing defined as : *A Cloud is a type of parallel and distributed system consisting of a collection of inter-connected and virtualized computers that are dynamically*

provisioned and presented as one or more unified computing resource(s) based on service-level agreements established through negotiation between the service provider and consumers.

In general, cloud computing offers three main features, both to cloud providers and end users : *resource flexibility*, *high performance computing* and *cost efficiency*. In the user perspective, the resources can be increased or decreased by the users according to their applications and requirements. Adding resources comes with additional charges for users; However, hosting application in cloud platforms is less costly than setting up and maintaining them in a local infrastructure, even for large-scale services that require thousands of resources, such as Netflix[1]. From the cloud providers perspective, they can achieve cost benefits by sharing their infrastructure across multiple tenants.

The Figure2.1 shows the service models and deployment model that are present in the cloud. In following paragraphs, each parts will be described in detail.

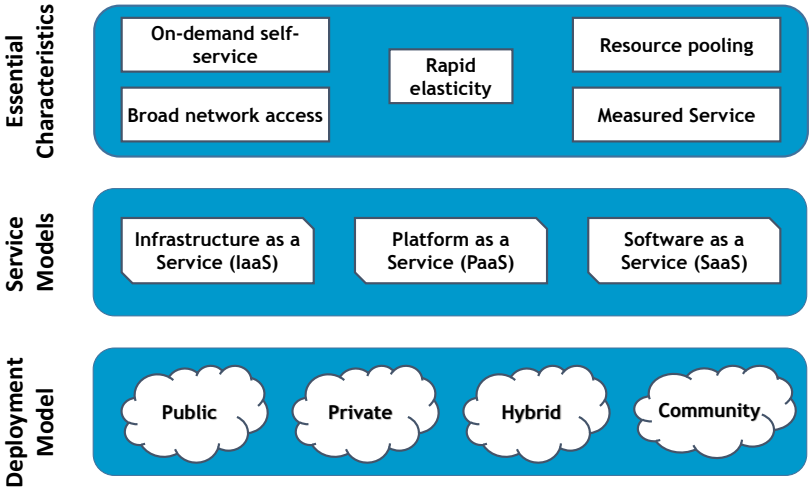

Figure 2.1: The NIST visual model of cloud computing definition

Essential Characteristics

Here are the five main characteristics that cloud computing offers businesses today.

- *On-demand self-service*: computer services such as email, applications, network or server service can be provided without requiring human interaction with each service provider. An important point is that what you are using is service-based (I need 10 computing units). You do not know, and in most cases you should not care where your computing resources really being allocated. Cloud providers such as Amazon Web Services (AWS), Microsoft, Google and IBM are providing on demand self services.

[1] https://aws.amazon.com/solutions/case-studies/netflix/

- *Broad network access*: You can access these resources from anywhere you can access the Internet, and you can access them from a browser, from a desktop with applications designed to work with them, or from a mobile device.
- *Resource pooling*: The provider's computing resources are pooled to serve multiple consumers using a multi-tenant model, with different physical and virtual resources dynamically assigned and reassigned according to consumer demand. There is a sense of location independence in that the customer generally has no control or knowledge over the exact location of the provided resources but may be able to specify location at a higher level of abstraction (e.g., country, state, or datacenter). Examples of resources include storage, processing, memory, network bandwidth, and virtual machines.
- *Rapid elasticity*: Cloud services can be rapidly and elastically provisioned, in some cases automatically, to quickly scale out and rapidly released to quickly scale in. To the consumer, the capabilities available for provisioning often appear to be unlimited and can be purchased in any quantity at any time.
- *Measured Service*: Cloud computing resource usage can be measured, controlled, and reported providing transparency for both the provider and consumer of the utilised service. Cloud computing services use a metering capability which enables to control and optimise resource use. This implies that just like air time, electricity or municipality water, IT services are charged per usage metrics – pay per use. The more you utilise the higher the bill.

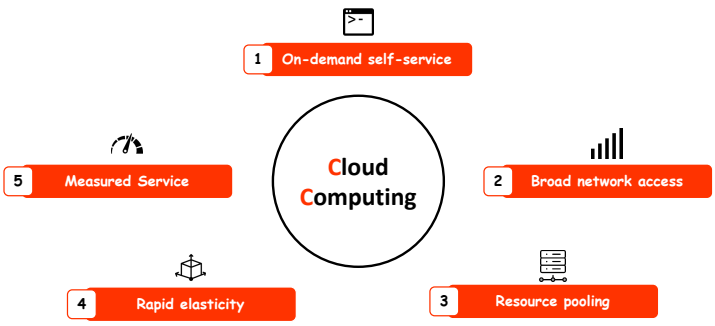

Figure 2.2: The five main characteristics of cloud computing

Service Models

Cloud Computing provides a variety of services. The service models are usually shown as "XaaS" or "Something" as a Service. Cloud services are shown in Figure 2.3. Cloud services are categorized into three different categories : *Infrastructure as a Service (IaaS)*, *Platform as a Service (PaaS)* and *Software as a Service (SaaS)*. Each of these categories offers various services to end users.

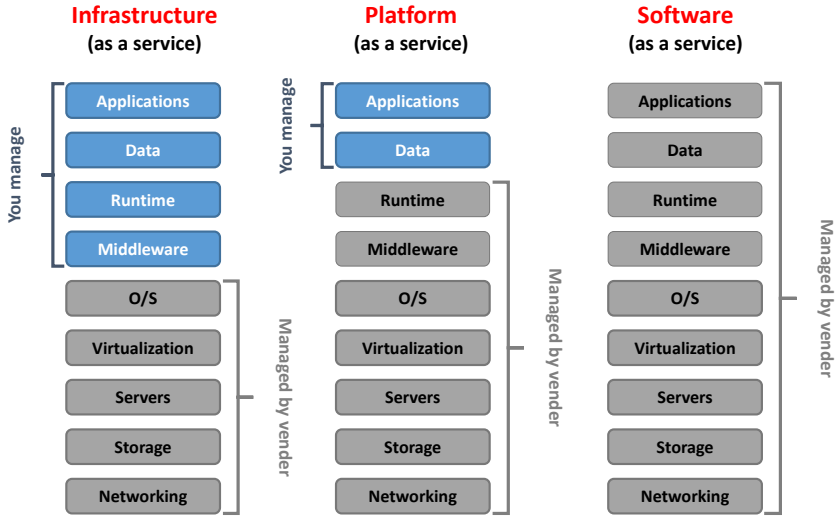

Figure 2.3: Service models

Infrastructure as a Service (IaaS)

Infrastructure as a Service (IaaS) contains the basic building blocks for cloud IT and typically provide access to networking features, computers (virtual or on dedicated hardware), and data storage space. It provides access to hardware (server, storage and network), and associated software (operating systems, virtualization technology, file system), as a service [BJJ10]. To deploy your applications in IaaS, you have to install OS images and related application software on the cloud infrastructure. In this model, it's your responsibility to patch/update/maintain the OS and any application software you install. The Cloud provider will typically bill you on computing power by the hour and the amount of resources allocated and consumed (as per its service level agreement (SLA). The resource delivery process in IaaS clouds is based on virtualisation technologies. Virtualization technologies enable multiple virtual Machines with separate operating systems which are isolated from each other, on the same physical host. Virtualization helps users to have their customized framework. Another benefit of Virtualization is that, IaaS providers can supply virtually unlimited resource instances to the user and use hosting hardware in an efficient way. There are several IaaS providers such as Amazon Elastic Compute Cloud (EC2), Microsoft Azure, GoGrid and Rackspace.

Platform as a Service (PaaS)

Platforms as a service remove the need for organizations to manage the underlying infrastructure (usually hardware and operating systems) and allow you to focus on the deployment and management of your applications. This helps you be more efficient as you need not be concerned about the running OS or updates (service packs) and hardware upgrades. The Provider regularly patches your OS, updates platform features (such as the core .NET platform or SQL database engine) and updates hardware on demand to meet your demand. In simple terms, PaaS helps users to deploy

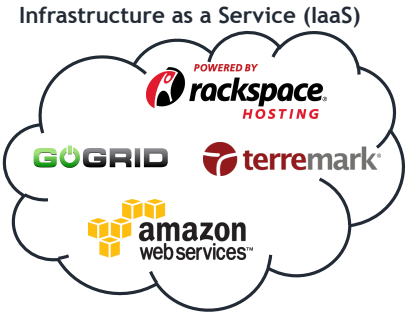

Figure 2.4: example of IaaS cloud providers

their own application with provided programming languages, libraries, services and tools, without knowing about underlying cloud infrastructure. The service providers have responsibility of hosting, managing, maintaining and updating the underlying hardware platform and the consumers have responsibility of the implemented programs and configuration settings for cloud environment. In PaaS, simillar to IaaS, users only pay for their usage of services. PaaS allows multiple users to work on the procedure for applications development simultaneously. The main benefit of PaaS is the budget saving in implement and management of applications [GB12]. Google Apps, Microsoft Azure platform and AWS Elastic Beanstalk are three best-known examples of PaaS Clouds.

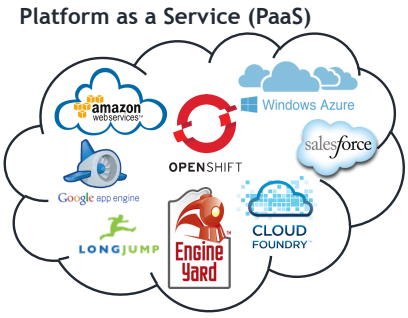

Figure 2.5: example of PaaS cloud providers

Software as a Service (SaaS)

Software as a Service (SaaS) provides you with a completed product that is run and managed by the service provider. In most cases, people referring to Software as a Service are referring to end-user applications. With a SaaS offering you do not have to think about how the service is maintained or how the underlying infrastructure is managed; you only need to think about how you will use that particular piece software. A common example of a SaaS application is web-based email where you can send and receive email without having to manage feature additions to the email product or maintaining the servers and operating systems that the email program is running on. The application service providers have the responsibility of the application hosting

and is limited by service level agreements (SLA), regard to performance, accessibility, uptime and availability. SaaS allows companies to host applications in house which help to reduce their own setup, infrastructure, licensing and maintenance costs. SaaS provides software to users with cheaper price than buying software, licenses, buying the infrastructure and maintaining it.

One of the popular examples of the SaaS model is Salesforce.com's Customer Relationship Management (CRM) software, Microsoft Office 365.

Figure 2.6: example of SaaS cloud providers

Deployment Model

A cloud deployment model represents a specific type of cloud environment, primarily distinguished by ownership, size, and access. It tells about the purpose and the nature of the cloud.

There are four common cloud deployment models [MG11]:

Public Clouds

The cloud infrastructure is provisioned for open use by the general public. It may be owned, managed, and operated by a business, academic, or government organization, or some combination of them. Services can be dynamically provisioned and are billed based on usage alone. This model is a true representation of cloud hosting, and provides the highest degree of cost savings while requiring the least amount of overhead. Public Cloud helps organizations to scale their shared resources based on their requirements [XPF13]. Generally, public cloud service providers like Amazon Web Services (AWS), Microsoft and Google own and operate the infrastructure at their data center and access is generally via the Internet.

Private Clouds

The cloud infrastructure is provisioned for exclusive use by a single organization comprising multiple consumers (e.g., business units). It may be owned, managed, and operated by the organization, a third party, or some combination of them, and it may exist on or off premises. This model does not bring much in terms of cost efficiency: it is comparable to buying, building and managing your own infrastructure.

Community Clouds

A community cloud contains features of the public and private cloud models. Like a public cloud, the community cloud may contain software, data storage, and computing resources that are utilized by multiple organizations. The community members generally share similar privacy, performance and security concerns. Where this model differs from the public model is that the infrastructure is only utilized by a group of organizations that are known to each other. Similarly to a private cloud, these organizations are responsible for the operation of their own infrastructure. The community cloud model can provide greater cost savings than the private cloud while offering some of its security features.

Hybrid Clouds

The cloud infrastructure is a composition of two or more distinct cloud infrastructures (private, community, 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). It can then use a public cloud storage provider for backing up less sensitive information. At the same time, it might share computing resources with other organizations that have similar needs. By combining the advantages of the other models , the hybrid model offers organizations the most flexibility.

In Figure 2.7 a general overview of cloud deployment models is represented.

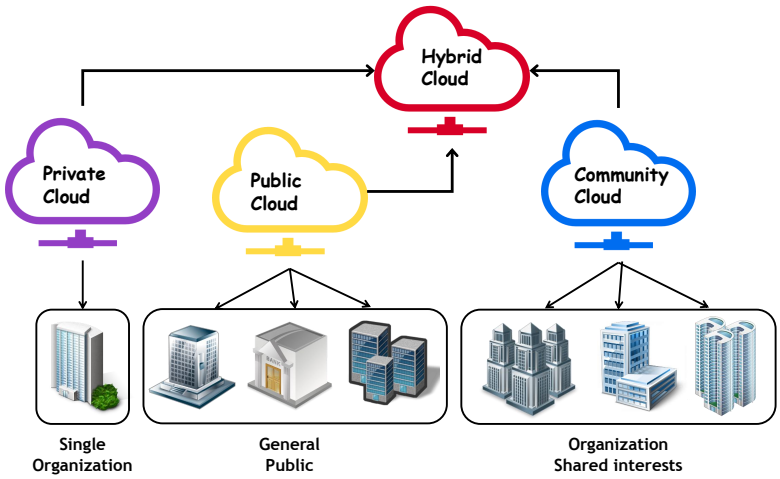

Figure 2.7: review of cloud deployment models

Chapter 3

Scheduling Problem in the Cloud

Task Scheduling is defined as a assignment strategy between a set of resources and tasks or jobs in order to meet certain performance objectives defined as the scheduling target. The efficiency of the executing application on the target platform critically depends on the methods used to find the schedule map between tasks and the set of available resources. The scheduling problem has been extensively studied in many areas, such as computer science, manufacturing, operations research and economics. Therefore, the scheduling problem has been extensively explored by researchers in the past few decades.

Scheduling can be classified into several categories such as: job-shop scheduling, multiprocessor scheduling (workflow scheduling) and real-time scheduling. The job shop problem is to determine the start and completion time of operations of a set of jobs on a set of machines, subject to the constraint that each machine can handle at most one job at a time and and each job has a specified processing order through the machines. Multiprocessor scheduling is a technique to exploit the underlying hardware in a multiprocessor system so that existing parallelism in an application program can be fully utilized and interprocessor communication time can be minimized. The real-time scheduling algorithm determines the interleaving g of execution for jobs of any real-time instance I. The goal of a real-time scheduling algorithm is to produce a schedule that ensures that every job of I is executed on the processor during its scheduling window. In this thesis, we focus on second category, i.e. workflow scheduling problem, in order to find best schedule map between interdependent tasks and available resources to meed user defined requirements.

Scheduling Structures Overview

Generally, the scheduling algorithms contain three main phases:

- *prioritizing* phase: to give a priority to each task in workflow application.
- *task selection* phase: which selects a task for scheduling in each step of the algorithm.
- *processor selection* phase: for selecting a suitable resource in order to meet the schedule objective functions.

The last two phases are repeated until all tasks are scheduled to suitable processors. However, depending on number of concurrent application to be scheduled, the definition may be redefined. In terms of number of concurrent application considered in the scheduling problem, the scheduling problem can classified into main categorises: single and multiple workflow scheduling. The only difference between these two classes is that, in the multiple workflow scheduling, new phases, namely filling the ready tasks pool phase, will be added. It is used for selecting ready tasks among available applications and to made a ready task pool, which will be used in the task selection phase. For this phase, two common methods are: selecting a single ready task from each application or add all ready tasks from each application into the ready pool.

In this thesis, we consider single workflow application and defined new strategies for each phase of scheduling process in our proposed algorithm.

Scheduling Objective

Generally, scheduling strategies generate the mapping between tasks and available resources based on some particular objectives defined by user. In this case, scheduling algorithms need to takes into account these requirement QoS parameters in their scheduling function to satisfy user requirements. The commonly used scheduling objectives in a cloud computing environment are related to the tasks completion time (makespan) and resource utilization. However, there are other QoS parameters that could be considered such as cost, energy, fault-tolerance and reliability.

Scheduling algorithms can have a single objective or multiple objectives. The scheduling problem becomes more challenging when two or more QoS parameters are considered. In this case, the scheduling algorithm tries to find a suitable schedule map between the workflow's tasks and the available resources in order to meet its objective function, which could be to optimize or to constrain the problem to a single or multiple QoS parameters.

In this thesis and in the proposed algorithm, we consider two relevant and conflicting QoS parameters, namely, time and cost as objective functions. The proposed algorithm focuses on these two QoS parameters to find a feasible schedule map that satisfies the user-defined deadline and budget constraint values.

Application Model

Generally, there are two major application models, namely, Bag-of-tasks (BoT) and the interdependent model, that used in many disciplines, including computer science, engineering, astronomy, physics and chemistry. The most obvious distinction between these two models lies in whether they have precedence constraints that task t_i can not cannot start before successful completion of its parents.

Bag-of-tasks model

Bag-of-Tasks (BoT) are parallel applications with no inter-task communication in many scientific and engineering applications. A variety of problems in several fields, such as computational biology [PBC+03], image processing [SCB02] and massive searches [AGM+90] have been modelled as BoT applications. Although these type of applications consist of independent tasks, the performance of their execution is measured by all tasks and, only after executing all tasks, the analysis can be done. Therefore, the optimization of the aggregate set of results is important, and not the optimization of a particular task or group of tasks. Due to the independence between jobs in BoT applications, they can be easily executed on multiple resources in parallel mode to reduce the total turnaround time of application or meet the application deadline parameter. A scheduler can distribute tasks of a BoT application among several resources in order to speed up its total turnaround time. The turnaround of the application is then calculated based on the finish time of all executed tasks in BoT application.

Interdependent tasks model

In this model, the tasks of application workflow are inter-dependent. A typical workflow application can be represented by a Directed Acyclic Graph (DAG), i.e., a directed graph with no cycles. A DAG can be modeled by a three-tuple $G =< T, E, Data >$. Let n be the number of tasks in the workflow. The set of nodes $T = \{t_1, t_2, \cdots, t_n\}$ corresponds to the tasks of the workflow. The set of edges E represent their data dependencies. A dependency ensures that a child node cannot be executed before all its parent tasks finish successfully and transfer the required child input data. $Data$ is a $n \times n$ matrix of communication data, where $data_{i,j}$ is the amount of data that must be transferred from task t_i to task t_j. The average communication time between the tasks t_i and t_j is defined as:

$$\overline{C}_{(t_i \to t_j)} = \overline{L} + \frac{data_{i,j}}{\overline{B}} \tag{3.1}$$

where $\overline{B}$ is the average bandwidth among all processor pairs and $\overline{L}$ is the average latency. This simplification is commonly considered to label the edges of the graph to allow for the computation of a priority rank before assigning tasks to processors [THW02].

Due to heterogeneity, each task may have a different execution time on each processor. Then, $ET(t_i, p_j)$ represents the Execution Time to complete task t_i on processor p_j in the available processors set P. The average execution time of task t_i is defined as:

$$\overline{ET}(t_i) = \frac{\sum_{p_j \in P} ET(t_i, p_j)}{|P|} \tag{3.2}$$

where $|P|$ denotes the number of resources in processors set P.

In a given DAG, a task with no predecessors is called an *entry task* and a task with no successors is called an *exit task*. We assume that the DAG has exactly one entry task t_{entry} and one exit

task t_{exit}. If a DAG has multiple entry or exit tasks, a dummy entry or exit task with zero weight and zero communication edges is added to the graph.

In addition to these definitions, there are some common attributes used in task scheduling, which will be used in the following sections.

- $pred(t_i)$ and $succ(t_i)$ denote the set of immediate predecessors and immediate successors of task t_i, respectively. $FT(t_i)$ is defined as the Finish Time of task t_i on the processor assigned by the scheduling algorithm.

- Schedule length or *makespan* denotes the finish time of the last task of the workflow and is defined as $makespan = FT(t_{exit})$.

- $EST(t_i, p_j)$ and $EFT(t_i, p_j)$: denotes Earliest Start Time (EST) and the Earliest Finish Time (EFT) of a task t_i on processor p_j, respectively, and are defined as:

$$EST(t_i, p_j) = \max\left\{ T_{Available}(p_j), \max_{t_{parent} \in pred(t_i)} \{AFT(t_{parent}) + C_{(t_{parent} \to t_i)}\} \right\} \tag{3.3}$$

$$EFT(t_i, p_j) = EST(t_i, p_j) + ET(t_i, p_j) \tag{3.4}$$

  where $T_{Available}(p_j)$ is the earliest time at which processor p_j is ready. The inner max block in the EST equation is the time at which all data needed by t_i arrives at the processor p_j. The communication time $C_{(t_{parent} \to t_i)}$ is zero if the predecessor node t_{parent} is assigned to processor p_j. For the entry task, $EST(t_{entry}, p_j) = 0$. Then, to calculate EFT, the execution time of task t_i on processor p_j (ET) is added to its Earliest Start Time.

Cloud Providers

Workflow Scheduling

Workflow scheduling has been extensively investigated. Considering the number of workflow applications in the scheduling problem, the scheduling strategies can be classified into two main classes: single and multiple workflow scheduling. In terms of time complexity, workflow scheduling strategies have been proposed under two main categories: heuristic and search-based or meta-heuristic approaches. The heuristic-based algorithms allow approximate solutions - often good solutions, but not necessarily the best ones - with low time complexity. On the other hand, search-based or meta-heuristic algorithms may achieve better solutions by performing more iterations, which results in higher running time than heuristic methods. In this section, we present a brief survey of task scheduling algorithms.

Single Workflow Scheduling Algorithms

Single workflow scheduling algorithms are designed to schedule only a single workflow at a time. If all information about tasks, such as execution and communication costs for each task and the relationship with other tasks are known beforehand, the scheduling method is categorized as a *Static scheduling* strategy; if such information is not available and decisions are made at runtime, it is categorized as a *Dynamic scheduling* strategy. Dynamic scheduling is adequate for situations where the system and task parameters are not known at compile time, which requires decisions to be made at runtime but with additional overhead. A sample environment is a system where users submit works, at any time, to a shared computing resource. In this situation, a dynamic algorithm is required because the workload is only known at runtime, as is the status of each processor when new tasks arrive and, consequently, cannot optimize any QoS parameters based on the entire workflow. By contrast, a static approach can optimize or be constrained to QoS parameters, which are defined as schedule objectives or constraints, by considering all tasks independently of execution order or time because the schedule is generated before execution begins. In this thesis, in the case of single workflow scheduling, we consider that the information about the system and the workflow are known at compile time. Generally, the scheduling algorithms for a single workflow contain three main phases: the *prioritizing* phase, to give a priority to each task; the *task selection* phase, which selects a task for scheduling; and the *processor selection* phase, for selecting a suitable processor in order to meet the schedule objective functions. The last two phases are repeated until all tasks are scheduled to suitable processors.

The scheduling problem is further characterized as single or multi-objective, as one or several QoS parameters are considered as the objectives of scheduling algorithm.

Cost-optimization, deadline-constraint

Abrishami et al. [ANE13] proposed two scheduling algorithms named IaaS Cloud Partial Critical Paths (IC-PCP) and the IaaS Cloud Partial Critical Paths with Deadline Distribution (IC-PCPD2) were proposed for cost minimization constrained to a deadline, extending their previous PCP algorithm in [ANE12]. The PCP scheduling algorithm is proposed for utility grid computing environments. In order to consider some differences between commercial cloud environments and utility grid models such as (i) the on-demand (dynamic) resource provisioning, (ii) the homogeneous bandwidth among resources and (iii) the pay-as-you-go pricing model, the authors adapted the PCP scheduling algorithm and proposed two novel workflow scheduling algorithms, i.e., IC-PCP and IC-PCPD2, for IaaS Cloud environments. The main differences between two proposed algorithms and the original PCP algorithm are the deadline distribution and the planning phases which are modified to adapt to the cloud platforms. In other hand, IC-PCP algorithm is a one-phase algorithm, whereas IC-PCPD2 is a two-phase algorithm. The main idea of these algorithms is to distribute the given deadline among tasks of the workflow. It starts by assigning the given deadline as the latest finish time of the exit task. Then, it continues by determining the partial

critical path (PCP) for the exit task, which is the longest path to the exit node. The IC-PCP algorithm schedules each partial critical path on a single cloud resource type, while IC-PCPD2 is a two-phase algorithm which, first, distributes the overall deadline on the workflow tasks and, then, schedules each task based on its subdeadline. Later, in. [CB14], Rodrigo *et al.* have proposed the Enhanced IC-PCP with Replication (EIPR) algorithm, an extension to IC-PCP able to use idle instances and budget surplus to replicate tasks, which considers data transfer time and cloud resources boot time during the provisioning and scheduling process. The EIPR algorithm uses idle time of provisioned resources to the replication of tasks in order to increase the chance of meeting the application deadlines. The experimental results show that the likelihood of meeting deadlines is increased by using task replication. Authors in [CLCG13] proposed a bi-direction adjust heuristic (BDA) to assign tasks to suitable VM instances while maintaining the given deadline for the workflow application. The BDA scheduling algorithm consists of two stages. In the first stage, by ignoring the time interval pricing model in cloud environments, each task is assigned to the appropriate VM type. In the initialize tasks assignment, the BDA scheduling algorithm uses CPLEX to get an initial scheduling map. At the second stage, a bi-direction adjust process, composed of the forward and backward scheduling procedures, is applied to allocate tasks to VM instances according to the initial scheduling map. Sharif *et al.* in [STZN13] proposed the Multi-terminal Cut for Privacy in Hybrid Clouds (MPHC) to minimize the cost of executing workflows, while satisfying both task/data privacy and deadline constraints. In [CLG16], the authors proposed a multiple complete critical-path based heuristic (CPIS) to schedule tasks in XaaS clouds. The CPIS algorithm divides the workflow deadline into task deadlines, then a proposed list based heuristic namely LHCM is used to schedule tasks on rented shareable service instances. Mao *et al.* [MH11] proposed the Scaling-Consolidation-Scheduling (SCS) scheduling algorithm to ensure that all tasks are finished before the given deadline at minimum financial cost. The SCS algorithm tries to minimize the execution cost by using heuristic strategies such as task bundling (merging tasks into a single one and execute them in the same instance to save data transfers by using the temporary results stored locally), schedules task on the most cost-efficient instances and instance consolidation strategy in order to achieved the full utilization of all instances. With the same objective function, in [BM11], a Hybrid Cloud Optimized Cost (HCOC) algorithm combines the usage of private and public clouds. HCOC decides which resources should be leased from the public cloud to increase the processing power of the private cloud to execute a workflow within its deadline. In [KKYC14], heuristic based workflow scheduling scheme composed of two phases, namely: VM packing and MRSR (Multi Requests to Single Resource) phases, is proposed. In VM packing phase, the algorithm tries to combine tasks assigned to different VM generated by any existing algorithm into a single VM. Then, the packed tasks are merged in parallel with the deadline assurance in the MRSR phase. Malawski *et al.* [MFB+15] present a cost optimization model for scheduling scientific workflows on IaaS clouds under a deadline constraint. Authors in [WHLR16] proposed a heuristic algorithm called minimal slack time and minimal distance (MSMD) to schedule a workflow application within the deadline constraint value. The MSMD algorithm minimizes the number of allocated instances and the total VM instances/hour while the deadline constraint

value for the workflow application is guaranteed. In [CS13], a level based scheduling algorithm to minimize the total execution cost while meeting the deadline on hybrid cloud environment was proposed. By using hybrid cloud, the resources can be provided by the public or the private cloud platform. The proposed algorithm assigns a sub-deadline to each task based on the total deadline of the workflow and tasks' runtime and communication. Then, if the task can be executed on the resource from the private cloud without its sub-deadline violation, it will execute on the private cloud. Otherwise, it is allocated to an appropriate resource on the public cloud. Verma *et al.* [VK14b] proposed a Deadline Constrained Heuristic based Genetic Algorithms (HGAs) to schedule applications to cloud resources that minimise the execution cost while meeting the deadline. In [CZ12], a discrete version of the comprehensive learning PSO (CLPSO) algorithm based on the set-based PSO (S-PSO) method for the cloud workflow scheduling problem is proposed.

Time-optimization, budget-constraint

Zeng *et al.* in [ZVL12] proposed ScaleStar, a budget-conscious scheduling algorithm to effectively trade-off between the execution time and cost for the execution of many large scale many-task workflows in clouds with budget constraints. They proposed a novel objective function namely comparative advantage (CA) which effectively balances the execution time-and-monetary cost goals. The ScaleStar assigns the selected task to the resource with higher comparative advantage value. In first the phase, each task in the workflow application is assigned to the resource with highest CA1 value to effectively deal with time-cost metric trade-off for the current task. Later, to consider the budget constraint parameter, this initial assignment of the tasks will be changed by recomputing the CA2 value which compares the total execution time and cost for each task reassignment. Fard *et al.* [FFP13] proposed a cost-constraint time optimization scheduling algorithm in public commercial clouds based on a set of rescheduling operations. First, a new Cost Efficient Fast Makespan (CEFM) algorithm is proposed to optimize both the makespan and cost of a workflow execution. The CEFM algorithm starts by assigning each task to a VM instance, which executes the task with the lowest finishing time. Then, based on this initial mapping, CEFM tries to reduce the total cost of the workflow execution without increasing the makespan. By using an output schedule map obtained by the CEFM approach, a Budget-Constrained Scheduling in Clouds (BCSC) algorithm is proposed to schedule a given workflow application with a nearly optimal makespan while meeting a specified budget constraint. The main idea behind BCSC is to use a tradeoff between a decremental cost obtained by rescheduling tasks to cheaper VM instances and the incremental workflow makespan. Mao *et al.* in [MH13b] proposed two auto-scaling solutions, namely scheduling-first and scaling-first, to minimize job turnaround time within budget constraints for cloud workflows. The scheduling-first algorithm distributes the total budget to each individual task based on the task priority and then assigns them to the appropriate VM instance, while the scaling-first algorithm determines the number of the cloud VM according to the given budget and then schedules the tasks based on their priority. Xiangyu *et al.* [LW13] proposed a heuristic Critical-Greedy (CG) algorithm to minimize the workflow end-to-end delay under a user-specified financial cost constraint for scientific workflows in IaaS cloud environments. The

proposed CG starts with the cheapest assignment as an initial least-cost schedules which schedule all tasks to the resource with minimum cost consumption. Then, the CG algorithm repeats the rescheduling process to reduce the total execution time until no rescheduling is feasible with the left budget. Chase *et al.* [WC16] proposed Multi-Cloud Workflow Mapping (MCWM) scheduling algorithm to minimize the total execution time under budget constraint values in IaaS multi-cloud environments. Wu *et at.* [WLY+15] proposed the Critical-Greedy algorithm scheduling to minimize the workflow makespan under a user-specified financial constraint for a single datacenter cloud. In the first step, the proposed Critical-Greedy algorithm generates an initial schedule where the cloud meets a given budget for the workflow application. Then, in the next step, by iterative searching, it tries to reschedule critical tasks in order to reduce the total execution time until no more rescheduling is possible. Zeng *et al.* [ZVL15] introduced a Security-Aware and Budget-Aware workflow scheduling strategy (SABA) to minimize the total workflow execution time while meeting the data security requirement and budget constraint in cloud environments. Taking into account data security requirements, they defined two type of datasets : *moveable* datasets and *immoveable* datasets to impose restrictions on data movement and duplication. For the scheduling phases, they introduced an objective function referred to as Comparative Factor (CF) to make a balance between execution time and consumption cost. The resource with best CF will be selected for task assignment.

Time-optimization, cost-optimization

Zeng *et al.* [ZVZ15] proposed an Adaptive Data-Aware Scheduling (ADAS) to improve the total makespan in communication-intensive workflow applications by establishing coordination between task execution and data management. By building the clusters for the workflow tasks and datasets and executing them on multiple data centres, their approach can effectively improve the workflow completion time and utilization of resources. [CWG09] introduces an optimized algorithm for task scheduling based on ABC (activity based costing) in cloud computing. The ABC algorithm assigns priority levels for each task and uses cost drivers. It measures both the cost of the objects and the performances of activities. An Improved ABC is presented in [SS10] for making efficient mapping of tasks to available resources in cloud. Nguyen *et al.* [MH13a] proposed a novel scheduling heuristics, Cost with Finish Time-based (CwFT), an extension of the popular Heterogeneous Earliest Finish Time (HEFT) scheduling algorithm, to select the best processing unit for each task of a workflow. The CwFT balances between performance of application schedule and the mandatory cost. It shows significant cost savings of workflow execution along with reasonable total execution time of workflow compared to HEFT. Jian *et al.* [LSC+11] designed a cost-conscious scheduling algorithm (CCSH) to reduce the execution time and cost of workflow application. The main idea of CCSH is using a cost-conscious factor which effectively balances the trade-off between execution time and monetary cost. Based on this cost-conscious factor, they presented a new Cost-EFT (CEFT) for selecting the best appropriate resources in the resource selection phase. Su *et al.* [SLH+13] presented a cost-efficient task-scheduling algorithm using two heuristic strategies, namely, Pareto Optimal Scheduling Heuristic (POSH) and Slack Time

Scheduling Heuristic (STSH). The POSH strategy assigns tasks to the most cost-efficient VM based on pareto dominance and, then, the STSH strategy tries to reduce monetary costs of non-critical tasks while keeping the makespan. Lee *et al.* [LZ13] proposed the critical-path-first (CPF) scheduling algorithm, which efficiently stretches out the schedule aiming to proactively preserve critical path length - CPL. Then, they developed an algorithm to compact the output schedule and minimize resource usage with no makespan increase. Bessai *et al.* [BYO+12] used the pareto approach to solve a bi-criteria (execution time and cost) for scheduling workflows on distributed cloud resources to minimize the overall cost execution and makespan of application. Their algorithm consists of three pareto approach algorithms in accordance with three resource selection polices : cost-based, time-based and cost-time-based. The first algorithm aims to minimize the the execution cost of the workflow. The second algorithm attempts to reduce the overall completion time. Finally, the third algorithm is called cost time-based approach, combines the objectives of the two first algorithms by selecting only the non-dominated solutions. Later, the same authors in [BYOG13] propose a set of algorithms for business process scheduling in Cloud computing environments. Here, simultaneously execution of the same process on several instances is taken into account. Huu *et al.* [HKA+11] proposed several different resource allocation strategies to reduce infrastructure costs while optimizing the application performance. The proposed strategies used a cost/performance trade-off to find a resource map. Yassa *et al.* [YCKG13] proposed a new approach for multi-objective workflow scheduling in clouds, and present the hybrid PSO algorithm to optimize the scheduling performance. The proposed method is based on the Dynamic Voltage and Frequency Scaling (DVFS) technique to minimize energy consumption, called DVFS Multi-Objective Discrete Particle Swarm Optimization (DVFS-MODPSO). It simultaneously optimizes several conflicting objectives, namely, the makespan, cost and energy. DVFS-MODPSO presents a set of non-dominated solutions which provides more flexibility for users to assess their preferences and select a schedule that meets their QoS requirements. A market-oriented hierarchical scheduling strategy for multi-objective in cloud workflow systems was proposed in [WLN+13]. They analyzed meta-heuristic-based workflow scheduling algorithms, such as GA, ACO and PSO, in cloud environments aiming to satisfy the QoS requirements.

Deadline-constraint, budget-constraint

Verma *et al.* in [VK14a] proposed Bi-Criteria Priority based Particle Swarm Optimization (BPSO) to schedule workflow applications on cloud environments that minimize the execution cost while meeting the deadline and budget constraints for delivering the result. They used the Particle Swarm Optimization (PSO) as an evolutionary technique to find the best schedule map between tasks and resources. Poola *et al.* [PGB+14] presented a robust and fault-tolerant scheduling algorithm in cloud environments, considering deadline and budget constraints. They proposed several resource allocation policies to solve the problem of uncertainties such as performance variations and failures in the cloud environment by adding slack time based on deadline and budget constraints. It is concluded that Robustness-Time-Cost (RTC) policy gives highest robustness and lower makespan and the Robustness-Cost-Time (RCT) Policy gives a robust schedule, but has a marginally higher

cost of execution. Rahman *et al.* [RLP11] presented an Adaptive Hybrid Heuristic (AHH) for scheduling data analytics workflows on hybrid cloud environments. The proposed AHH optimises cost of workflow execution and satisfies users requirements, such as budget or deadline. It also has the capability of adapting to changes in the cloud environment. It is designed to first generate a task-to-resource mapping with minimum execution cost using GA (Genetic Algorithm) within the user's budget and deadline. This initial schedule is then utilized to distribute the workflow-level budget and deadline to task levels. Finally, the Dynamic Critical Path (DCP) heuristic is employed to dynamically schedule the ready tasks level-by-level based on the initial schedule, budget and deadline constraints as well as changed status of resources. Shi *et al.* [SLD+16] proposed an elastic resource provisioning and task scheduling mechanism to perform scientific workflow jobs in the cloud environment. Their method tries to complete as many high-priority workflows as possible under budget and deadline constraints.

Multiple Workflow Scheduling Algorithms

In contrast to single workflow scheduling, concurrent or multiple workflow scheduling has not received much attention. As with single workflow strategies, multiple workflow scheduling algorithms can be divided into two main categories: static and dynamic strategies. In static strategies, workflows are available before the execution starts, that is, at compile time. After a schedule is produced and initiated, no other workflow is considered. This approach, although limited, is applicable in many real-world applications, for example, when a user has a set of nodes to run a set of workflows. This methodology is applied by the most common resource management tools, where a user requests a set of nodes to execute his/her jobs exclusively. On the other hand, dynamic strategies exhibit online behavior, where users can submit the workflows at any time. When scheduling multiple independent workflows that represent user jobs and are, thus, submitted at different moments in time, the completion time (or turnaround time) includes both the waiting time and execution time of a given workflow, extending the makespan definition for a single workflow scheduling. However, in both cases, single or multiple QoS parameters can be defined as the scheduling objectives.

In [ZHL16], authors proposed a workflow scheduling system, namely Dyna, in order to minimize the monetary cost of each submitted workflows in IaaS clouds while satisfying their predefined deadline. In the proposed system, the cloud performance and price are captured dynamically by Dyna. The main idea of Dyna is to find the best suitable instance for each task of a given workflow so that the total execution cost is minimized while the predefined deadline is satisfied. Li *et al.* [LZW+12] introduced trust into workflow's QoS target and proposed a novel customizable cloud workflow scheduling model. They proposed two stage workflow model: the macro multi-workflow stage is based on trust and micro single workflow stage assigning tasks to the real resources based on QoS demands. In the second stage, workflows are classified into main categories, namely time-sensitive and expenses-sensitive. For time-sensitive type, it tries to reduce the total completion time of workflow, while for expenses-sensitive type, it tries to reduce consumption cost. Li *et al.* [LQM+12] proposed two online dynamic resource allocation algorithms

for the IaaS cloud system with preemptable tasks. Both algorithms can adjust the resource allocation dynamically based on the updated information of the actual task executions. Their approach contains two online dynamic task scheduling algorithms: dynamic cloud list scheduling (DCLS) and dynamic cloud min-min scheduling (DCMMS). Experimental results show that the dynamic procedure with updated information provides significant improvement in the situation where resource contention is fierce. Sharif *et al.* [STZN14] presented two online algorithms to schedule multiple workflows under deadline and privacy constraints, while considering the dynamic nature of hybrid cloud environment. Xu *et al.* [XCWB09] proposed an algorithm was proposed for scheduling multiple workflows with multiple QoS constraints on the cloud. The resulting multiple QoS-constrained scheduling strategies of multiple workflows (MQMW) minimize the makespan and the cost of the resources and increase the scheduling success rate. The algorithm considers two objectives, time and cost, that can be adapted to the user requirements. In [ZH14], authors proposed ToF, a general transformation-based optimization framework for workflows in the cloud platform. Bochenina, in [Boc14], introduced a strategy for mapping the tasks of multiple workflows with different deadlines on the static set of resources. Jiang et al. in [JHC+11] proposed a method to minimize the total execution time of a scheduling solution for concurrent workflows in the HPC cloud. Their method tries to take advantage of any schedule gaps. First, a workflow is partitioned into several tasks, grouped by using a clustering-based PCH approach [BM10, BM07]. Then, the proposed distributed gap search is applied to allocate these task groups to processors. The difference between the original gap search algorithm and proposed distributed gap search method is that by using the original gap search method, an entire task group is allocated to a single gap on a specific resource, but the proposed distributed gap approach allows for allocating the tasks of the same group to different gaps on different resources.

Chapter 4

Deadline-Budget Workflow Scheduling (DBWS)

Recently cloud computing has gained popularity among e-Science environments as high performance computing platform. From the viewpoint of the system, applications can be submitted by users at any moment in time and have different QoS requirements. To achieve higher rates of successful applications attending to their QoS demands, an effective resource allocation (scheduling) strategy between workflow's tasks and available resources is required. Several algorithms have been proposed for QoS workflow scheduling, but most of them use search-based strategies that generally have a high time complexity, making them less useful in realistic scenarios. In this thesis, we present a heuristic scheduling algorithm with quadratic time complexity that considers two important constraints for QoS-based workflow scheduling, time and cost, named Deadline-Budget Workflow Scheduling (DBWS) for cloud environments. Performance evaluation on some well-known scientific workflows shows that the DBWS algorithm accomplishes both constraints with higher success rate in comparison to the current state-of-the-art heuristic-based approaches.

Introduction

Cloud computing infrastructures are the new platforms for tackling the execution needs of large-scale applications. Cloud computing promises the important benefits such as providing nearly-unlimited computing resources to execute application's task, on-demand scaling and pay-per-use metered service. Computing resources (i.e. virtual machines (VMs)) are dynamically allocated to user tasks based on application requirements, and users just pay for what they used. Each large-scale workflow application contains several tasks. Generally, workflow application can be represented by a Directed Acyclic Graph (DAG) that includes independent tasks, which can be executed simultaneously, or dependent task which needs to be executed in a given other. In order to meet user's application QoS parameters, we need to find an efficient schedule map to execute the application tasks on multiple resources.

The majority of studies about workflow scheduling focus on single workflow application scheduling. However, these approaches are not adequate for cloud infrastructures due to two major features: pay-as-you-go pricing model and on-demand resource provisioning. For example, in [ANE13, SV15, WLY+15, ZVZ15, ZVL15, CB14, LW13] authors considered fixed number of resources to the whole life time of the workflow application. But in our work, resources can be acquired at any time and released when they are idle, which saves the total charged cost. Further, another approaches such as in [ANE13, SV15, WLY+15, ZVZ15, ZVL15], did not consider the hourly charging billing model in the cost model or the data transfer time in total time reservation of the virtual machine, which affects the effectiveness of the algorithm. In cloud computing infrastructures, such as Amazon EC2 [1], the charging policy is based on an hour billing model even if the last reservation interval is not used. In this case, time fractions produced by previous tasks can be used by later tasks to save total renting cost. On the other hand, the workflow scheduling problem becomes more challenging when we consider multiple QoS parameters. Many algorithms have been proposed for multi-objective scheduling, but in most of them meta-heuristic methods or search-based strategies have been used to achieve good solutions. However, these methods based on meta-heuristics or search-based strategies usually need significantly high planning costs in terms of the time consumed to produce good results, which makes them less useful in real platforms that need to obtain map decisions on the fly.

In this chapter, a low-time complexity heuristic, named Deadline-Budget Workflow Scheduling (DBWS), is proposed to schedule workflow applications on cloud infrastructures constrained to two QoS parameters. In our model, the QoS parameters are time and cost. The objective of the proposed DBWS algorithm is to find a feasible schedule map that satisfies the user defined deadline and budget constraint values. To fulfill this objective, the proposed approach implements a mechanism to control the time and cost consumption of each task when producing a schedule solution. To the best of our knowledge, the algorithm proposed here is the first *low-time* complexity heuristic in cloud computing environment addressing two QoS parameters as constraints.

Workflow application model

Scientific workflow applications are commonly represented by a Directed Acyclic Graph (DAG), a directed graph with no cycles. Formally, a workflow application is a DAG represented by a triple $G =< T, E, data >$, where $T = \{t_1, t_2, \ldots, t_n\}$ is a finite set of tasks and n denotes the number of tasks in the workflow application. The set of edges E represent their data dependencies. A dependency ensures that a child node cannot be executed before all its parent tasks finish successfully and transfer the required input data. Let $data$ be a $n \times n$ matrix of communication data, where $data(t_i, t_j)$ is the file size required to be transmitted before task t_j execution from task t_i. The $\overline{C}_{(t_i \to t_j)}$ represents the average transfer time between the tasks t_i and t_j which is calculated based on the average bandwidth and latency among all resources pair. In a given DAG, a task with no predecessors is called an *entry task* and a task with no successors is called an *exit task*. We assume

[1] http://aws.amazon.com/ec2

that the DAG has exactly one entry task t_{entry} and one exit task t_{exit}. If a DAG has multiple entry or exit tasks, a dummy entry or exit task with zero weight and zero communication edges is added to the graph.

Cloud resource model

The target cloud computing platform is composed of a set of m heterogeneous resources $R = \{\cup_{j=1}^{m} r_j \mid r_j \in VM_{type}\}$, that provide services of different capabilities and costs. Each resource includes computation service, e.g., Amazon Elastic Cloud Compute (EC2)[2] and storage services, e.g., Amazon Elastic Block Store (EBS)[3], used as a local storage device for saving the input/output files. In this study, all computation and storage resources are assumed to be in same data center or region so that average bandwidth between computation resources is almost equal. Notice that the transfer time between two tasks being executed on the same VM is 0. Also, resources are offered in form of different type of virtual machines (VM_{type}). Each VM type has its own configuration for CPU capacity, memory size and an associated cost. Further, it is assumed that there is no limitation of the number of resources (VM) used by a workflow application and leasing a VM requires an initial boot time in order to be properly initialized and made available to the user; this time is not negligible and needs to be considered in the scheduling plan [MH12]. Similarly, on current commercial clouds, the pricing model is based on pay-as-go billing model for the number of *time interval* usage of a VM and it is specified by the cloud provider. The user will be charged for each complete *time interval* even if it does not completely use the time interval.

In this study, each resource r_j can be of any type as provided by Amazon EC2 (e.g. `m1.small`, `m1.large`, `m1.xlarge` and `c1.medium`). For a given resource r_j of a certain instance type, the average performance measured in GFLOPs and its price per hour of computation are known. The average performance in billions of floating point operations per second (GFLOP) of four different Amazon EC2 instance types thorough extensive benchmark experimentation are evaluated in [IOY+11]. In our model, we assume that each task is executed in any of these resources can benefit from a parallel execution using all the virtual cores exposed by the instance [DP14]. Also, according to Amazon cloud provider, users are charged based on the time interval of one hour ($interval\ time = 3600\,s$). Table.4.1 summarises the mean performance, the cost per hour of computation ($Cost_{r_j}$), and the ratio GFLOPs per invested dollar of these resources. Since all resource are located in the same data center or region, the internal data transfer cost is assumed to be zero.

Unlike most previous researches in this area, here, we present an array of release/acquire ($VM_{r/a}$) timestamp for each used VM resources which will be updated during the scheduling process. The array of timestamp $VM_{r/a} = \{(S_1, F_1), (S_2, F_2), \ldots\}$ where each pair (S, F) represents the Start and Finish time of consecutively execution of the target VM. These timestamps are calculated based on assigned tasks to the target VM.

[2] http://aws.amazon.com/EC2/
[3] http://aws.amazon.com/EBS/

Table 4.1: Performance and price of various Amazon EC2 instances

Instance type	Mean performance [GFLOPS]	Price[$/h]	GFLOPS/$
m1.small	2.0	0.1	19.6
m1.large	7.1	0.4	17.9
m1.xlarge	11.4	0.8	14.2
c1.medium	3.9	0.2	19.6

During the scheduling process and after making final decision of the appropriate resource (r_{sel}) for execution of the current task (t_{curr}), if the current task could not benefit from the last executed task on r_{sel} to reduce its execution cost, i.e., using the remaining last interval from the last previous scheduled task on r_{sel} , the $VM_{r/a}$ of resource r_{sel} is updated in the way that: a) add the release time after execution of the last scheduled task ; b) add the start (acquire) time according to the start time of t_{curr}. Otherwise, the release time of resource r_{sel} will be updated according to the finish time of the current task. Obviously, each resource can be rented for as many times and hours as required for finishing all the tasks scheduled on it. Additionally, we keep the set of scheduled tasks on each resource r_j denoted as $sched_{r_j} = \{t_i \mid AR(t_i) = r_j\}$, where $AR(t_i)$ represents the Assigned Resource on which the task t_i is scheduled to be executed. Each set $sched_{r_j}$ is sorted based on the finish time of its tasks.

Problem definition

The scheduling problem is defined as the finding a map between tasks and resources in order to meet the QoS parameters defined for each job. The problem here consists in finding a schedule map in such a way that the total execution time (makespan) and economical costs are constrained to user's defined values for time and cost. We describe next how the two measures are computed.

Makespan

For computing the total execution time (makespan) of a given workflow, it is necessary to defined the *Time Reservation* (TR) of execution for task t_i on resource r_j as the sum of the execution time of task t_i on resource r_j ($ET(t_i, r_j)$) and the time required for transferring the biggest input data from any parent of task $t_p \in pred(t_i)$. The information of task execution time (ET) can be gathered via benchmarking or via precise performance models based on existing estimation techniques (e.g. historical data [JWT+04] and analytical modelling [NKP+00]).

$$TR(t_i, r_j) = \max_{t_p \in pred(t_i)} \left\{ \overline{C}_{t_p \to t_i} \right\} + ET(t_i, r_j) \tag{4.1}$$

Considering the existence of data transfer time between tasks, for each task t_i to be executed in resource r_j, the resource r_j needs to be deployed before the task t_i starts transferring data from its parent and can be released after its execution is finished and data is transferred to its child. First,

we define $avail(r_j)$ as the the earliest start time of task t_i on resource r_j without considering its parents.

$$avail(r_j) = \begin{cases} 0 & , sched_{r_j} = \emptyset \\ \\ FT(t_l, r_j) & , sched_{r_j} \neq \emptyset \end{cases} \tag{4.2}$$

where t_l is the last task in sorted tasks scheduled list for resource r_j ($sched_{r_j}$). Based on $avail(r_j)$, we defined the Release Time of resource r_j ($R_T(r_j)$) as the last rental period of one hour ($interval\ time = 3600\,s$) for the last scheduled task on it. After that, resource r_j will be released if no other task starts executing on the resource.

$$R_T(r_j) = \left\lceil \frac{avail(r_j)}{interval\ time} \right\rceil \times interval\ time \tag{4.3}$$

Figure 4.1 shows a sample schedule generated for scheduled task t_1, t_2 and t_3 and current task t_4 which is selected to be scheduled. The Release Time of three resources, calculated by Eq. 4.3, are indicated as the last interval used by their last scheduled task. Note that, task t_4 is not scheduled and assigned to its target resource yet.

Next, the Start Time (ST) and Finish time (FT) of task t_i on each resource r_j are calculated as:

$$ST(t_i, r_j) = \max \left\{ \max_{t_p \in pred(t_i)} \{FT(t_p)\}, avail(r_j) \right\} \tag{4.4}$$

$$FT(t_i, r_j) = \lambda_{(t_i, r_j)} + ST(t_i, r_j) + TR(t_i, r_j) \tag{4.5}$$

where $\lambda_{(t_i, r_j)}$ is defined as required boot time for acquiring resource instance r_j. If task t_i can be started at last interval time for resource r_j, no boot time required to be considered for task's completion, otherwise, the target resource r_j need to be lunched and its boot time should be considered as a delay in task finish time. For example, in Figure 4.1, only if task t_4 wants to scheduled on resource VM_3, a boot time is required to be consider on its finish time because it starts after current release time of VM_3. The $\lambda_{(t_i, r_j)}$ is calculated by:

$$\lambda_{(t_i, r_j)} = \begin{cases} 0 & , ST(t_i, r_j) < R_T(r_j) \quad OR \quad sched_{r_j} = \emptyset \\ \\ boot\_time_{r_j} & , otherwise \end{cases} \tag{4.6}$$

where $boot\_time_{r_j}$ is the VM startup/boot time. In this study, we consider $boot\_time_{r_j} = 97\,s$ based on the measurements reported in [MH12] for the Amazon EC2 cloud. Please note that, during the resource selection phase for each task t_i, the $\lambda_{(t_i, r_j)}$ value is calculated according to the current situation of the target resources r_j, i.e. previous scheduled task on it.

The *makespan* or Schedule length is finally defined as the finish time of the last task of the workflow:

$$\text{DAG}_{makespan} = FT(t_{exit}) \tag{4.7}$$

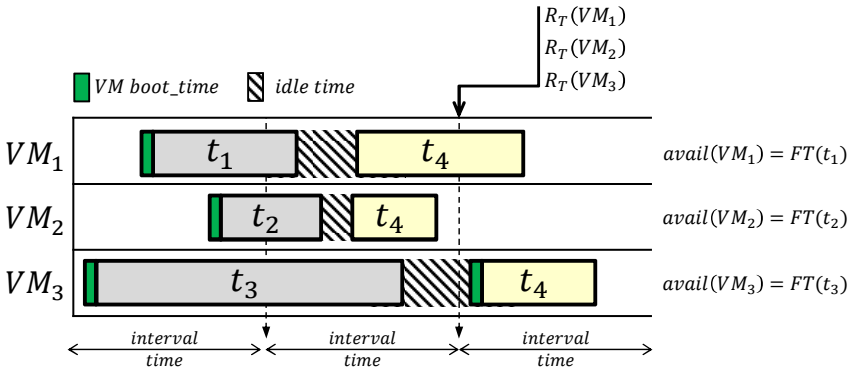

Figure 4.1: Example of a schedule. Tasks t_1, t_2 and t_3 are scheduled; and task t_4 is evaluated for scheduling

Financial cost

The financial cost of task t_i on resource r_j is calculated based on total usage time for complete task execution, considering data transfer time, execution time and resource usage price. In this research, we consider Amazon EC2 instance as the our platform which makes the hour price billing for each instance. In this model, if a task assigned to resource could not fully used the last hour interval, it will be charged for whole of it, i.e. partial hours are rounded up. As a consequence, if other tasks can be executed during that paid interval, they will not be charged for it. We define total usage time of task t_i as the payable period to be charged.

$$pay_{time}(t_i, r_j) = \begin{cases} FT(t_i, r_j) - ST(t_i, r_j) & , R_T(r_j) < ST(t_i, r_j) \\ 0 & , FT(t_i, r_j) < R_T(r_j) \\ FT(t_i, r_j) - R_T(r_j) & , otherwise \end{cases} \tag{4.8}$$

The *otherwise* condition will be applied if the task t_i starts before the current release time of resource r_j ($R_T(r_j)$) and finishes after it. So, in this case, the time slice before $R_T(r_j)$ is paid by previous tasks and should not be considered in the current usage time of task t_i.

The pay_{time} equal to zero means that the task can executed on previous paid interval (but not fully used) without any additional charge. For example, in Figure 4.1, if current task t_4 is scheduled on resource VM_2 the $pay_{time}(t_4, VM_2) = 0$. By considering Eq. 4.8, the pay time for resource VM_1 is equal to $pay_{time}(t_4, VM_1) = FT(t_4, VM_1) - R_T(VM_1)$ and for resource VM_3 we have $pay_{time}(t_4, VM_3) = FT(t_4, VM_3) - ST(t_4, VM_3)$. Please not that, the *boot_time* of resource VM_3 is already considered in $FT(t_4, VM_3)$ by Eq. 4.5.

The cost execution of task t_i for $pay_{time} > 0$ on resource r_j is computed by:

$$Cost(t_i, r_j) = \left\lceil \frac{pay_{time}(t_i, r_j)}{interval\ time} \right\rceil \times P_{r_j} \tag{4.9}$$

where P_{r_j} is the associated cost of resource r_j for each usage interval (Table. 4.1). Thus, the overall

cost for executing a workflow application is:

$$\mathrm{DAG}_{cost} = \sum_{t_i \in T} \left\{ Cost(t_i, r') \mid t_i \in sched_{r'} \right\} \tag{4.10}$$

Proposed Deadline–Budget workflow Scheduling (DBWS) algorithm

In this section, we present the Deadline-Budget Workflow Scheduling (DBWS) for cloud environments, which aims to find a feasible schedule within a budget and deadline constraints. The DBWS algorithm is a heuristic strategy that in a single step obtains a schedule that always accomplishes the deadline constraint and that may accomplish or not the budget constraint. If the cost constraint is met, we have a successful schedule, otherwise we have a failure and no schedule is produced. The algorithm is evaluated based on the success rate.

Before the description of the DBWS algorithm, next we present the attributes used in the algorithm.

- t_{curr} denotes the current task to be schedule, selected on the task selection phase among all ready tasks;
- r_{sel} denotes the target resource to execute t_{curr} on it;
- $FT_{min}(t_{curr})$ and $FT_{max}(t_{curr})$ denote the minimum and maximum finish time of current task among all tested resources;
- $\ell(t_i)$ denotes the level of task t_i; it is an integer value representing the maximum number of edges of the paths from the entry node to t_i. For the entry node, the level is $\ell(t_{entry}) = 1$, and for other tasks, it is given by:

$$\ell(t_i) = 1 + \max_{t_p \in pred(t_i)} \left\{ \ell(t_p) \right\} \tag{4.11}$$

- $Cost_{min}(t_{curr})$ and $Cost_{max}(t_{curr})$ denote the minimum and maximum execution cost of the current task among all tested resources;
- $Cost_{high}(DAG)$ and $Cost_{low}(DAG)$ represent the total execution cost for scheduling the target application workflow on the set of homogeneous VM with highest and lowest cost among all possible VM types in our platform. Here, we use the HEFT[THW02] algorithm to schedule a workflow application.

The DBWS algorithm consists of two phases, namely a *task selection* phase and a *resource selection* phase as described next.

Task selection

Tasks are selected according to their priorities. To assign a priority to a task in the DAG, the upward rank (*rank_u*) [THW02] is computed. This rank represents, for a task t_i, the length of the

longest path from task t_i to the exit node(t_{exit}), including the computational time of t_i, and it is given by Eq. 4.12:

$$rank_u(t_i) = \overline{ET}(t_i) + \max_{t_{child} \in succ(t_i)} \left\{ \overline{C}_{t_i \to t_{child}} + rank_u(t_{child}) \right\} \tag{4.12}$$

where $\overline{ET}(t_i)$ is the average execution time of task t_i over all resources, $\overline{C}_{t_i \to t_{child}}$ is the average communication time between two tasks t_i and t_{child}, and $succ(t_i)$ are the set of immediate successor tasks of task t_i. To prioritize tasks it is common to consider average values because they have to be prioritize before knowing the location where they will run. For the exit node, $rank_u(t_{exit}) = \overline{ET}(t_{exit})$.

Resource Selection

The target VM to be selected to execute the current task is guided by the following quantities related to cost and time. To select the best suitable resource, a trade-off between these two variables, time and cost, is evaluated. We define a variable S_{DL} as the time limit. S_{DL} is defined as the sub-deadline that is assigned to each task based on the total application deadline. First, all tasks are divided in different levels based on their depth in the graph. We defined level execution ($Level_{exe}$) as the maximum execution length of all tasks in corresponding level and is given by:

$$Level^j_{exe} = \max_{\substack{t_i \in T \\ \ell(t_i) == j}} \left\{ ET_{max}(t_i) + \max_{t_p \in pred(t_i)} \{\overline{C}_{t_p \to t_i})\} \right\} \tag{4.13}$$

where $ET_{max}(t_i)$ represents the maximum execution time for task t_i among all VM_{type}. In the next step, we distribute the user deadline (D_{user}) among all levels. The sub-deadline value for level j ($Level^j_{DL}$) is computed recursively by traversing the task graph downwards, starting from the first level, as shown bellow:

$$Level^j_{DL} = Level^{j-1}_{DL} + D_{user} \times \frac{Lvel^j_{exe}}{\sum_{1 \le j' \le \ell(t_{exit})} Level^{j'}_{exe}} \tag{4.14}$$

For the first level ($Level^1_{DL}$), the first part of Eq.(4.14) is considered zero. Finally, all tasks belonging to the same level have the same sub-deadline.

$$S_{DL}(t_{curr}) = \left\{ Level^j_{DL} \mid \ell(t_i) == j \right\} \tag{4.15}$$

Note that, the task's sub-deadline is a soft limit as in most deadline distribution strategies; if the scheduler cannot find a resource (VM) that satisfies the sub-deadline for the current task, the resource that can finish the current task at the earliest time may is selected.

The resource selection phase is based on the combination of the two QoS factors, time and cost, in order to obtain the best balance between time and cost minimum values. We define two relative quantities, namely, Time Quality ($Time_Q$) and Cost Quality ($Cost_Q$), for current task t_{curr}

on each resource $r_j \in R \cup R'$, where R represents the set of resources (VM instances) used in the previous steps of scheduling, and R' is defined as the set of one temporary resource from each available VM_{type}. At each step after selecting the suitable resource r_{sel} for task t_{curr}, R is updated by $R = \{R \cup r_{sel} \mid r_{sel} \notin R\}$.

Both time and cost quantities are shown in (Eq. 4.16) and (Eq. 4.17), respectively. Both quantities are normalized by their maximum values.

$$Time_Q(t_{curr}, r_j) = \frac{\xi \times S_{DL}(t_{curr}) - FT(t_{curr}, r_j)}{FT_{max}(t_{curr}) - FT_{min}(t_{curr})} \tag{4.16}$$

$$Cost_Q(t_{curr}, r_j) = \frac{Cost_{max}(t_{curr}) - Cost(t_{curr}, r_j)}{Cost_{max}(t_{curr}) - Cost_{min}(t_{curr})} \times \xi \tag{4.17}$$

where

$$\xi = \begin{cases} 1 & \text{if} \quad FT(t_{curr}, r_j) < S_{DL}(t_{curr}) \\ 0 & otherwise \end{cases} \tag{4.18}$$

$Time_Q$ measures how much closer to the task sub-deadline (S_{DL}) the finish time of current task on resource r_j is. The sub-deadline is defined as the maximum allowance of task completion time. Consequently, resources with higher $Time_Q$ values, i.e. larger distance between finish time and sub-deadline, have higher possibility to be selected. If the current task has higher finish time on resource r_j than its sub-deadline, $Time_Q$ assumes a negative value for r_j, reducing the possibility for this resource to be selected. Similarly, $Cost_Q$ measures the difference between the actual cost on resource r_j and the maximum execution cost.

In the case that none of the resources can guarantee the current task sub-deadline ($S_{DL}(t_{curr})$), $Cost_Q$ is zero for all of them, and $Time_Q$ for each resource r_j is a negative value that represents the relative finish time obtained with r_j, i.e., the lower finish time causes to the lower negative value. And, the resource with higher $Time_Q$, i.e., close to zero, would be selected.

Finally, to select the most suitable resource for current task, the Quality measure (Q) for each resource r_j is computed as shown in Eq(4.19):

$$Q(t_{curr}, r_j) \;=\; Time_Q(t_{curr}, r_j) \times (1 - C_F) \;+\; Cost_Q(t_{curr}, r_j) \times C_F \tag{4.19}$$

where Cost-efficient Factor (C_F) is the tradeoff factor and defined as:

$$C_F = \frac{Cost_{low}(DAG)}{B_{user}} \tag{4.20}$$

Both time quantity ($Time_Q$) and cost quantity ($Cost_Q$) parameters are weighted by the ratio of the cheapest cost execution of the whole workflow application ($Cost_{low}(DAG)$) over the user defined available budget (B_{user}) so that the effectiveness of both time and cost factors can be controlled. A lower value of C_F means that the user prefers to pay more to execute the application

faster, so that the time quality ($Time_Q$) is more predominant in the resource Quality measure (Q). In the same way, a higher value of C_F means that the user available budget is close to the cheapest possible execution cost of the workflow so that the time quality ($Time_Q$) is inconspicuous while the cost quality ($Cost_Q$) becomes more influential, allowing the selection of more cheap resources that guarantee a lower execution cost for t_{curr}.

The DBWS algorithm is shown in Algorithm 1. After some initializations in lines 1-2, first, the possibility of finding a schedule map under a user defined budget is checked in line 3. Then, the sub-deadline value for each task is computed according Eq. 4.15 in line 8. The DBWS algorithm starts to map all tasks of the application (while looping in lines 9–18). At each step, on line 10, among all ready tasks, the task with highest priority ($rank_u$) is selected as the current task (t_{curr}). Then, in lines 11–13, the Quality measure for assigning t_{curr} to the resource r_j ($Q(t_{curr}, r_j)$) is calculated. Note that, first, the finish time (FT) and execution cost of the current task is calculated and then the quality measure for all resources is calculated. Next, the resource (r_{sel}) with the highest quality measure among all resources is selected (line 14). Finally, after assigning the current task to the resource, the release/acquire timestamp for the resource r_{sel} is updated as explained in the*resource model* in section 4.3.

Algorithm 1 DBWS algorithm

Require: a DAG and user's QoS Parameters values for Deadline (D_{user}) and Budget (B_{user})
1: Compute and sort all tasks based on their upward rank ($rank_u$) value
2: Calculate HEFT schedule cost on the resources with cheapest ($Cost_{low}$) and most expensive ($Cost_{high}$) cost from VM_{type}
3: **if** $B_{user} < Cost_{low}(DAG)$ **then**
4: **return** no possible schedule map
5: **else if** $B_{user} > Cost_{high}(DAG)$ **then**
6: **return** HEFT scheduled map on most expensive VM_{type}
7: **end if**
8: Compute the Sub-DeadLine value (S_{DL}) for each task
9: **while** there is an unscheduled task **do**
10: t_{curr} = the next ready task with highest $rank_u$ value
11: **for all** $r_j \in R \cup R'$ **do**
12: Calculate Quality measure $Q(t_{curr}, r_j)$ using Eq.4.19
13: **end for**
14: r_{sel} = resource r_j with highest Quality measure (Q)
15: Assign current task t_{curr} to resource r_{sel}
16: Update $R = \{R \cup r_{sel} \mid r_{sel} \notin R\}$
17: Update $VM_{r/a}(r_{sel})$
18: **end while**
19: **return** Schedule Map

In terms of time complexity, DBWS requires the computation of the upward rank ($rank_u$) and Sub-DeadLines (S_{DL}) for each task that have complexity $O(n.p)$, where p is the number of available resources and n is the number of tasks in the workflow application. In the resource selection phase, to find and assign a suitable resource for the current task, the complexity is $O(n.p)$

for calculating ST and FT for the current task among all resources, plus $O(p)$ for calculating the Quality measure. The total time is $O(n.p + n(n.p + p))$, where the total algorithm complexity is of the order $O(n^2.p)$.

Experimental results

This section presents performance comparisons of the DBWS algorithm with four most recently published algorithms, Hybrid [SLH+13], MTCT (Min-min based time and cost tradeoff) [XYQA16], CwFT (Cost with Finish Time-based) [MH13a] scheduling algorithms and SABA (Security-Aware and Budget-Aware) [ZVL15]. We choose MTCT [XYQA16] for comparison because it outperforms LOSS algorithms [SZTD07] and IC-PCP [ANE13].

Budget and deadline parameters

To evaluate the DBWS algorithm, in our experiment, we need to define a value for time and cost as Deadline and Budget constraint parameters. To specify these parameters, we need to defined boundary values for each of them. By using HEFT scheduling algorithm, we calculate the total execution time (makespan) of the workflow scheduled on the set of homogeneous VM instance with highest and lowest associated cost as the minimum (min_D) and the maximum (max_D) deadline boundary value. In the same way, the corresponding execution costs are the maximum (max_B) and minimum (min_B) execution cost of the workflow application. With these highest and lowest bound values, we define for the current application a unique Deadline and Budget constraint, as described by Eq. 4.21 and Eq. 4.22:

$$D_{user} = min_D + \alpha_D \times (max_D - min_D) \tag{4.21}$$

$$B_{user} = min_B + \alpha_B \times (max_B - min_B) \tag{4.22}$$

where the deadline parameter α_D and budget parameter α_B can be selected in the range of $[0 \dots 1]$. In this thesis, to observe the ability of finding valid schedule maps, we selected a low set of values, $\{0.1, 0.3, 0.5\}$, for time and cost parameters (α_D and α_B) to test the performance of each algorithm on harder conditions. Undoubtedly, increasing values for α_D and α_B, we would be able to achieve higher successful percentage rates.

Performance metric

To evaluate and compare our algorithm with other approaches, we consider the Planning Successful Rate (PSR), as expressed by Eq. 4.23. This metric provides the percentage of valid schedules obtained in a given experiment.

$$\text{PSR} = 100 \times \frac{Successful\ Planning}{Total\ Number\ in\ experiment} \tag{4.23}$$

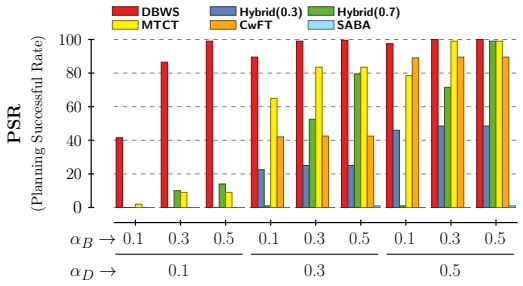

Figure 4.2: PSR value for CYBERSHAKE

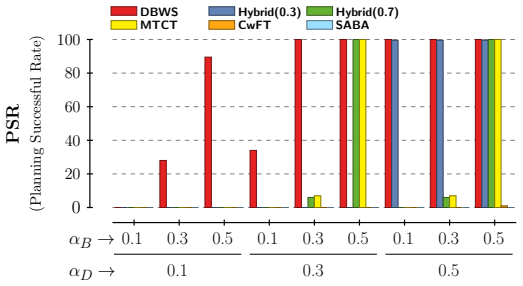

Figure 4.3: PSR value for EPIGENOMIC

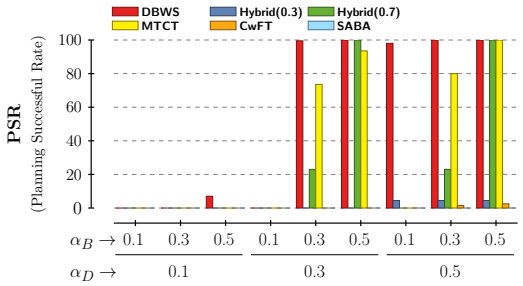

Figure 4.4: PSR value for LIGO

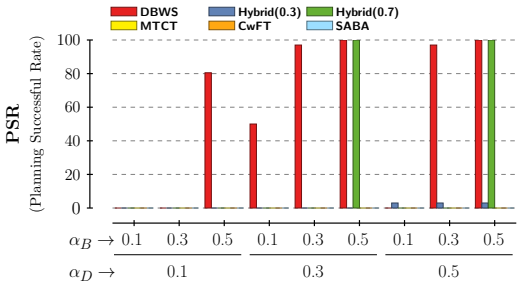

Figure 4.5: PSR value for MONTAGE

In addition, to investigate the quality of results, we normalized the achieved scheduling makespan and execution cost by each algorithm, namely *NM* (Normalized makespan) and *NB* (Normalized Budget) as described in Eq. 4.24 and Eq. 4.25:

$$\text{NM} = \frac{D_{user}}{\text{DAG}_{makespan}} \tag{4.24}$$

$$\text{NB} = \frac{B_{user}}{\text{DAG}_{cost}} \tag{4.25}$$

Note that, both metrics *NM* and *NB* are calculated for each schedule map, even for not successful ones, achieved by the algorithm. Basically, lower value than 1 for *NM* and *NB* metrics means that the schedule map could not meet the constraint values for time and cost, respectively.

Results and discussion

To evaluate the algorithms on a standard and real set of workflow applications, a set of workflows were generated using the code developed in Pegasus toolkit[4]. Four well known structures were chosen [JCD+13], namely: CYBERSHAKE, EPIGENOMIC, LIGO and MONTAGE. The workflows that we use are characterized as CPU-bound (EPIGENOMIC), I/O-bound (MONTAGE), data-intensive (CYBERSHAKE), workflows with large memory requirements (CYBERSHAKE, LIGO), and workflows with large resource requirements (CYBERSHAKE, LIGO). For each type of these real world applications workflows, we generated 1000 DAGs with a number of tasks equal to 50, 300 and 1000 tasks.

[4]https://confluence.pegasus.isi.edu/display/pegasus/WorkflowGenerator

The original implementation of the compared scheduling algorithms assumed a fixed number of resources during the schedule map. In our implementation of those algorithms, we assigned an initialized fixed number of resources equal to the maximum number of concurrent tasks among all levels in the workflow application. Also, to apply the cost consumption during the scheduling process, we consider the same approach to calculated the cost execution of each task (Eq.4.9) for all algorithms. Also, for the Hybrid [SLH+13] scheduling algorithm, we consider two version, namely Hybrid ($\alpha = 0.3$) and Hybrid ($\alpha = 0.7$), where the α parameter represents the user's preference for the execution time and the monetary cost, i.e lower α corresponds to less monetary cost.

Figures 4.2-4.5 shows the average Planning Successful Rate (PSR) obtained for different real workflow application considered here. The main result is that the algorithm DBWS obtains good performance in comparison to other state-of-the-art heuristic-based algorithms, for the range of budget and deadline values considered here. By increasing budget factor (α_B), PSR values obtained by DBWS are improved.

Figures 4.6 and 4.7 represent the normalized makespan and cost, obtained by each algorithm. In other to have a better presentation, *NM* and *NB* values are divided into two main categories, where *safe* is represented by the green color and *unsafe* is represented by the red color. A value greater than 1 for the *NM* metric (Eq.4.24) means that the algorithm could obtain a schedule makespan lower than the user defined deadline. But a lower value $NM < 1$ means that the algorithm failed to find the schedule map with a makepsan lower than the user defined deadline. The same explanation can be considered for *NB* metric (Eq.4.25). As it is shown in Figure 4.6, the makespan of the schedule map obtained by proposed DBWS algorithm always meet the user defined deadline constraint value for all range of budget factor α_B. However, for the total cost execution in Figure 4.7, by decreasing the time factor α_D, the execution time of schedule map obtained by DBWS becomes higher than the user defined budget. Note that the PSR values represented in Figures 4.2-4.5 are calculated based on meeting both time and cost constraint value by the schedule map. For example, despite of the best reduction in execution cost by CwFT scheduling algorithm in Figures 4.2-4.5, due to failing in meeting the deadline value, the CwFT approach shows the worst performance in terms of PSR value (Figure 4.2-4.5). Also, for SABA scheduling algorithm, it fails in most cases as shown by the PSR metric. The reason can be explained due to the strategy used for VM assignment, namely Comparative Factor (CF). The CF approach used the trade-off balance between time and cost factors and did not control the cost consumption during the scheduling process. As seen from Figures 4.6 and 4.7, SABA shows the best performance in total execution time reduction and worst one for total cost.

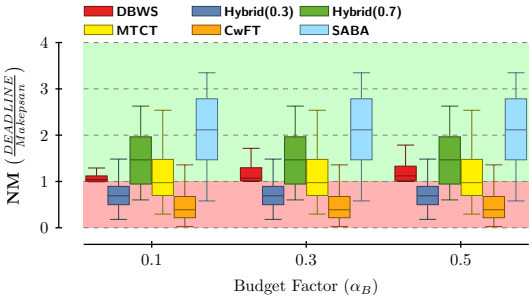

Figure 4.6: Normalized Makespan

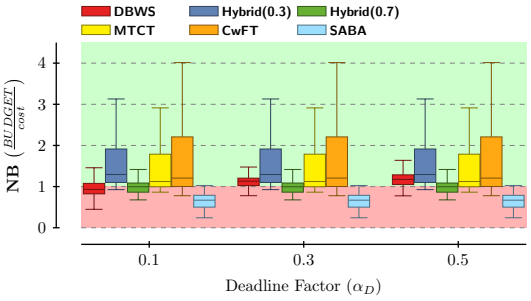

Figure 4.7: Normalized Budget

Chapter 5

Summary and Conclusion

In this section, I summarized the main contributions achieved in this thesis.

Summary of Contribution

This thesis focus on a scheduling algorithm for workflow applications on cloud computing platforms based on user's QoS requirements. The two major QoS parameters considered in this thesis are cost and time.

The contributions of this thesis are:

- a review of multiple QoS parameter workflow scheduling on cloud computing environment;
- a new heuristic algorithm with quadratic complexity for workflow application scheduling, constrained to time and cost;
- extensive evaluation with results for real-world applications.

Conclusions

In this thesis, we presente the Deadline-Budget Workflow Scheduling (DBWS) for cloud environments, which maps a workflow application to cloud resources constrained to user-defined deadline and budget values. The algorithm was compared with other state-of-the-art heuristic-based scheduling algorithms. In terms of time complexity, which is a critical factor for effective usage on real platforms, our algorithm has quadratic time complexity similarly to the state-of-the-art heuristics. In terms of the quality of results, DBWS achieves better rates of successful schedules compared to other heuristic-based approaches for tested real work workflow applications. For the range values of deadline and budget constraints considered in this work, DBWS shows a significant improvement of the successful planning rate for the workflows and cloud platform considered.

In conclusion, we have presented the DBWS algorithm for budget and deadline constrained scheduling, which has proved to achieve better performance than other heuristic-based approaches, namely Hybrid [SLH+13] MTCT [XYQA16] and CwFT [MH13a].

Directions for Future Research

In future work, we intend to compare the proposed approach with search-based or meta-heuristic strategies which may improve the quality of results by performing a higher number of iterations on the solution search space. Also, we intend to extend the algorithm to consider dynamic concurrent workflow applications which can be submitted by any user at any time.

References

[ADC05] Tim Abels, Puneet Dhawan, and Balasubramanian Chandrasekaran. An overview of xen virtualization. *Dell Power Solutions*, 8:109–111, 2005.

[AGM+90] Stephen F Altschul, Warren Gish, Webb Miller, Eugene W Myers, and David J Lipman. Basic local alignment search tool. *Journal of molecular biology*, 215(3):403–410, 1990.

[ANE12] Saeid Abrishami, Mahmoud Naghibzadeh, and Dick HJ Epema. Cost-driven scheduling of grid workflows using partial critical paths. *IEEE Transactions on Parallel and Distributed Systems*, 23(8):1400–1414, 2012.

[ANE13] Saeid Abrishami, Mahmoud Naghibzadeh, and Dick HJ Epema. Deadline-constrained workflow scheduling algorithms for infrastructure as a service clouds. *Future Generation Computer Systems*, 29(1):158–169, 2013.

[BDF+03] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. Xen and the art of virtualization. In *ACM SIGOPS Operating Systems Review*, volume 37, pages 164–177. ACM, 2003.

[BJJ10] Sushil Bhardwaj, Leena Jain, and Sandeep Jain. Cloud computing: A study of infrastructure as a service (iaas). *International Journal of engineering and information Technology*, 2(1):60–63, 2010.

[BM07] Luiz F Bittencourt and Edmundo RM Madeira. Fulfilling task dependence gaps for workflow scheduling on grids. In *Signal-Image Technologies and Internet-Based System, 2007. SITIS'07. Third International IEEE Conference on*, pages 468–475. IEEE, 2007.

[BM10] Luiz Fernando Bittencourt and Edmundo RM Madeira. Towards the scheduling of multiple workflows on computational grids. *Journal of grid computing*, 8(3):419–441, 2010.

[BM11] Luiz Fernando Bittencourt and Edmundo Roberto Mauro Madeira. Hcoc: a cost optimization algorithm for workflow scheduling in hybrid clouds. *Journal of Internet Services and Applications*, 2(3):207–227, 2011.

[Boc14] Klavdiya Bochenina. A comparative study of scheduling algorithms for the multiple deadline-constrained workflows in heterogeneous computing systems with time windows. *Procedia Computer Science*, 29:509–522, 2014.

[BYO+12] Kahina Bessai, Samir Youcef, Ammar Oulamara, Claude Godart, and Selmin Nurcan. Bi-criteria workflow tasks allocation and scheduling in cloud computing environments. In *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*, pages 638–645. IEEE, 2012.

[BYOG13] Kahina Bessai, Samir Youcef, Ammar Oulamara, and Claude Godart. Bi-criteria strategies for business processes scheduling in cloud environments with fairness metrics. In *IEEE 7th International Conference on Research Challenges in Information Science (RCIS)*, pages 1–10. IEEE, 2013.

[BYV+09] Rajkumar Buyya, Chee Shin Yeo, Srikumar Venugopal, James Broberg, and Ivona Brandic. Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation computer systems*, 25(6):599–616, 2009.

[CB14] Rodrigo N Calheiros and Rajkumar Buyya. Meeting deadlines of scientific workflows in public clouds with tasks replication. *Parallel and Distributed Systems, IEEE Transactions on*, 25(7):1787–1796, 2014.

[CLCG13] Zhicheng Cai, Xiaoping Li, Long Chen, and Jatinder ND Gupta. Bi-direction adjust heuristic for workflow scheduling in clouds. In *Parallel and Distributed Systems (ICPADS), 2013 International Conference on*, pages 94–101. IEEE, 2013.

[CLG16] Zhicheng Cai, Xiaoping Li, and Jatinder ND Gupta. Heuristics for provisioning services to workflows in xaas clouds. *IEEE Transactions on Services Computing*, 9(2):250–263, 2016.

[CS13] Nitish Chopra and Sarbjeet Singh. Deadline and cost based workflow scheduling in hybrid cloud. In *Advances in Computing, Communications and Informatics (ICACCI), 2013 International Conference on*, pages 840–846. IEEE, 2013.

[CWG09] Qi Cao, Zhi-Bo Wei, and Wen-Mao Gong. An optimized algorithm for task scheduling based on activity based costing in cloud computing. In *2009 3rd International Conference on Bioinformatics and Biomedical Engineering*, pages 1–3. IEEE, 2009.

[CZ12] Wei-Neng Chen and Jun Zhang. A set-based discrete pso for cloud workflow scheduling with user-defined qos constraints. In *Systems, Man, and Cybernetics (SMC), 2012 IEEE International Conference on*, pages 773–778. IEEE, 2012.

[DP14] Juan J Durillo and Radu Prodan. Multi-objective workflow scheduling in amazon ec2. *Cluster Computing*, 17(2):169–189, 2014.

[FFP13] Hamid Mohammadi Fard, Thomas Fahringer, and Radu Prodan. Budget-constrained resource provisioning for scientific applications in clouds. In *Cloud Computing Technology and Science (CloudCom), 2013 IEEE 5th International Conference on*, volume 1, pages 315–322. IEEE, 2013.

[GB12] Saurabh Kumar Garg and Rajkumar Buyya. Green cloud computing and environmental sustainability. *Harnessing Green IT: Principles and Practices*, pages 315–340, 2012.

[HKA+11] Tram Truong Huu, Guilherme Koslovski, Fabienne Anhalt, Johan Montagnat, and Pascale Vicat-Blanc Primet. Joint elastic cloud and virtual network framework for application performance-cost optimization. *Journal of Grid Computing*, 9(1):27–47, 2011.

[IOY+11] Alexandru Iosup, Simon Ostermann, M Nezih Yigitbasi, Radu Prodan, Thomas Fahringer, and Dick HJ Epema. Performance analysis of cloud computing services for many-tasks scientific computing. *Parallel and Distributed Systems, IEEE Transactions on*, 22(6):931–945, 2011.

[JCD+13] Gideon Juve, Ann Chervenak, Ewa Deelman, Shishir Bharathi, Gaurang Mehta, and Karan Vahi. Characterizing and profiling scientific workflows. *Future Generation Computer Systems*, 29(3):682–692, 2013.

[JHC+11] He-Jhan Jiang, Kuo-Chan Huang, Hsi-Ya Chang, Di-Syuan Gu, and Po-Jen Shih. Scheduling concurrent workflows in hpc cloud through exploiting schedule gaps. In *Algorithms and Architectures for Parallel Processing*, pages 282–293. Springer, 2011.

[JWT+04] S Jang, Xingfu Wu, Valerie Taylor, Gaurang Mehta, Karan Vahi, and Ewa Deelman. Using performance prediction to allocate grid resources. Technical report, Technical Report 2004-25, GriPhyN Project, USA, 2004.

[KKYC14] Dong-Ki Kang, Seong-Hwan Kim, Chan-Hyun Youn, and Min Chen. Cost adaptive workflow scheduling in cloud computing. In *Proceedings of the 8th International conference on ubiquitous information management and communication*, page 65. ACM, 2014.

[LQM+12] Jiayin Li, Meikang Qiu, Zhong Ming, Gang Quan, Xiao Qin, and Zonghua Gu. Online optimization for scheduling preemptable tasks on iaas cloud systems. *Journal of Parallel and Distributed Computing*, 72(5):666–677, 2012.

[LSC+11] Jian Li, Sen Su, Xiang Cheng, Qingjia Huang, and Zhongbao Zhang. Cost-conscious scheduling for large graph processing in the cloud. In *High Performance Computing and Communications (HPCC), 2011 IEEE 13th International Conference on*, pages 808–813. IEEE, 2011.

[LW13] Xiangyu Lin and Chase Qishi Wu. On scientific workflow scheduling in clouds under budget constraint. In *Parallel Processing (ICPP), 2013 42nd International Conference on*, pages 90–99. IEEE, 2013.

[LZ13] Young Choon Lee and Albert Y Zomaya. Stretch out and compact: Workflow scheduling with resource abundance. In *Cluster, Cloud and Grid Computing (CCGrid), 2013 13th IEEE/ACM International Symposium on*, pages 219–226. IEEE, 2013.

[LZW+12] Wenjuan Li, Qifei Zhang, Jiyi Wu, Jing Li, and Haili Zhao. Trust-based and qos demand clustering analysis customizable cloud workflow scheduling strategies. In *Cluster Computing Workshops (CLUSTER WORKSHOPS), 2012 IEEE International Conference on*, pages 111–119. IEEE, 2012.

[MFB+15] Maciej Malawski, Kamil Figiela, Marian Bubak, Ewa Deelman, and Jarek Nabrzyski. Scheduling multilevel deadline-constrained scientific workflows on clouds based on cost optimization. *Scientific Programming*, 2015:5, 2015.

[MG11] Peter Mell and Tim Grance. The nist definition of cloud computing. *Computer Security Division, Information Technology Laboratory, National Institute of Standards and Technology Gaithersburg*, 2011.

[MH11] Ming Mao and Marty Humphrey. Auto-scaling to minimize cost and meet application deadlines in cloud workflows. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, page 49. ACM, 2011.

[MH12] Ming Mao and Marty Humphrey. A performance study on the vm startup time in the cloud. In *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*, pages 423–430. IEEE, 2012.

[MH13a] Nguyen Doan Man and Eui-Nam Huh. Cost and efficiency-based scheduling on a general framework combining between cloud computing and local thick clients. In *Computing, Management and Telecommunications (ComManTel), 2013 International Conference on*, pages 258–263. IEEE, 2013.

[MH13b] Ming Mao and Marty 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*, pages 67–78. IEEE, 2013.

[NKP+00] Graham R Nudd, Darren J Kerbyson, Efstathios Papaefstathiou, Stewart C Perry, John S Harper, and Daniel V Wilcox. Pace—a toolset for the performance prediction of parallel and distributed systems. *International Journal of High Performance Computing Applications*, 14(3):228–251, 2000.

[PBC+03] Vijay S Pande, Ian Baker, Jarrod Chapman, Sidney P Elmer, Siraj Khaliq, Stefan M Larson, Young Min Rhee, Michael R Shirts, Christopher D Snow, Eric J Sorin, et al. Atomistic protein folding simulations on the submillisecond time scale using worldwide distributed computing. *Biopolymers*, 68(1):91–109, 2003.

[PGB+14] Deepak Poola, Saurabh Kumar Garg, Rajkumar Buyya, Yun Yang, and Kotagiri Ramamohanarao. Robust scheduling of scientific workflows with deadline and budget constraints in clouds. In *2014 IEEE 28th International Conference on Advanced Information Networking and Applications*, pages 858–865. IEEE, 2014.

[RLP11] Mustafizur Rahman, Xiaorong Li, and Henry Palit. Hybrid heuristic for scheduling data analytics workflow applications in hybrid cloud environment. In *Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW), 2011 IEEE International Symposium on*, pages 966–974. IEEE, 2011.

[SCB02] Shava Smallen, Henri Casanova, and Francine Berman. Applying scheduling and tuning to on-line parallel tomography. *Scientific Programming*, 10(4):271–289, 2002.

[SLD+16] Jiyuan Shi, Junzhou Luo, Fang Dong, Jinghui Zhang, and Junxue Zhang. Elastic resource provisioning for scientific workflow scheduling in cloud under budget and deadline constraints. *Cluster Computing*, 19(1):167–182, 2016.

[SLH+13] Sen Su, Jian Li, Qingjia Huang, Xiao Huang, Kai Shuang, and Jie Wang. Cost-efficient task scheduling for executing large programs in the cloud. *Parallel Computing*, 39(4):177–188, 2013.

[SS10] S Selvarani and G Sudha Sadhasivam. Improved cost-based algorithm for task scheduling in cloud computing. In *Computational intelligence and computing research (iccic), 2010 ieee international conference on*, pages 1–5. IEEE, 2010.

[STZN13] Shaghayegh Sharif, Javid Taheri, Albert Y Zomaya, and Surya Nepal. Mphc: Preserving privacy for workflow execution in hybrid clouds. In *Parallel and Distributed Computing, Applications and Technologies (PDCAT), 2013 International Conference on*, pages 272–280. IEEE, 2013.

[STZN14] Shaghayegh Sharif, Javid Taheri, Albert Y Zomaya, and Surya Nepal. Online multiple workflow scheduling under privacy and deadline in hybrid cloud environment. In *Cloud Computing Technology and Science (CloudCom), 2014 IEEE 6th International Conference on*, pages 455–462. IEEE, 2014.

[SV15] Jyoti Sahni and Deo Vidyarthi. A cost-effective deadline-constrained dynamic scheduling algorithm for scientific workflows in a cloud environment. *IEEE Transactions on Cloud Computing*, PP(99):1–1, 2015.

[SZTD07] Rizos Sakellariou, Henan Zhao, Eleni Tsiakkouri, and Marios D Dikaiakos. Scheduling workflows with budget constraints. In *Integrated research in GRID computing*, pages 189–202. Springer, 2007.

[THW02] Haluk Topcuoglu, Salim Hariri, and Min-you Wu. Performance-effective and low-complexity task scheduling for heterogeneous computing. *Parallel and Distributed Systems, IEEE Transactions on*, 13(3):260–274, 2002.

[VK14a] Amandeep Verma and Sakshi Kaushal. Bi-criteria priority based particle swarm optimization workflow scheduling algorithm for cloud. In *Engineering and Computational Sciences (RAECS), 2014 Recent Advances in*, pages 1–6. IEEE, 2014.

[VK14b] Amandeep Verma and Sakshi Kaushal. Deadline constraint heuristic-based genetic algorithm for workflow scheduling in cloud. *International Journal of Grid and Utility Computing*, 5(2):96–106, 2014.

[WC16] Chase Q Wu and Huiyan Cao. Optimizing the performance of big data workflows in multi-cloud environments under budget constraint. In *Services Computing (SCC), 2016 IEEE International Conference on*, pages 138–145. IEEE, 2016.

[WHLR16] Hao Wu, Xiayu Hua, Zheng Li, and Shangping Ren. Resource and instance hour minimization for deadline constrained dag applications using computer clouds. *IEEE Transactions on Parallel and Distributed Systems*, 27(3):885–899, 2016.

[WLN+13] Zhangjun Wu, Xiao Liu, Zhiwei Ni, Dong Yuan, and Yun Yang. A market-oriented hierarchical scheduling strategy in cloud workflow systems. *The Journal of Supercomputing*, 63(1):256–293, 2013.

[WLY+15] Chase Qishi Wu, Xiangyu Lin, Dantong Yu, Wei Xu, and Li Li. End-to-end delay minimization for scientific workflows in clouds under budget constraint. *Cloud Computing, IEEE Transactions on*, 3(2):169–181, 2015.

[XCWB09] Meng Xu, Lizhen Cui, Haiyang Wang, and Yanbing Bi. A multiple qos constrained scheduling strategy of multiple workflows for cloud computing. In *2009 IEEE International Symposium on Parallel and Distributed Processing with Applications*, pages 629–634. IEEE, 2009.

[XPF13] Gaochao Xu, Junjie Pang, and Xiaodong Fu. A load balancing model based on cloud partitioning for the public cloud. *Tsinghua Science and Technology*, 18(1):34–39, 2013.

[XYQA16] Heyang Xu, Bo Yang, Weiwei Qi, and Emmanuel Ahene. A multi-objective optimization approach to workflow scheduling in clouds considering fault recovery. *KSII Transactions on Internet and Information Systems*, 10(3):976–995, 2016.

[YCKG13] Sonia Yassa, Rachid Chelouah, Hubert Kadima, and Bertrand Granado. Multi-objective approach for energy-aware workflow scheduling in cloud computing environments. *The Scientific World Journal*, 2013, 2013.

[ZH14] Amelie Chi Zhou and Bingsheng He. Transformation-based monetary costoptimizations for workflows in the cloud. *IEEE Transactions on Cloud Computing*, 2(1):85–98, 2014.

[ZHL16] Amelie Chi Zhou, Bingsheng He, and Cheng Liu. Monetary cost optimizations for hosting workflow-as-a-service in iaas clouds. *IEEE Transactions on Cloud Computing*, 4(1):34–48, 2016.

[ZVL12] Lingfang Zeng, Bharadwaj Veeravalli, and Xiaorong Li. Scalestar: Budget conscious scheduling precedence-constrained many-task workflow applications in cloud. In *2012 IEEE 26th International Conference on Advanced Information Networking and Applications*, pages 534–541. IEEE, 2012.

[ZVL15] Lingfang Zeng, Bharadwaj Veeravalli, and Xiaorong Li. Saba: A security-aware and budget-aware workflow scheduling strategy in clouds. *Journal of Parallel and Distributed Computing*, 75:141–151, 2015.

[ZVZ15] Lingfang Zeng, Bharadwaj Veeravalli, and Albert Y Zomaya. An integrated task computation and data management scheduling strategy for workflow applications in cloud environments. *Journal of Network and Computer Applications*, 50:39–48, 2015.