THE UNIVERSITY OF
SYDNEY

Doctoral Thesis

A Framework for Approximate Optimization of BoT Application Deployment in Hybrid Cloud Environment

Author:
MohammadReza
HoseinyFarahabady

Supervisor:
Prof. Albert Y. Zomaya

A thesis submitted in fulfillment of the requirements
for the degree of ***Doctor of Philosophy***

in the

Centre for Distributed & High Performance Computing
School of Information Technologies
The University of Sydney

March 2015

Declaration of Authorship

I, MohammadReza HoseinyFarahabady, declare that this thesis titled, 'A Framework for Approximate Optimization of BoT Application Deployment in Hybrid Cloud Environment' and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: _____

Date: 9 March 2015

Dedicated to my beautiful wife:

Nazanin

for her support and our endless love.

Acknowledgements

First and foremost I would like to thank and deeply acknowledge my supervisor **Prof. Albert Zomaya** for the continuous support of my study and research. He gave me insightful advice on my research, and showed me what a great mentor looks like. He has been always supportive, cheerful, and energetic.

My research would never have been fruitful without **Dr. Young Choon Lee**, at the University of Sydney, who helped me to explore interesting problems and guided me to conduct meaningful research. I sincerely thank him. I would like to thank and acknowledge the team at National ICT Australia (NICTA), and Prof. Aruna Seneviratne, for their support and valuable advices.

I would like to thank all my current and past supervisor and colleagues I worked with over the years, especially Prof. Hamid Sarbazi-Azad, Prof. Ali Jannesari, Prof. Ahmad Khonsari, Prof. Ali Movaghar, Prof. Shahram Rahimi, Prof. Mohammad Ghodsi, Prof. Jafar Habibi, Reza Moraveji, Mostafa Rezazad, Hooman Raiesi, Dr. Saeed Bastani, Dr. Hashem Hashemi, Hamid Samani, and Ali Bloori. I learned something new from each of them, and for that I am very grateful.

I owe many deep thanks and appreciation to my deceased father (forever in my heart), mother, sister and brothers taught me important things in my life and always support me. Also, thanks to my wife's parents, Reza and Shahin FarhangSadr, who have been a tremendous kindly support to me and my family.

Behind the scenes, our staff Evelyn Riegler, and Shari Lee made everything happen. I can't thank them enough.

The generous financial support of the **University of Sydney** and **NICTA** Institute is gratefully acknowledged, too.

I dedicate this work to my wife, **Nazanin**, in hope for building a better tomorrow. I would not have been able to go through this without her support. I am very fortunate to have her in my life.

And last, but not least, I would like to thanks all people in my life taught me how to be a better person.

Abstract

Recently, there has been a spectacular hike in the popularity of cloud computing systems in oder to facilitate the rental of IT resources on-demand over a network, charge according to a pay-as-you-go substratum, and multiplexe several users on the same physical infrastructure. Based on the deployment models, Clouds can be classified as public, private or hybrid. Although most of the main issues with cloud environment have been addressed in the past researches, there are still a number of challenges posing both industrial and scientific users to execute their application properly on Cloud. On the one hand, users want to cut down the total expenses incurred while meeting their performance wishes; on the other hand, choosing the key tradeoffs for attaining the most cost-effective deployment plan to scale out the application is almost out of the knowledge for common users, especially with the existing diversity of cost to performance ratio in cloud resources. To gain a complete view of the impacts of ever-increasing heterogeneity of resources and users' utility function on the efficiency of scheduling tasks in the Infrastructure as a Service environment (IaaS), it is essential that a framework be designed to take meticulously these factors into consideration.

In this thesis, we adopt a systematic approach to investigate the efficiency of near-optimal deployment of large-scale CPU-intensive Bag-of-Task applications running on hybrid cloud resources with the non-proportional cost to performance ratios (resource heterogeneity). We present analytical solutions in both known and unknown running time of the given application. Our framework tries to optimize the users' utility by choosing the most desirable tradeoff solution between the application completion time (makespan) and the total expenses incurred.

In particular, we approach the "cloud resource management" problem from various aspects which make this thesis bring following major contributions.

1- We propose a schedule schema to provide a near-optimal deployment of Bag of Task application regarding users' preferences. Our approach is to provide a set of Pareto-optimal solutions to the user, and then user may select one of the possible scheduling points based on her internal utility function regarding both the deadline and total cost factors. Our framework can work effectively when user's utility function follows specific patterns like p-norm models.

2- Our proposed framework can cope with uncertainty in the tasks' execution time. Uncertainty of the environment, here in the execution time of tasks, poses even harder challenges for reaching an optimal resource allocation in any high distributed computing system including Cloud environment. We address the case of coping with uncertainty in

task execution time by introducing two methods. First, we present an estimation method based on a well-known Monte Carlo sampling method called AA algorithm, which uses the minimum possible number of sampling to predict the average task running time. Second, we assume that our framework has access to some different code analyzer, code profiling or estimation tools, and then we introduce a hybrid method to evaluate the accuracy of each estimation tool in certain interval times for improving resource allocation decision.

3- We propose approximate (near-optimal) resource allocation strategies that run on hybrid cloud computing system. In essence, the proposed strategies first determine either an estimated or an exact optimal resource allocation schema based on the information provided from both users' side and environmental parameters. Then, in assignment phase, we exploit dynamic resource allocation methods to assign tasks to resources in order to reach an optimal schema as close as possible. Here we propose two approaches. A fast yet simple method based on First Fit Decreasing (FFD) algorithm, and a more complex approach based on the approximation solution of the transformed problem into a subset sum problem. Extensive real experiment results conducted on a testbed hybrid cloud (using Amazon EC2 as a public cloud and our in-house 2x40-core cluster as a private cloud) confirm that our proposed framework can deliver a near optimal resource allocation respecting user's utility function to improve effectiveness of cloud computing systems. Moreover, our approach outperforms an enhanced version of list scheduling algorithm by reducing both makespan and total cost significantly, in both clairvoyant and non-clairvoyant cases.

Publications

Some of the material in this thesis have been published in peer-reviewed conferences or journals as follows.

- M.Reza HoseinyFarahabady, Y. C. Lee and A. Y. Zomaya, "A Randomized Approximation Schema for Resourc Allocation of Cloud Environment Based on Monte Carlo Sampling", The Journal of Supercomputing, Vol. 69, Issue 2, pp. 576-592, Aug. 2014.

- M.Reza HoseinyFarahabady, Y. C. Lee and A. Y. Zomaya, "Performance Versus Cost: Pareto-Optimal Cloud Bursting, IEEE Transactions on Parallel and Distributed Systems", Vol. 99., pp. 1-11, 2013

- M.Reza HoseinyFarahabady, H. R. Samani, L. M. Leslie, Y. C. Lee and A. Y. Zomaya , "Handling Uncertainty: Pareto-Efficient BoT Scheduling on Hybrid Clouds" , 42nd Intl. Conf. on Parallel Processing (ICPP-2013), 2013, France

- M.Reza HoseinyFarahabady, Y. C. Lee and A. Y. Zomaya, "Non-Clairvoyant Assignment of Bag-of-Tasks Applications Across Multiple Clouds", 13th IEEE Intl. Conf. on Parallel and Distributed Computing, Applications and Technologies (PDCAT'12) Dec. 2012, CHINA, pp. 423-428, **Best Student Paper Award**

- R. Moraveji, J. Taheri, M.Reza HoseinyFarahabady, N. Babaii R. and A. Y. Zomaya, "Data-Intensive Workload Consolidation on Hadoop Distributed File System", 3th ACM/IEEE Intl. Conf. on Grid Computing (Grid 2012), China, pp. 95-103, Sept. 2012

Contents

List of Figures

List of Tables

Chapter 1

Introduction

If you want to shine like sun first you have to burn inside
–Persian proverb

Cloud computing can be defined as a model for delivering computing services (such as infrastructures, platforms, and softwares) through a network (typically Internet, or web-based tools and applications). This new business paradigm can help companies to reduce the cost of information management, since they are not required to own their hardware, storage, network, licensed software *etc.* any longer; and only pay the cloud provider based on their consumption (or PAYG[1] maner). Server virtualization, the isolated and transparent hosting of multiple servers in the same hardware, has accelerated the successful achievement of cloud computing in the last decade. In the clod computing context, almost every resources such as servers, computing, networks, and storage are construct and delivered to the end-user in a virtualized form. Virtualization allows to share resources among different applications in order to optimize the server utilization.

There are three fundamental models for delivering services in Cloud computing, namely Software as a Service, Platform as a Service, and Infrastructure as a Service. These collection of services can be considered to be built on top of each other. *Saas* allows end users to access complete applications remotely. It is mainly accessed through a web portal and service oriented architectures based on web service technologies Users typically use thin clients, whereas all computations are run, maintained, and supported by the service vendor (e.g Google docs, GMail). In *PaaS* model, virtual machines with already included operating systems, tools and frameworks required for a particular application are delivered to the user. The principal users of this layer are developers seeking to develop and run a specific application for a particular platform. Users of PaaS has

[1] Pay-As-You-Go

typically no access to control the operating system, hardware or network infrastructure (e.g. Amazon Elastic MapReduce). There are at least two important factors that *IaaS* virtual machines can be characterized: the promised level of processing capacity and the reserved amount of access to storage or network access. Clients can operate directly on operative system, storage, deployed application and sometimes firewall , in a convenient, on-demand fashion. Nevertheless, the customer cannot control the underlying cloud infrastructure. The most notable example in this category is *Amazon EC2*. In all three mentioned models, cloud services are completely transparent to the final user and require minimal interactions with the cloud provider. In our study, we consider cloud as a large pool of easily accessible visualized computer resources that are dynamically presented to and highly abstracted from the end-user in terms of a service level agreement[2] between user and the provider of cloud services[3].

Depending on the organizational structure or the provisioning location, cloud services can be deployed in different ways, too. Three main deployment models are distinguished as *public*, *private*, and *hybrid* service usage. Having been available transparently to the all type of end users, public cloud services and resources are working in a massive resource pools to be shared by users via a public network. Public cloud provider can signicantly reduce the prices for cloud resources, operation, support, and maintenance by optimization techniques of related operations such as load balancing, consolidation, *etc.* Having made services accessible only for users of designated organization, private cloud services are offering within the data centers belonging to the company. While it is not as large-scale as public cloud, private cloud give the advantage of control over corporate data, security guidelines, and system performance to the enterprise. To gain the maximum advantage from the good aspects of each of above mentioned models, most enterprises acquire cloud services by using a mix of multiple private and public cloud service types (known as hybrid model). Hybrid cloud solution provides fairly balance between scalability, performance, security, reliability, and the total cost of ownership.

One of the biggest challenges posed by a hybrid cloud is how to efficiently balance the workload between private and public clouds to reach the highest possible performance. Other challenges include keeping security and privacy, lack of standards, and compliance with evolving technologies coming to the market continuously. In the subsequent chapters, we will discuss more on our solution to overcome the issue of optimized capacity expansion of large scale application into hybrid cloud.

[2]SLA

[3]This is in accordance with IaaS paradigm

1.1 Large-Scale Bag-of-Tasks Application on Hybrid Cloud

It has been widely recognized that a large fraction of the workloads submitted to the high-performance and high-throughput computing systems can be categorized as a form of Bag-of-Tasks application. Massive parallelism has been seen in many programs in science (such as study of microbial genomes), engineering (such as parameter sweeps) and business analytic applications, all of them can be classified as BoT. This type of workload has been reportedly very common and that being as much as 70% in parallel systems and up to 96% in grid environment. The good news is that BoT jobs are formed of lots of tasks that are independent from each other, hence, can be run by their own. In other words, BoT jobs inherently exhibit horizontal scalability and can be fit naturally to the hybrid cloud deployment model. However, optimized scheduling of tasks to resources in a dynamic large-scale environment thousands of heterogeneous resources is a complicated problem. In fact, myriad of BoT scheduling algorithms have been suggested for different types of distributed systems, mostly tightly-coupled homogeneous systems. However, there are few researches in approaches which focus on the idea of optimizing multi-objectives simultaneously (in our context, minimizing the total execution cost as well as decreasing the running time of BoT applications) in a hybrid cloud environment. Having no prior accurate information about the running time of submitted tasks thwarts the most existing solutions in practical situations, too.

To counter such lack of solutions for cost-effective large-scale BoT scheduling in hybrid Clouds, we present several algorithms which are able to accomplish the submitted applications to satisfy conflicting users' objectives (makespan and total cost) with or without prior knowledge of the processing time of tasks. The approach can be used when either users have no knowledge about their personal preferences (regarding time or money) or they would like to hide it for some reasons (Chapter 5 and 6), or when their utility function characteristic follows a specific form (Lebesgue p-Norms functions) (Chapter 4).

1.2 Research Objectives

In contrast to scheduling of tasks in traditional distributed computing systems, cloud scheduler has less controller over the available cloud resources, which make cloud scheduling problem even more challenging than the traditional one. Therefore, the well-studied scheduling approach proposed previously for the traditional parallel systems cannot be directly applied to cloud environment. Below we identify the main objectives we take into account in this thesis.

It is normal that cloud environment possess resources with non-proportional cost to performance ratio. For example, this ratio for Amazon EC2 *m1.small* versus *c1.medium* instance is close to 1 over 5, at the time of writing this thesis. So, to scale out the CPU-intensive BoT application by dynamically renting public cloud resources, scheduler must be carefully aware about the degree of the performance that is achievable, since it is not in exact harmony with the usage cost of these resources generally. One objective of this thesis is to identify factors and major obstacles that influence reaching an effective deployment over the hybrid cloud environment.

The second goal of this project is to give the user a practical way to compromise between two conflicting objectives she normally possesses. On one side, the uesr tries to maximize the performance of tasks' execution on the distributed system; on the other side, this goal is in conflicting with achieving a minimum total cost (especially in heterogeneous environment). This issue will be discussed in more details in Chapter 5, and we will give a practical way how the cloud middle-ware can help users to choose among different possible allocation schema, even she is not fully aware about her utility function, which ties time and cost together. To this end, we use the concept of Pareto frontier and introduce some algorithms to reach solutions for approximating the set of possible achievable Pareto scheduling points. We try to develop some fully polynomial approximation schema for the seemingly NP-hard cloud resource allocation problem respecting the users' preferences. This will be discussed in Chapters 5 and 6. In addition, users may have different utility function model that can represent the relation between the time and cost that they actually want to spend on finishing their applications in the cloud. We introduce a new user-tunable utility function model in Chapter 4 which is based on p-norm model and try to reach an optimized cloud resource allocation for this model by giving a solution for the obtained binary nonlinear programming problem.

Several existing scheduling schemes are based on assumption of complete knowledge of tasks and environmental parameters, but in reality there is not any advanced information about these characteristics. Some of these parameters, such as tasks' running time, can be estimated using other sources, such as historical data, profiling and analysis tools. Because the classical deterministic approaches provide a schedule which is far from the optimal one in uncertain situation, we investigate how one can achieve a Pareto optimal resource allocation under uncertainty in the cloud infrastructure. These factors are introduced and discussed in more detail in Chapter 6.

1.3 General Approach

Our solution tries to use the strengths of private in-house infrastructure computing with the almost unlimited resource availability of public cloud system to provide an adaptive reliable environment that can control the trade-off between total incurred expense and the quality of scheduling in terms of makespan. In general, we often referred to a *Pareto-frontier optimization* approach for dealing with conflicting objectives of users. We generally decompose the resource allocation problem into several steps:

- *Submission.* In order to reach the best balance between time and cost in the resource allocation problem, it is important that users be able to submit their information as well as preferences properly to the system. In some cases, the user has *a priori* knowledge about the execution time of submitted tasks and is fully aware about her preference about relation of time an cost. However, in several other cases, users are doubtful about their preferences regarding the two conflicting goals. In addition, estimation of tasks' execution time is either too difficult or inaccurate to be followed by common users. Our framework provide facilities for getting the user's knowledge provided that this information is ready. If user has no knowledge about the parameters, the framework can still effectively continue its duty and reach a near-optimal solution. In Chapter 4, we specifically discuss the case when user's utility function form follows p-norm model. In Chapter 5, we introduce an approach to handle the case of unknown utility function. And in Chapter 6, our solution deals with the problem when both utility function as well as tasks characteristics are unknown in advance.

- *Optimization.* Based on the submitted information and collected data, our framework builds an optimized scheduling schema by taking the user's interest into consideration. Specifically, if the users' utility function follows the p-norm model, we transform the resource allocation problem into a binary nonlinear programming problem which a its relaxed version is solvable using the Lagrange multiplier technique. Details of such approach will be provided in Chapter 4. If task's running time is unknown in advance, then we exploit a Monte Carlo sampling method called $\mathcal{AA}$ algorithm to predict the average of task running time, and then an FPRAS[4] scheme will be used to find out the most efficient allocation. If user does not have any information about her utility function or reluctant to reveal her preferences to the cloud middle-ware, then our approach is to find all the possible

[4] An FPRAS algorithm can produce a solution in a certain distance, called $(1+\epsilon)$ factor, from optimal value with a confidence interval of δ

Pareto frontier schedule points and present this set to the user. Then framework asks user to choose one of the scheduling point based to realize the maximum utility. Finally, our system finds a near-optimal schedule by transforming the original problem into the famous subset sum problem. We use an FPTAS schema to solve the relaxed version of task allocation problem. Details of the process will be presented in Chapter 5.

In the case that user does not have any information about neither tasks execution time nor her utility function, but access to a some prediction tools which can estimate the running time of submitted tasks, our system can use the estimation tools to generate a set of best possible Pareto-optimal scheduling points that can be delivered to the user. To improve the measurement of each estimation tool, we devise a method that adds several breakpoints to each task. So, our system can compare the actual running time with the estimation values given by each estimator tool to assign an accurate weight to each one. Based on the desirable resource allocation schema selected by user, the workload of each machine is calculated, and the actual task assignment will be made by a technique inspired by First Fit Decreasing method. In order to control and correct the scheduling policy, the predicted tasks' running times values are compared with the real ones in specific breakpoints. Based on the difference between predicted and actual values, the system recalculates the amount of workload for each resource periodically. Details of this process will be presented in Chapter 6.

- *Deployment.* After finding the optimal solution for the above-mentioned circumstances, a deployment plan is necessarily to reach the suggested optimal point as close as possible. One can use a simple to understand, and easy to implement deployment plan such as *List* heuristic, a more complex solution such as subset sum-based approximate deployment plan which needs a lot of efforts to understand and implement, or a paradigm between these two approach, such as one based on First Fit Decreasing algorithm. We use all of these deployment schema in our work, and provide both advantages and disadvantages of each method individually. In addition, we will present a comprehensive performance analysis of each plan through Chapters 4 to 6.

1.4 Main Contributions

This thesis presents research on the methods of optimized deployment of Bag of Tasks application on Hybrid cloud computing environment. We answered several problems in

the form of new algorithms and frameworks, aiming optimization of different objectives. The tangible results of this work can be summarized as follows.

- **A schedule technique that propose optimized deployment regarding users' preferences** We designed a hybrid cloud scheduler which can propose a set of Pareto-optimal points to the user, and then user may select one of the possible scheduling points based on her internal utility function about time and cost. In this way, users are not needed to explicitly reveal their utility function. It is useful because sometimes users are not fully aware about their complex preferences, or they are not willing to express it, as they may think it may reduce their chance for any bargain in future. In addition, our framework can work effectively if user's utility function follows specific patterns such as p-norm models.

- **A framework that can cope with uncertainty in the tasks' running time** Uncertainty in task's running time is a clear fact. While in some occasion the lower and upper bound of this duration might be provided, the exact duration of a task is generally unknown until finishing of the task. A schedule must take the uncertainty associated with unknown values of such parameters into consideration and be able to reach effectively to an optimized solution. We will provide two methods to cope with such a phenomenon. First, we introduce an estimation method based on a well known Monte Carlo sampling method to predict the average task running time of submitted jobs. This method is an FPRAS algorithm which can be considered as the most efficient randomized algorithm for handling stochastic NP-hard problem. Second, we assume that our framework has access to different code analyzer, code profiling or estimation tools, and then we introduce a method to evaluate the accuracy of each estimation tool in certain times. Our algorithm can calculate the workload of each resources in an optimized manner based on the information provided in previous steps, and then try to reach the optimized resource allocation via employing an FPTAS algorithm. Details will be discussed in Chapter 6.

- **Near optimal resource allocation strategies that utilize hybrid cloud computing via dynamically renting available resources** Our infrastructure first determine either an estimated or an exact optimal resource allocation schema based on the information provided from both users side as well as environmental parameters. The next challenging step is how to actually assign tasks to resources in order to reach an optimal schema as close as possible. We exploit mainly two methods in order to approximate the optimal solution, including a simple and fast approach based on First Fit Decreasing algorithm, and a more complex approach

based on the approximation solution for subset sum problem which yields a closer solution to suggested values. All the introduced techniques have been extensively evaluated by real experiments run on top of Amazon EC2 as the public cloud and our in-house system as the private cloud.

1.5 Thesis Organization

This thesis consists of six chapters. Beside Chapter 1 presenting the introduction, the other six chapters can be followed as follows. Chapter 2 introduces necessary background knowledge and technical terms used in this thesis. The provided background knowledge includes an overview of main concepts in cloud computing environment such as virtualization, hypervisor, and different layers and deployment models of cloud environment. Then, we present several important features and notations of scheduling theory and resource allocation in cluster, grid, and cloud systems. And finally we introduce BoT application and present important previous works related to BoT scheduling in HPC environment.

In Chapter 3, we introduce a motivating example to describe the issues when a real world application is running on the hybrid cloud environment. Based on this example we try to analyze our research problems profoundly.

In Chapter 4, we propose a new cost model (based on the p-norm function) from users perspective to tie two goals of minimizing the total cost and makespan together. We reformulate the the problem of resource allocation of hybrid clouds as a binary nonlinear programming problem, and develop an FPRAS[5] to find a near optimal solution for it when there is no advanced knowledge about the processing time of BoT applications.

in Chpater 5, we present a framework for hybric cloud deployment of large-scale BoT applications with the main objective of optimizing the performance to cost ratio. In this scenario, user submits a BoT job by specifying both application and resource information. Here, task lengths of the BoT application are assumed to be known in advance. Then, our framework generates all scheduling candidates in Pareto-frontier and sends them back to the user to select one of them. Finally, our framework try its best to propose and deploy a schedule plan which is close to the selected point.

[5]fully polynomial time randomized approximation scheme

Chapter 6 extends the ideas presented in the previous chapters such that the framework can handle effectively with parameter uncertainty. In this way, we assume that task's running time is not give *a priori*, and there are some tools available to estimate it using sources such as historical data, profiling or analysis tools. The main question here is that how a scheduler can effectively use these tools to achieving an optimal cloud resource allocation. Again, the scheduler try to provide user with Pareto-frontier possible scheduling without asking the user to present her utility function, $\mathcal{U}(time, cost)$. And in the final step, the framework tries to reach the optimal scheduling based on the user's choice. This is not an easy job, because the tasks execution time will be revealed in the run-time, and may be different from the initial estimation. So, scheduler needs to take this into account and adjust the resource allocation dynamically. Finally, a summary of our contributions and future works will be presented in Chapter 7.

Chapter 2

Background and Related Work

In this chapter we try to provide a basic knowledge of characteristic, methodologies and modeling related to Cloud computing, followed by quick overview of virtualization technology, and then scheduling of BoT in HPC systems. Throughout this thesis, the chapter can be used as the first choice of reference for the reader, where further references will be given for deeper understanding. The chapter starts with basic idea of cloud computing, and virtualization concepts. Then, a closer look at the traditional scheduling theory is introduced. Finally. important characteristics and definitions of resource allocation and BoT scheduling in both traditional and modern HPC systems, which play a central role in the research target of later chapters, are presented. Since these concepts are presented in brief, we refer readers to [1] for more information about these basic concepts.

2.1 Cloud Computing Environment

With the advent of Cloud paradigm, all type of users all over the world, expect certain places, can access and utilize a set of hardware and software computing resources in the remote data-centers as services by means of API's [2]. This paradigm enables customers to avoid costly yet complicated provisioning plan, instead allowing them to utilize the infrastructure, platform and software offered by commercial or open source cloud initiatives.

Amazon was one of the major drivers of cloud computing, by first modernizing their internal data centers to use a cloud-like model and then creating a new product to provide cloud services to external customers in the form of Amazon Web Services (AWS) in 2007. Since then, cloud computing has rapidly gained attention

and importance, and has become a new industry buzzword. Cloud computing can take one of the well-known service models of Infrastructure as a Service, Platform as a Service, or Software as a Service (refer to [3] and Section 2.2.2). Deployment models can be either Public, Private, or Hybrid cloud. Furthermore, there are three distinguished models for deployment plan in Cloud paradigm, namely public cloud, private cloud, and hybrid cloud.

Several other key features have been recognized for cloud computing, too, including self-servicing usage , accessing a pool of resources via internet worldwide, very quickly elasticity for multi-talent and on-demand resource consumption in a controlled and measured way which altogether lead to increasing efficiency, faster return on investment, better utilization of the available resources, and ultimately reduction on IT-related costs. In addition, this on-demand feature allows big companies to dynamically handle burst computing needs (which is unpredictable usually) of their IT usage.

Cloud computing's adoption has been largely driven by advances in virtualization. The use of virtual machines allows for rapid configuration of software packages regardless of the actual underlying hardware. Those VMs can reside on the same physical host (almost) without the users being aware of each other[1]. Cloud services can offer a wide range of software services, too. Having provided several APIs or development tools, it allows developers making scalable applications on top of available services, which leads to approach the ultimate goal of running everyday IT needs in the cloud. API web services are valuable tools to enable information from one system (even running on different messaging product platform) to be transparently available to other applications over the network (or Internet generally). Several service providers are providing a common mechanism for delivering their services to be accessible by employing protocols such as SOAP or REST[2], well known examples include Amazon, Google, and Facebook.

SOA can provide standards to bring a loosely coupled distributed computing which is not depend on any protocols but simple ones such as SOAP or REST. In this paradigm, software resources (like Object, Methods, and so on) can be seen as *services* which are self contained and well-defined modules for providing standard business functionality completely independent from the state of others. So, to accomplish a complex business logic, an enterprise application will be made as a set of several services interacting together. In a cloud realm, an enterprise

[1] In practice, this leads to some degree of drop in both performance and security of the host
[2] known as SOA (service-oriented architecture)

application can be made by compiling several software services. Services can be run on the same or different cloud providers. Some well known examples of such building block services include authentication, accountant, time management, or payment system.

2.2 Virtualization

Today's large-scale data centers, which are normally composed of thousands of computers, are built to serve many users and host many disparate applications. For the purpose of both overcoming most operational issues of building and maintenance of such big data centers as well as improving the utilization of computer systems, *hardware virtualization* is currently considered as a perfect approach and plays a key factor indeed. This technique let several operating systems with different software stacks can execute on a single workstation [4].

Being adapting the virtualization technique as a mainstream component technology in today's data-centers, several researchers have addressed three basic issues regarding the management of a virtualized infrastructure, namely workload isolation, application migration, and consolidation[5].

Workload isolation is defined as the ability of separation of multiple software stacks in their own VMs to attain improved security, higher reliability, and even better performance. Both of computational and network resources can be isolated (either physically or logically) so that the resource consumption of one application or it failures cannot affect the others as shown in Figure 2.1.

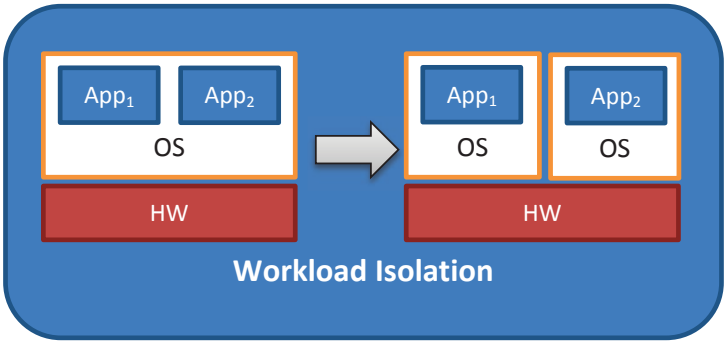


Fig. 2.1: **Workload isolation as one of the main virtualization usages: Isolating software stacks in their own VMs can improve system security and reliability**

Virtualized workload migration is defined as the ability of moving workloads (both data and applications) non-disruptively between dispersed virtual environments in real time without violating service level agreements. Since the introduction of virtualized server environments in progressive IT enterprises, workload mobility has been targets to address especial needs of the business such as disaster planning strategies and disaster avoidance, data center maintenance without downtime, applications' dynamic deployment, Data Center capacity expansion, migration or consolidation of data centers, resource distribution, and workload balancing. To achieve these goals, a true workload migration must support non-disruptive live migration of the VM memory content and its state, while both the VM volatile and non-volatile data have to be available throughout the live migration process.

Workload consolidation, one of the most significant opportunity introduced by virtualization technology, is a promising strategy to multiplex servers' capacity by hosting multiple VM workloads on a single server. This technique delivers a simple, cost-effective solution to not only reduce the number of servers in a data-center, but also can increasingly raise the utilization levels of servers [6]. Reduction of the number of servers leads to a lower consuming energy and carbon emissions level, freeing up the rack space, easier administration and management tasks, lower software licensing costs, and reduction in the total cost of ownership as a whole. Before initiating a complex server consolidation program, having a good analysis of end user requirements, application requirements, and the given infrastructure environment is absolutely essential [7, 8]. However, determining which VMs should be consolidated on which target to find the optimal performance and energy point is a computationally challenging problem for corporate data-centers with large numbers of servers and comprehensive traces of workload characteristics.

A noteworthy work in [8] introduced the relationships between performance degradation, energy consumption, CPU utilization and disk utilization in the consolidation problem. Using the traditional bin-packing problem, their proposed task consolidation algorithm relys on the Pareto front to reach a balance between the power consumption and system performance. Another notable study in [9] presents a comprehensive study of the latest solutions for controlling performance overheard in different cloud scenarios by highlighting the causes and metrics of VM performance overhead in IaaA cloud. They argue that concurrent live migration of VMs, concurrent deployment and snapshotting of several VMs, and inability

to isolate shared storage and network resources between VMs are commonly occurring routine operations in an IaaS cloud datacenter which are the main root of VMs performance overhead within a datacenter and significantly degrade their performance. Authors of [10] proposed two different energy-conscious consolidation heuristics, namely *MaxUtil* and *ECTC*, to decrease the total amount of energy consumed in a cloud environment without any negative affect on the system performance. The work improved in [11] by proposing a task consolidation algorithm that finds the best compromise between MaxUtil and ECTC heuristics. In order to incorporate task consolidation into the power management, Nathuji *et al.* [12] presented the *VirtualPower* approach which can exploit both hardware power scaling and software-based methods for controlling the power consumption of underlying virtualized platforms. For a detailed technical overviews of the above-mentioned concepts, the interested reader is referred to [13–21].

2.2.1 Hypervisor

Hypervisor firmware is a technology at the heart of system virtualization that can use to run virtual machines within a physical host server. The hypervisor manages the execution of multiple instances of a variety of guest operating systems by sharing the virtualized hardware resources such as processor, memory, disk and networking. Hypervisors provide the means to logically divide a single, physical blade server, allowing multiple operating systems to run securely on the same machine to increase its utilization. Popek and Goldberg breaks these middle-wares down into two different catagorize of Type-1 or *bare metal*, and Type-2 or *hosted hypervisors*[22]. Being completely independent from the operating system, Type-1 hypervisors are interacting directly with hardware, partitioning it into multiple virtual machines, and offering a higher level of virtualization efficiency and security. While Type-2 hypervisors are those that run within the host operating system, so the underlying hardware is directly managed by the host OS.

By building a a more efficient virtual data center environment, bare-metal hypervisor are dominantly used in both the current cloud computing market as well as business-wide virtual desktop deployments. Furthermore, concepts such as server consolidation and server orchestrating are made more efficient by introducing Type-1 virtualization. As there is no underlying operating system, the install footprint can be greatly reduced (for exmaple the overall footprint of the current version of ESXi is only 150 MB. The management functionality of Type-1 platform is often accomplished through remote management tools.

VMware vSphere/ESX Server/ESXi, Microsoft Hyper-V, Citrix XenServer, Xen, Intel VT, Oracle VM server and VirtualBox, IBM PowerVM, and AMD-V are the current existing major players in the market of underlying hypervisor and virtualization platform.

2.2.2 Cloud Types

Inspired by the abstraction level of the provided capability, services in cloud computing can be categorized to three broad classes of Infrastructure as a Service, Platform as a Service, and Software as a Service [2].

IaaS is the most basic on demand cloud-service model which offers virtualized resources such as computation, mass storage, bandwidth and communication as an abstracted view on the hardware to the users. By allocation physical resource, the virtualization layer provide each virtual machine with an environment that it thinks the whole underlying physical resource belongs to them totally. Being considered as the bottom layer of cloud systems, there are several dedicated infrastructure services available currently in the market for calculation tasks (Hadoop), mass storage (Amazon S3, Dropbox) or even for the networks (OpenFlow). Amazon Web Services [23] mainly offers VMs carrying a software stack. These customizable VMs can be configured very similarly to the ordinary way of configuring a physical server, but with less effort to save lots of time. More discussion on the technical points of Amazon cloud service architecture can be found on [24]. Table 2.1 shows an extended list of IaaS cloud serveices and tools.

In PaaS model, a computing platform and a solution stack are presented to the user as an abstracted programming platform with encapsulated infrastructure. In this service model, developers are able to create and deploy their own applications without knowing or concerning with the number of processors or amount of memory that applications will be using. There are normally different levels of scalability and maintenance offered by these services. Sometimes other facilities for application design, development, marshaling, database integration, security, scalability, testing, and deployment are also included. Google AppEngine, a well-known PaaS example, provide an environment for developing or hosting scalable (Python or Java) web applications. Other famous PaaS technology platform providers include Azure by Microsoft, Facebook Platform, Salesforce, and Zoho Creator.

Software as a Service, the most dominant cloud model, is an application delivery model in order to alleviate the burden of software maintenance, local

Table 2.1: **Infrastructure as a service: offerings and tools**

Organization	Cloud service	Description
Amazon	Elastic Compute Cloud (EC2)	Virtual servers
Amazon	Dynamo	Key-value pair store
Amazon	Simple Storage Service (S3)	Mass storage
Amazon	SimpleDB	Database as a Service (DaaS)
Amazon	CloudFront	Content Distribution Network (CDN)
Amazon	SQS	Message queues
Dropbox	Dropbox Cloud Storage	Mass storage
Reservoir	Open Nebula	Virtual open source server pools
Google	Google Big Table	Distributed storage of structured data
Google	Google File System	Distributed file system
Eucalyptus Sys.	Eucalyptus	Open source AWS implementation
10gen	Mongo DB	Database for cloud storage

installation, development and testing. In this model, the target application entirely resides on the remote server, enables users to access desirable services of the application on demand. Several applications such as DBMS software, management software, CAD software, development software, accounting, CRM, ERP, CM) and service desk management have relied on the SaaS model, experiencing a growing market of $21bn in 2014 [25]. Microsoft live (office) applications, Google docs, Salesforce CRM, Adobe Photoshop Express eCloud Manager are among the well-known industry players who are aggressively adapting and developing SaaS applications.

2.2.3 Cloud Deployment Models

Recommended by the National Institute of Standards and Technology (NIST), A deployment model distinguishes the cloud environments based on some important factors such as the ownership, size, customization capabilities, and security requirements. While public cloud model has been introduced as the initial approach to building IT cloud services, other types of service deployment have been emerged to provide companies with different degree of flexibility or security control, each owns its advantages and disadvantages. In this sense, we accept *"public"*, *"private*, and *"hybrid/mixed"* clouds as the main deployment plans that organizations can choose to deploy their applications.

A "public" cloud infrastructure comprises cloud services owned by a large cloud service provider and publicly accessible to customers. The cloud provider is responsible for the creation and on-going maintenance of the public cloud and its IT resources. In this model, customers are able to dynamically provision computing resources from cloud provider over the Internet. Studies suggest that using of resource in public cloud can be the most cost effective deployment model for the small and medium sized companies. In return, customers must agree to receive a reduced control and monitoring in terms of resource security and governance.

Scenarios where cloud infrastructure is dedicated, delivered or belonged exclusively to a specific company, are referred to as "private" cloud. Sharing many characteristics of a traditional client-server architecture, the available services are delivered to the customers on demand from a distributed infrastructure in a transparent way. Security is the main concern that a private cloud would be preferred over a public one. So, control over the sensitive data remains within the organization for its internal use. The best-known examples of cloud management platforms are Eucalyptus, CloudStack, OpenStack and OpenNebula to turn a data-center into a private cloud.

In the hybrid cloud model, services from both the public and the private cloud are brought together. Being the most common method of cloud deployment in a large organization, hybrid cloud tries to combine the advantages of the other models to offer firms the maximum flexibility. So, companies can leverage third party cloud providers to control any unpredicted sudden burst in workload by transferring certain operations or load peaks to the public cloud. Several issues such as performance tuning, business continuity, disaster recovery, standards and governance issues, defining and achieving quality of service (QoS), data storage or its ownership, the interoperability issues, and security and privacy issues at various levels of the enterprise application need to be addressed properly when an enterprise decide to take the advantages of using a hybrid cloud architecture.

2.3 Bag of Task Application

Bag-of-Tasks are referred to applications with a large amount of independent tasks. A *task* is a basic compute unit in the concerned infrastructure. If several tasks are grouped in a larger structure that needs to be completed in full, it is refereed as Bag of Task application. BoT application, also known as *parameter sweep*, are composed of myriad of independent tasks with a huge demand of computational

power such that easily overwhelm the compute power of todays' personal box. Previous studies showed that Bag of Tasks are highly proper for executing on large distributed systems such as cluster, grid or cloud platforms [26–30].

There are at least two reasons why studying BoT applications are important for distributed systems' research community these days. First, many applications in science and engineering exhibit massive parallelism and are composed of thousands of independent tasks; hence, can be classified as BoT applications. Second, profound analysis of characterization of workloads in parallel and distributed systems shows that, with few expectations, around 96% (70%) of total CPU time consumption in the today's grid (or cluster) environment is generated by BoTs [31–33]. Because of this large fraction of job workloads in today's parallel and distributed system, proper resource allocation regarding of true characteristics of BoT applications is critical for any resource managing system that desire to achieve an acceptable level of performance. However, scheduling of such applications is not a routine work. There are lots of parameters, from both theoretical and practical aspects, which make it a complex and difficult decision to properly allocate the available resources of to run BoT tasks. Heterogeneity of platform resources (or instances in cloud environment), dealing with not only the computation intensive tasks but also data intensive ones, presence of users with conflicting interests and vague objective functions, designing a fault tolerance managing system, and complexity of stochastic behavior of submitted workload or job are among of most important factors that have been proven to cause the designing of an optimal resource management system of BoT application be as one of the most daunting problem seen in the field. A comprehensive survey about significant efforts on design of scheduling algorithm to manage Bag of Task applications on different homgeneous or heterogeneous distributed platforms such as Cluster, Grid, or Cloud computing will be presented in the next sections.

2.4 Algorithmic Complexity & Approximation Schema

Complexity theory is a mathematical powerful tools to classify problems as "hard" or "easy", from computational point of view. There exist several problems that can be solved in polynomial time non-deterministically, whilst a provided solution can be verified for the correctness in polynomial time. These problem are classified as NP complexity class. P class contains different decision problems in which are solvable on deterministic Turing machine in the polynomial time. NP-complete

complexity class contains all problems that are in NP class and if can be decided in polynomial time, then other problems in NP class, which are reducible to that problem in polynomial time, would be solvable in polynomial time, too. NP-hard complexity class contains all problems that are reducible from at least one NP-complete problem in polynomial time. NP-hard problems do not necessarily belong to NP class. Regarding the input size of the problem, nobody can yet discovered a method for solving NP-complete problems runs in a reasonable amount of time. Figure Figure 2.2 shows a schematic diagram of relation between P, NP, NP-complete, and NP-hard complexity classes, provided that $P \neq NP$. A *pseudo-polynomial time* algorithm is one that its execution time is polynomial in terms of the actual numeric value of its input, but still is exponential in the number of input digits, which is also known as the size of the input). A problem belogns to *weakly NP*-complete class provided that it is an NP-complete problem with known pseudo-polynomial time algorithms, otherwise it is called *strongly NP*-complete [34]. The interested reader is referred to the Garey & Jahnson book for extensive points of this theory [35].

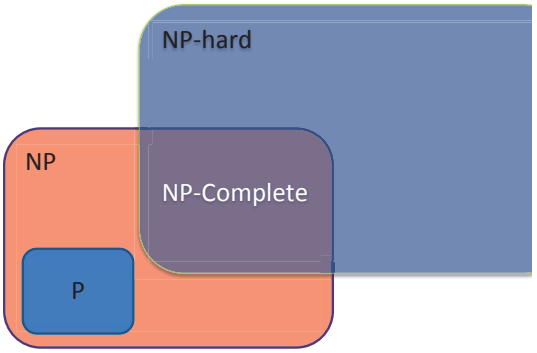


Fig. 2.2: **Diagram of the relationship between P, NP NP-complete, and NP-hard complexity classes provided that $P \neq NP$**

The most common approach to deal with the problems in NP-complete or NP-hard class is to relax the requirement of finding an optimal answer, and alternatively establish a good solution that are provably close to the optimum. All various type of heuristics such as simulated annealing, genetic algorithms, and Tabu search are examples of techniques yield good results in practice. *Approximation techniques* follows the same approach, trading quality for time. Approximation algorithm tries to find a solution as close as possible to the optimal value in a low-ordered reasonable amount of time. More formally:

Definition 2.4.1. *a ρ-approximation algorithm for an minimization (maximization) optimization problem is a polynomial time algorithm for which it has been*

proven that for all instances of the problem produces a solution whose value of the approximate solution will not be more (less) than a factor ρ times of the optimum solution.

ρ is also known as "performance guarantee ratio", "approximation ratio", or "approximation factor". ρ can be figured as the algorithm's quality measure. In other words, closer value of ρ to "one" shows a better solution. If the approximation ratio is bounded by a constant in polynomial-time, then the problem is said to belong to the "APX" class. Another important class of polynomial time approximation algorithms is known as "PTAS" (polynomial time approximation scheme) class [36]. An algorithm of this class receives $\epsilon > 0$ value as well as an instance of an optimization problem as the input, and it produces a solution with performance guarantee of $1 + \epsilon$ [37, 38]. The execution time of a PTAS algorithms is polynomial regarding n, for a fixed value of ϵ. However, a major issue with PTAS algorithms in practice is that its running time can increase dramatically when ϵ approaches to zero, for example consider the running time of an algorithm bounded by $O(n^{\frac{1}{\epsilon}})$. To put more restriction on the algorithm running time, and make it useful in practical situation indeed, a class of algorithms called "FPTAS" (fully polynomial-time approximation scheme) has been introduced which guarantees the algorithm to be polynomial in terms of both n and ϵ^{-1}. It is obvious that an FPTAS schema is the most desirable achievable result can be derived for an $NP-$ problem [38].

2.5 Scheduling Theory

The decision-making process of allocation of available resources to submitted tasks in order to optimize one or more objectives of interest is called "scheduling". The reason why searching for a good schedule is worthy lies behind the fact that the choice of schedule plays one of the most significant role on the system's performance. The scheduling problem are among the most difficult problem in combinatorial optimization and stochastic modeling theories, as characteristics of both the resources and tasks can have many different formats. For example, tasks may take specific priority level, starting time or certain due date. The target objective may take several shapes, too. For example, one may be interested solely in the minimization of the completion time of the last task, while another may be interested of the minimization of the number of tasks completed in a given period,

or minimization of the total cost of scheduling, or even a combination of several objectives.

2.5.1 Deterministic Scheduling: Preliminaries & Notation

A very considerable amount of research was undertaken to help understand the deterministic scheduling theory. A very useful notation has been invented and developed during this time. Throughout this thesis, an enhanced version of this notation will be used. We use n to refer to the total number of jobs, m to show the number of machines, subscript j and i to refer to a task and a machine, respectively. Both n and m are assumed to be finite. The p_{ij} represents the time required of task j to be processed completely on machine i. The *due date* d_j of task j represents the latest time that job should be delivered to users. Completion of task j after d_j is allowed; however, a penalty may be incurred. To show the importance of task j relative to the others in the system, we normally use a priority wight factor of w_j.

Scheduling problems can be described by a triplet $\alpha|\beta|\gamma$, where α field describes the machine environment, the β field corresponds to the processing characteristics and problem constraints, and the γ notation reflects the target objective function(s) should be optimized. Some traditional values specified in the α field are: single machine ($\alpha = 1$), Identical parallel machines ($\alpha = P_m$), parallel machines with different speeds ($\alpha = Q_m$), and unrelated parallel machines ($\alpha = R_m$). In our study, the machine environments model can be categorized as Q_m.

The β field captures the various processing restrictions on the problem. Release dates ($\beta = r_j$), preemptions allowance ($\beta = prmp$), precedence constraints ($\beta = prec$), and machine breakdowns ($\beta = brkdwn$) are some possible entries. In this study, we will use additional constraints as appears in the Bag of Task scheduling in cloud environment. γ field shows the objective to be optimized (often minimized) and essentially captures the designing of scheduling algorithm. Although a large list of possible objective functions can be figured out, there are still a few which are fundamentally well studied before by different researchers. Examples of possible objective functions include makespan ($\gamma = C_{max}$), maximum lateness ($\gamma = L_{max}$), total weighted completion time ($\gamma = \sum w_j C_j$), discounted total weighted completion time ($\gamma = \sum w_j(1 - e^{-rC_j})$, total weighted tardiness ($\gamma = \sum w_j T_j$), weighted number of tardy jobs ($\gamma = \sum w_j U_j$), total cost and so on [39]. In this study we mostly focus on the makespan and the total cost of scheduling, as well as different combinations of them.

2.5.2 Scheduling Theory Background

The work presented by Horowitz and Sahni in [40] is probably the first important research study discussed the problem of scheduling of a set of independent submitted tasks in a environment consists of several heterogeneous multiprocessor units with the goal of minimizing the makespan. Later Graham showed that list scheduling (LS) rule can produce an $O(2 - \frac{1}{m})$ approximation schema for any given instance of $P||C_{max}$ problem and this worst case is tight [41]. List algorithm selects the next available task in a pre-specified queue for scheduling and assign it to free processors greedily. Although the worst ratio of list scheduling seems to not be impressive, further study on the average performance of this algorithm showed that if jobs' processing time are uniformly distributed, then the solution provided by list scheduling rule is asymptotically optimal when n grows faster than m [42].

Authors of [43] studied several algorithms which has been already presented for the problem of scheduling of n tasks that are on independent from each other and ready to run on m processors which are non-identical, too. They particularly showed that, for large n in this problem, LPT schema (longest processing time first) yields a near optimal solution. The non-preemptive assignment of this problem has been examined thoroughly in [44–48]. Authors showed that MULTIFIT[3] and LPT algorithms can produce $\frac{13}{11}$ (which is tight) and $\frac{4}{3} - \frac{1}{3m}$ approximation solutions, correspondingly.

Burkard and He in [49] showed a worst case bound which is tight for scheduling of jobs by means of MULTIFIT algorithm. The underlying system that they were interested was two parallel uniform machines, and they combine the MULTIFIT heuristic with FFD and LPT heuristic to reach a better solution. It is noteworthy to mention that the non-preemptive scheduling case for n jobs which are independet from each other on a set of uniform resources in order to reach a minimize solution for makespan is NP-hard, even in the case of two resources.

Hochbaum and Shmoys of [50] presented the first PTAS approximation for which the solution has a relative error $\epsilon > 0$ comparing to the optimum value. Having used an ϵ-dual approximation algorithm, the main idea was to take tasks execution times away into $k = \frac{1}{\epsilon^2}$ equal span intervals, and then round each execution times to the lower end of containing interval. Although the new version of this problem (with fixed number of task lengths) now is solvable by using a simple

[3]The MULTIFIT algorithm is based on the bin-packing heuristic algorithm called the first-fit decreasing (FFD). MULTIFIT uses FFD and binary search to find the shortest feasible schedule.

dynamic programming technique in polynomial time, its running time is so huge $(O\big((\frac{n}{\epsilon})^{\frac{1}{\epsilon^2}}\big))$, which makes it disputable in practical situation.

Authors of [51] presented the first efficient polynomial $O(\log m)$ and 6− approximations for the problem of $Q|prec|C_{max}$[4]. In [52], it has been shown that there is not any approximation solution for the problem of $P|prec|C_{max}$ with approximation ratio better than $\frac{4}{3}$ unless $P = NP$. In [53], author derived an improved 2-approximation version for these two problems. The essential idea is to give priority to those tasks that have highest level within the critical path (CP). In [54], several PTAS solutions for scheduling of n jobs with given processing time on a set of uniformly related machines with different weights have been presented. They analyzed several objective functions corresponded to the makespan or completion time somehow, such as optimizing different l_p norms related to the completion time vector[5], or the problem of job scheduling with rejection for which delay to run a job after its deadline associates with a specific amount of penalty, and the final aim of the problem is to minimize the maximum amount of completion time in addition to minimize the total penalty of rejected tasks. The basic idea used in [54] was replacing the size of all jobs by a nearest constant number, and then giving an exact solution for the rounded instance. Finally, they handle to re-construct a near optimal solution for the original scheduling problem using a backward method. We will use a similar rounding technique in our work, too, which will be presented in Chapters 4 and 5.

Sahni presented an FPTAS algorithm for the minimum criteria problem of $P_2||\sum w_jC_j$ [57]. The main idea used in that work was to order tasks based on the weighted shortest processing time (WSPT) rule. This technique later used by different researchers to solve many other scheduling problems (*e.g.,* look at a discussion presented in [58] introduced an FPTAS for $Q_m||\sum w_jC_j$).

If each job has a release date r_j then minimizing $\sum C_j$ is NP-hard in the non-preemptive case. Lenstra [59] showed that even the problem with single machine case $(1|r_j|\sum C_j)$ belongs to NP-hard class. Baker presented an $O(nlogn)$ solution for the preemptive version of this problem, *i.e.* $1|prmt, r_j|\sum C_j$ [60]. Before, Smith [61] showed that the problem of $1|\bar{d}_j|\sum C_j$ can be solved in $O(nlogn)$ by choosing the job with largest processing time at each time. Interestingly, authors of [62] showed that the problem of $1|prmt, \bar{d}_j, r_j|\sum C_j$ is NP-hard. It means that if there

[4]scheduling of jobs with some defined precedence-constrained running on a set of uniform machines in parallel manner

[5]This objective function is motivated by storage allocation problems [55, 56]

exist both release dates and deadlines in a problem, there is not any polynomial-time solution even in the simple case of only one machine. Hence, we can solve only problems with either deadlines or release dates, but not both.

First approximation algorithms for the problem of minimization of makespan for a set of jobs with precedence constrained and release date on a set of uniform machines in parallel manner ($Q_m|prec, r_j|C_{max}$) has been presented in [63]. Their algorithm has an $O(\sqrt{m})$ approximation ratio. The approximation bound for this problem has been improved significantly in [64] where Chudak invented an $O(\log m)$-approximation schema, after fifteen years. Recently, authors of [65] derived a heuristic algorithm with improved ratio bound of $1+O(\sqrt{m})$ for the problem of $Q_m|r_j|C_{max}$. Du showed that the problem of $P_m|prmt, r_j|\sum C_j$ is NP-hard, even for two identical and parallel machines [66]. Thus, when jobs have different release dates there is no hope to develop a polynomial time optimal solution for multiple machines. The case is much promising when jobs possess different deadlines. Authors in [67] gave a polynomial-time algorithm for $P_{m>1}|prmt, \bar{d}_j|\sum C_j$, when all tasks are submitted to be scheduled at time $t = 0$, preemption is allowed, and all jobs must complete its processing before or at their deadline. Afterward, Gonzales *et. al.* introduced a polynomial-time solution for $Q_m|prmt, \bar{d}_j|\sum C_j$ problem [68].

In [69], authors analyze the problem of scheduling n jobs without precedence constraints on a set of m uniform machines while preemption is allowed without any penalty. They have been successfully generating the entire Pareto-optimal curve of schedules for the flow-time and make-span objectives. Flow time, defined as $f_j = C_j - r_j$, measures how long a job remains in our system, waiting to receive the required service. From the quality of service point of view, long tasks can tolerate longer delays than short tasks. The strategy they used was to develop an initial algorithm that produces a schedule with minimum flow-time, subject to a fixed make-span deadline, which is an alternation between two well known rules of Shortest Processing Time on Fastest Machine (SPT-FM) as well as Longest Remaining Processing Time on Fastest Machine (LRPT-FM). A modified version of this idea is utilized in our study, which will be presented in Chapter 5.

Scheduling of identical independent jobs on a set of uniform parallel machines when processing times or machine speeds are random and preemption is not allowed has been addressed first in [70]. Authors discussed that in random case the meaning of efficiency and cost-effectiveness of a give schedule is difficult to define.

They argued that not only the expected value of an objective parameter is important, but the variance of those performance metrics should also be considered. In their work, authors addressed two performance measure of minimizing the expected sum of completion times and maximizing the probability that jobs will be finished before a common due date.

And finally, the problem of minimizing the total tardiness or total earliness has been shown to be challenging scheduling case even in the presence of single processor. All of the introduced techniques such as integer programming, branch and bound, *etc.* take too much computational time to reach a promising solution [71–76]. Surveys of techniques introduced for scheduling with objectives of total earliness and total tardiness can be found in [77–80]. There are other new criteria have been defined in this context, such as stretch factor, $s_j = \frac{f_j}{p_j}$, which reflects user willingness in a better way. Interested reader is referred to works on [79, 81–84]

2.6 Resource Allocation in HPC Systems

The well established paradigm of *distributed computing* tries to combine the capability of myriad of resources which are distributed through a network in an efficient way. This paradigm normally uses to satisfy major needs of distributed application such as scalability, reliability, and information sharing. In many type of applications, the nature of distributed applications demand using a communication network that connects many boxes. A distributed system often possess a higher reliability than a single CPU or machine. Apparently node failures in a distributed system should not stop running of the whole application comparing to failure in a single resource. Two important techniques to provide a reliable distributed system are *check pointing* and *replication*. For a thorough review of the original motivations from the users point of view into distributed systems, interested reader is referred to [1]. *Scalability* can be defined as the ability of a system to be enlarged to accommodate and handle a growing amount of workload. Adding more resources into a particular application can be categorized broadly as either horizontal or vertical scaling. Scaling out[6] is referring to add more nodes or resources to a system. For example, adding a new computing resource to an already running system which is distributed, too. Alternatively, a system can be scaled up[7]. It means to add one (or more) powerful resource(s) to a single node

[6] Scale horizontally
[7] Vertical scaling

in a system while keeping other resources unchanged. For example by adding a faster CPU (or replacing the old RAM units with faster memory units) to a single computer box, we literally scale up the system.

HPC systems can be categorized to three main classes of "cluster", "grid", and "cloud". *Cluster* systems (which is a base platform of all other distributed computing paradigm) is an aggregation of the local, independent resources work closely together to share the workload, with the main goal of representing a uniform system image to the external users. The befit of using a cluster system over a supercomputer is that normally the total cost of buying and setting up individual machines by a high-speed network is less than higher-priced supercomputer system, while a cluster results in higher reliability, higher availability, as well as lower maintenance costs. IBM's Parallel Sysplex [85], Microsoft Cluster Service [86], and Oracle Solaris Cluster [87] are different technologies from today big software vendors to implement a cluster system. A modern cluster constitutes of a set of computer systems which connected together via high-speed and low-latency network (or LAN switches) and have access to different storage devices, too [78]. Such systems can be served as execution of compute-intensive applications as well as replicated storage or backup servers.

Resources in *grid* computing are connected together in a more loosely coupled way than clusters. Some specialized software normally used in grid computing to make it easier to mange distribution of data and algorithms across the grid's machines. Grid systems have been known for different administrative domains which are dispersed geographically while each of these domain sites has possess their own access policy, make them a distinguishable contrast form cluster systems. Examples of famous grid system includes World Community Grid which aims to create the world's largest public computing grid to tackle most important scientific research projects [88], European Grid Infrastructure which tries to provide a grid computing environment by accessing to high-throughput computing resources across Europe [89], folding project in Stanford University which concerns of disease research to simulates protein folding [90], SETI@home runs on top of BOINC[8] environment and hosted by University of California, Berkeley with the main purpose of analyzing radio signals to detect intelligent life outside Earth [91], Einstein@Home which intend to search through data from the LIGO detectors for evidence of continuous gravitational-wave sources [92], and the list goes on.

[8]BOINC (Berkeley Open Infrastructure for Network Computing) is a famous open source middle-ware system used for volunteer and grid computing

Cloud computing, as mentioned several times before in this study, bring the dynamic provision of scalable virtualized resources over the network (or Internet) worldwide. Visualization is a key concept utilized by cloud paradigm to provide on-demand resource reservation. For a comprehensive comparison between grid and cloud paradigm from different aspects such as architecture, programming, data, security or compute model, and the business schema we refer interested reader to [78, 93]. Due to the fact that resource allocation scheam used for HPC systems have a huge impact on their performances, we try to collect a compact introduction to different feature of some well known resource management strategies used in these systems in the following sections. Interested readers are referred to [78, 93, 94] for a comprehensive surveys and system taxonomies.

2.7 Resource Allocation in Cluster

There are currently two major approaches of "batch scheduling" and "virtual slices" to share resources in a cluster. In batch systems, resources' requests are placed in some queues and wait till gain exclusive access to those resources, while in virtualized approach, allow users to have a virtual slice of actual physical resources. A major problem in both approaches is that a percentage of resources is not being fully utilized; hence, the resources are wasted. During last two decades, several prototypes and tools have been constructed to address properly the problem of sharing cluster resources. Here we present a brief summary of the top major contemporary and active projects for handling the problem of resource allocation in cluster systems. The list includes HTCondor [95], MOSIX [96, 97], QNX [98, 99], and LSF [100] projects.

HTCondor is a free open source project running by the Center for High Throughput Computing in the Department of Computer Sciences at the University of Wisconsin-Madison since 1984 [101]. Condor framework provides a job queuing mechanism and priority scheduling policy to monitor and mange effectively consumption of available resources as well as to accomplish workload management of compute-intensive jobs. It supports transparent migration of a job to a different machine if it detects that the current hosted machine is no longer available. The essential Conder's features includes check-pointing and automatic transparent migration[9], remote system calls, ordering of job execution to satisfy dependencies

[9] A job can be transparently migrated to a different machine if HTCondor detects that the current hosted machine is no longer available

among jobs, introducing pool of machines, constructing a grid computing environment, handling to return back resources promptly upon owner request, and classAd mechanism to match-make between resource and jobs. ClassAd means that while jobs can specify both requirements and preferences of themselves, each machine can also state preferences and requirements of jobs it is willing to execute. For a detailed specification of Conder system, interested readers are referred to [95] and a complete manual in [102].

MOSIX[10] is a distributed cluster operating system intended to improve the Linux kernel to manage cluster computing and multi-cluster clouds. It has been developed since 1977 at the Hebrew University of Jerusalem. It provides applications with a SSI[11] as a single Linux run-time environment. It supports automatic resource discovery and dynamic load-balancing, which means users can continuously run their normal applications in a sandbox while MOSIX transparently and automatically seek new resources or migrate processes among nodes or live queuing of batch jobs. form the scalability point of view, workstation and cluster nodes can join or leave at any time as a full transparency model. By using optimization algorithms that respond to variations in cluster resources and a decentralized control and autonomy model, MOSIX is more robust against failure. Several tests confirms that this cluster operating system is most suitable for running scientific and engineering applications with low to moderate amount of I/O. The interested reader is referred to [103–107] for a comprehensive review about MOSIX.

QNX is a commercial distributed operating system which was originally developed by Canadian company QNX Software Systems[12] and supports real-time and embedded systems by successfully applying the idea of micro-kernel operating systems. This operating system has been successfully used in a variety of devices such as shipping navigation systems, industrial control systems, medical instruments, defense systems, nuclear power plants, and other mission-critical applications. The main basic idea of QNX is that most of its kernels should run in the form of small tasks, or micro-servers which makes the whole operating system quite small. This idea is completely different from other traditional monolithic kernels like Windows 9x series or Linux, which thier kernel composed of huge numbers of parts with special abilities. The main kernel only includes CPU scheduling, inter-process communication, interrupt handler and timer. All other abilities (even memory management) must run as user processes.

[10] Multicomputer OS for unIX
[11] Single System Image cluster is a cluster environment that appears to be one single machine
[12] QNX company was acquired by BlackBerry in 2010

Other main features of QNX microkernel include transparent distributed processing, modular architecture, symmetric mulitprocessing (SMP), process monitoring, and following to POSIX API specifications. Because QNX separates components such as drivers, protocol stacks, file systems, and applications from each other and from the microkernel itself in a clean way, the system upgrades and software hot-swaps on the fly is supported without expensive downtime or any experience of interruptions in its servicing functionality [99]. In addition, applications and system services can distributed through the without any special code within the QNX microkernel system. QNX allows leveraging of massive multiprocessing compute power of the available multi-core and multi-processor platforms, while offering a light approach to fault isolation as well as automatic recovery, which provide a self-healing system [99]. The interested reader is referred to [99, 108–111] for more in-depth information on QNX.

Platform LSF[13] is an enterprise-class job scheduler and resource management suite for distributed HPC environments which was originally developed by Platform Computing and was acquired by IBM in 2012. It supports policy-driven allocation (by creating multiple queues) and high-throughput low-latency scheduling tools to handles job management across distributed resources for executing batch jobs on networked Unix and Windows machiens. LSF provides several tools for managing, monitoring, and analyzing the workload for a heterogeneous network of computers and it unites a group of UNIX and NT computers into a single system to make better use of the resources on a network. It automatically selects hosts in a heterogeneous environment by considering both the current load conditions as well as the resource requirements of the applications [14] into account [112]. It is also one of the job scheduler mechanisms used by GRAM[15] in Globus Toolkit. For a complete overview of LSF suite, the interested reader is referred to [113–116]

2.8 Resource Allocation in Grid

A grid system can be described as an agreement between several independent (and most likely selfish) organizations to share resources, with little or no primary controlling unit for any type of mandatory interaction. Each organization owns and runs a cluster which can possess identical or non-identical processors. The

[13] Load Sharing Facility

[14] batch processing

[15] Grid resource allocation manager is a software component of the Globus for accomplishing several features such as lcoating, submitting, monitoring, and canceling jobs on the resources of Grid system

objective of each organization could be minimizing the time at which all the locally produced jobs are finished. Interestingly, an organization does not care about the performance of other organizations or about the actual global makespan. In this setting the famous problem of multi-organization scheduling problem (MOSP) is defined as the minimization of the makespan of all jobs, which is not hard to show that MOSP is an NP-hard problem. In addition, a non-cooperative solution for which all the organizations compute and minimize their own make-span can be several times worse than the optimal one. From the game theory, we know that a specific price must be paid to get the organizations incentive to participate for producing a better solution within the platform when each companies tends to behave selfishly. More discussions about this approach can be found in [78, 117–120]

In a remarkable work in [121], authors addressed the problem of on-line scheduling of jobs in a hierarchical server environment. The goal is to minimize the maximum total load on each server. Here, being online means that decisions must be made without any knowledge of future; in addition, previous decisions can not be revoked. Hierarchical servers refers to the servers which form a hierarchy of capability. In this way, a job which could run on a given server may run on any higher server, too. This problem is an important practical paradigm for which involves different classes of service and it has interesting relationship with the problem of scheduling on related machines. They investigated several variants of the problem such as existing of weighted jobs, possibility of job fraction, and temporary jobs which may depart the system after specific amount of time. For each of these variants, some deterministic or randomized algorithms have been presented and their lower bounds and competitive ratios have been analyzed.

In [122], authors addressed non-clairvoyant online job scheduling in Grids. Their grid model consists of a large pool of identical CPU's within several machines. Jobs are submitted over time and cannot be executed on different machines once scheduled. For the problem of $GP_m|size_j|C_{max}$, their result proved that the guaranteed makespan by List scheduling in multiprocessing system, i.e. $2 - \frac{1}{m}$, is not achievable for Grid environment unless $P = NP$. In other words, list algorithm remarkably preforms worse in Grids comparing to its performance in traditional distributed systems. They improved conventional list scheduling and presented an approach that uses several lists for Grid environment. Their analysis showed that the approximation factor for the improved version is 3 in the concurrent-submission case and 5 in the over-time submission case.

Authors in [123] presented a dynamic, policy-based, and adaptive allocation algorithm which uses a two-layer hierarchical structure respecting the main properties of Grids for the problem of $GP_m|r_j, size_j|C_{max}$. Their analysis showed a decent bounds of the competitive ratio for certain workload pattern.

Authors of [124] address the problem of online scheduling of parallel jobs on available resources in a grid environment in a non-preemptive non-clairvoyant manner. They assumed there are some tools that can predict the unknown execution time of each task by using the history of tasks running time. To analyze the behavior of several resource allocation strategies, they evaluated different performance metrics, such as mean waiting time, mean slowdown, utilization, throughput, and sum of weighted completion times ($\sum p_j \times size_j \times C_j$[16] which are commonly used to show the goals of different stack-holders of a typical Grid environment (i.e., end-users, resource provider, and system administrator, respectively). They presented a deep analysis of performance evaluation of fourteen allocations strategies. The simulation results revealed that MPL and $LBal_S$ allocation strategies are more robust and can outperform the rest, for the objective of either minimizing the total waiting time, total weighted completion time, or total system slowdown. Another interesting observation made by these authors is that the users' run time estimate can not help to improve the performance of any strategy. So, they suggested that a simple scheduler with minimal requirement of information can achieve a significant performance in practice.

There have been several resource management systems for Grid environment introduced in the last few years with different architectural design, scope, scheduling policy, and platform target, including (but not limited to) 2k [126], AppLeS [127], Condor [101], European DataGrid [128], Globus Toolkit[129], Nimrod/G [130], and PUNCH [131]. For a deep understanding of characteristics and architectural concepts of above grid resource management systems, the interested rereads are referred to [132–139]. Further, a complete overview of the algorithms proposed for the workflow scheduling problem on grid systems can be found in [140–146]. Several heuristics have been proposed for scheduling of BoT applications on Grid, too; such as Max-Min, Min-Min, Round robin order Replication[147], Sufferage [148], XSufferage [149], and Storage Affinity [150]. Almost all of these algorithm can not guarantee the quality of schedule in practical situation, especially because they are too dependent on the prediction information about tasks and performance

[16] Refer to [125]) for getting more information about advantages of using this metric

of grid resources, which is rarely available. A brief description of the most well known resource allocation strategies in Grid can be found in Table 2.2.

Lee and Zomaya in [151] introduced two novel scheduling algorithms, called Shared-Input-data-based Listing (SIL) (target application is data-intensive BoT) and the Multiple Queues with Duplication (MQD) (target application is computationally intensive BoT) in grid platform to tackle large-scale bag-of-tasks scheduling problems. The primary strong point of both mentioned algorithms is the scheduling decision is made without fully accurate performance information. SIL is composed of two main phases of "Task Grouping Phase", and "Scheduling Phase". The former tries to categorize tasks into a collection of smaller lists regarding the patterns that are detected on data sharing observation. It then assigns these lists to appropriate sites. The latter step dynamically assigns tasks to different hosts. MQD Algorithm make the scheduling decisions by taking the most recent workload motif into account within the hosts. To address the the dynamic nature of the grid (such as resource failure and inefficient system utilization), it also uses a task duplication approach to both shorten the makespan and improve system utilization. Experimental results showed that both SIL and MQD outperforms many previously well-known Grid scheduling algorithms (i.e., SA, Max-Min, Min-Min, Sufferage and RR) in terms of both reducing the average makespan and handling resource failures.

Anglano and Canonico in [152] analyze the scheduling problem of a set of competing BoT applications that are submitted for execution on a Desktop Grid for the goal of minimizing both the makespan and the entire turnaround time[17]. The computing power provided by desktop grid make it particularly a perfect platform for execution of BoT applications. Due to the fact that collecting resources' or applications' information is too difficult or inaccurate, they presented a knowledge-free approach, *i.e.,* no information concerning the resources or the applications is available to the scheduler when jobs submitted to the system. Their main results confirmed that (1) a simple scheduling policy such as FCFS often results in unsatisfactory performance, and (2) there are suitable knowledge-free scheduling policies that can obtain a performance level which is comparable to that of knowledge-based strategies. Their proposed strategy consists of two main independent steps of "bag selection" and then "scheduling of individual bag". They used WQR-FT policy for scheduling of individual BoTs. WQR-FT strategy is an extended version of the classical *WorkQueue* algorithm by adopting check-pointing

[17]Turnaround time can be defined as the sum of the waiting times

and automatic resubmission of failed tasks (more details can be found in [153]). To evaluate the effectiveness of the proposed policy, authors of [152] performed an exhaustive study to compare its performance with different knowledge-free bag selection policies, such as FCFS-Excl, FCFS-Share, Round Robin, RR with No Replica First, and Longest Idle. Their findings declare that there is not a single strategy that can perform better for all different workload types. Specifically, FCFS-based strategies can manage better those workloads with small task granularity, while Round Robin based strategies perform better for larger granularity values. This result is consistent with our findings which will be presented in details in Sections 5 and 6.

Benoit *et al.* in [154] presented the first efficient online scheduling technique for multiple bag-oftasks applications on a master-worker platform which composed of heterogeneous resources [18]. In the online model there is not any beforehand static information about the workload distribution characteristics before the actual execution of tasks. Their aim was to minimizes the maximum stretch[19] of the concurrent applications[20]. The big picture of their idea is to first derive an optimal solution for the offline setting by solving a linear program system, and then this solution has been adapted and recomputed for online scenarios to react dynamically upon the arrival of a new tasks. The experimental comparisons showed a significant performance improvement against classical existing greedy heuristics such as FIFO, shortest processing time, shortest remaining processing time, round-robin, minimum completion time, and demand driven.

A notable work in [156] analyzed the scalability of Bag-of-Tasks applications running on dedicated, master–slave platforms which composed of homogeneous resources. They could elaborately define a lower bound on the reachable scalability on this infrastructure. Scalability can be categorized as either "fixed" or "scaled" problem size[21] [157, 158]. Both of the above mentioned scalability types have been analyzed in their work. They proposed "input file affinity" factor to measure the related scalability of an application. It is an intrinsic feature of the given application and quantifies the degree of input files' sharing among application tasks. By correctly using this measure, they presented a new two-phase

[18] It is quite similar to classical client-server model

[19] Stretch can be interpreted as the maximum ratio of the actual time that an application has spent in the system to the entire time that this application would have spent if executed alone[155]

[20] The aim is equivalent to minimize the largest slowdown of a given job which caused entirely by its concurrent execution[155]

[21] also known as iso-efficiency metric

scheduling algorithm called "dynamic clustering" to improve the scheduling scalability of running BoT applications on master–slave system. Their new scheduling strategy does not need to have *a priori* information about either the actual or the estimated task execution times, i.e., it is totally "oblivious" to running time of tasks. The result of comparing the scalability of their proposed algorithm to some other well-known classic solutions, such as WorkQueue with Replication (WQR) [153], min–min, Sufferage [159], and many others has shown that their proposed algorithm is significantly more scalable than conventional heuristics, even those which are not oblivious to task execution times.

Casanova *et. al.* in [160] studied the problem of non-clairvoyant scheduling of multiple BoT applications when tasks' characteristics follow an unknown distribution with the aim of maximization of system throughput. They considered star-shaped platforms made of a master machine linked to several heterogeneous worker processors, whlie links' communication pattern followed the "bounded multi-port" model presented by Hong and Prasanna[22] [161]. Their approach was to first drive a polynomial-time approximation scheme for the clairvoyant case (via modleing the problem as a linear program), and then splitting each application into several virtual applications such that members of same virtual application can be considered to have similar workload size by a factor of $1 + \epsilon$ (semi-identical partitioning). Through extensive experiments as well as theoretical analysis, they have demonstrated that their heuristic always reach significantly better performance than existing approaches such as on-demand heuristic, even in the very difficult non-clairvoyant context.

Bertin *et. al.* in [162] have proposed a fair optimal distributed scheduling algorithm for running multiple BoT applications with arbitrary communication-to-computation ratio on a master-worker Grid platform with the aim of maximizing throughput. The basic idea of their work was to convert the original problem into the well known optimal flow control of multi-path routing problem, and then applying the Distributed Lagrangian optimization[23] (DLO) methodologies. This solution gives a distributed self-stabilizing algorithm to control the supply and demand dynamically. While DLO has been already utilized extensively in the networking systems, its usage is very new yet appealing in the context of resource sharing problem due to the fact that designer can achieve the optimal solution via choosing among several fairness criteria in a hierarchical and distributed fashion.

[22] In this model, a processor can be involved simultaneously in several communications, however there is a limit for the sum of communications bandwidth per node (both upload and download)

[23] including usage of distributed gradient descent method

Their general proposed approach is based on following three steps. First, the original problem has been modeled as a concave non-linear maximization problem. Second, partial derivatives[24] has been solved regarding to each primal and dual variables; and finally, a distributed algorithm has been designed to implement the structure of optimal values suggested by the previous step. It also has been shown that the proposed algorithm is very efficient and converge in a acceptable running time. It performs very efficient for large and complex grid environment with lots of heterogeneous resources and combination of applications with CPU-bound and network-bound features. In chapter 5 and 6, we will exploit a similar approach based on Lagrangian optimization for the problem of BoT scheduling in Cloud platform.

2.9 Resource Allocation in Cloud

Cloud computing paradigm allows a cost-effective solution for dynamic provision and server consolidation to improve the utilization of system while keeping the total cost low. Virtual machine technology, as a building block of this new technology, can provide isolation, consolidation, and live migration of different workloads in an efficient way. There are two extreme models of "datacenter virtualization" and "infrastructure provision" from business point of view to cloud[25]. In the "datacenter virtualization" model, enterprise companies normally understand the cloud as an extension of virtualization in the datacenter. Their expectation from an infrastructure automation tool is orchestrating or simplifying the management of the virtualized resources. In this philosophy, which is mainly supported by VMware, the whole data center with its legacy applications is moved into virtualization infrastructure by building a private cloud on top of existing virtualized system. In the second model, however, the cloud management tool can be considered as a set of simple APIs to provision and supply virtualized resources on-demand. This suitable model for building enterprise public cloud has a simple management tools of virtual resources for launching new applications on IaaS platform.

Amazon Elastic Compute Cloud, Microsoft Azure, Google Application Engine, Eucalyptus, CloudStack, OpenNebula, and OpenStack are examples of well-known cloud computing solutions. Amazon EC2 is a classic standard IaaS model

[24] Lagrangian function
[25] Sometimes "Cloud-in" and "Cloud-out" terms are used instead

where enables a user to use a rental service of re-sizable virtual compute capacity over the Internet via API's[26] to start/stop instances with a customized pre-installed operating system. Virtualization in EC2 is doable by executing Xen on top of the physical machines[163].

Eucalyptus[27] is a Linux-based software framework dedicated to bring a cloud computing environment into companies[164]. It enables pooling compute, storage, and network resources that can be dynamically scaled up or down as application workloads change. Eucalyptus has a set of API's that is compatible with Amazon EC2, makes enterprises convince to go the hybrid cloud model by outsourcing the workload into the Amazon public cloud whenever is necessary. This software has six main parts as: "Node Controller" (NC) which hosts the virtual machine instances and manages the virtual network endpoints; "Cloud Controller" (CLC) that offers EC2-compatible interfaces; "Walrus" which is a put/get storage service and offers persistent storage to all of the virtual machines (similar to S3); "Cluster Controller" (CC) which manages instance execution and service level agreements per cluster; "Storage Controller" (SC)which manages block volumes and snapshots to the instances within its specific cluster (equivalent to AWS EBS); and the optional "VMware Broker" which physically runs on top of CC and provides an AWS-compatible interface for VMware environments. For a detail explanation of architecture of the Eucalyptus system, the interested reader is referred to [165].

Apache CloudStack and OpenNebula are two open source cloud computing softwares for providing a highly available scalable cloud computing platform service [166, 167]. They use existing hypervisors technologiess such as KVM, vSphere, and XenServer for deploying and managing large networks of virtual systems. In addition to components for accounting of network, compute and storage resources, CloudStack and OpenNebula possess native API's as well as Amazon S3/EC2 compatible API and primary/secondary storage support to build private, public and hybrid implementations of infrastructure as a service. The interested reader can find more detailed features and key functionality of CloudStack and OpenNebula in [166, 168], respectively.

OpenStack is an open source cloud operating system founded by Rackspace and NASA, and with partnerships with big companies such as AT&T, HP and IBM and more than 200 companies, for building public or private enterprise scale IaaS clouds. It enables enterprises and service providers to control, provision and

[26] web service interfaces
[27] Elastic Utility Computing Architecture for Linking Your Programs To Useful Systems[164]

manage large pools of compute, storage and networking resources within a data center by using a web-based administrative control dashboard. Its distributed and asynchronous architecture provides flexible designing an scalable cloud with ability to integrate with legacy systems and third party technologies. Having no proprietary hardware/software pre-requirements, OpenStack is designed to operate with multiple type of hypervisors, including KVM, LXC and XenServer. By offering high-performance computing (HPC) for intensive workloads and scaling vertically and horizontally, it can work nicely with Hadoop for big data needs, too. Other key features include petabytes object storage, live VM management via OpenStack image service , VM image caching, role based access control (RBAC), resource utilization quotas, Local Area Networks management, and several other useful features [169, 170].

Oprescu *et al.* in [171] presented BaTS to address the problem of executing bags of tasks in the cloud with the objective of minimizing the completion time under budget-constrained. Unlike similar studies with the same objectives, BaTS does not require *a priori* knowledge and dynamically learns application performance during runtime. In their task and machine model, they assume that the tasks preemption is allowed, and all machines belong to certain categories (like EC2 "Standard Large"). Their proposed algorithm needs no advance information about task completion times and can learn the application throughput at run-time. The basic idea is to anticipate an average value for the task execution time of each group of tasks using an initial sampling phase , and then apply a cumulative moving average mechanism. Similarly, their system estimates the average speeds of the available machines. Then, their approach formulate the problem of the best affordable makespan meeting cost constraints via a "non-linear integer programming" model (a Bounded Knapsack Problem), and finally solve it by means of dynamic programming approach.

Their work differs from our system in several ways: First, their approach assume that preemption with resume is allowed at arbitrary times. This turns the problem out to be simpler comparing to the case of non-preemption tasks in our study. We explain in more detail later how an optimized solution can be reached if preemption with resume is allowed. Second, they only introduced a heuristic to meet the objectives, we try to find the approximation factor of our solution, too. In addition, by delivering a set of Pareto frontier solutions, our approach provide the user with full flexibility of time-cost selection pairs, rather than forcing him/her to choice first the budget. Finally, while they only addressed the single public

cloud environment, our work exploits a hybrid cloud model including the local computation resources.

Authors of [172] presented a portfolio scheduling to address the problem of execution of long running time scientific applications on IaaS cloud resources. Several studies demonstrated that there is not a unique scheduling algorithm able to perform well against the various characteristics of scientific workloads. So, authors of [172] proposed to select the most suitable policy dynamically from a set of different available scheduling policies, called portfolio, based on the current workload or system conditions. The utility function to represent user experience in this work is quite similar to the p-norm model which we introduce in Chapter 4. The idea of portfolio scheduling is quite interesting and can be used with conjunction of our work, too. One drawback of this idea is that evaluating all policies may need a long time to accomplish in practice. Secondly, tackling bi-objective utility function has not been addressed in the above-mentioned work.

Authors of [173] introduced a budget-constrained scheduling framework, called *BaTS*, for executing heavy-tailed bags of tasks on Amazon EC2 platform. Likewise our framework, BaTS framework prodece a list of makespan/cost choices based on the estimation values, and delivers it to the user. Their idea, which is different from us, is to replicate running tasks onto multiple idle machines, however. The selection process seems to be very challenging and must affect significantly on the overall performance of the system. Authors claimed that tasks would only replicate on the idle machines, some questions arise here. For example, what is the optimal time framework that BaTS framework should wait to consider a machine as be an idle one. If this should be less than one ATU framework (i.e., one hour), then it seems that the idea of replication is pretty useless. Because almost all of the long tailed tasks will take more than one hour to accomplish. It means that replication of long running time tasks increase the total cost of scheduling. It seems that migration of task is a better idea than its replication if the current machine that currently execute the task is not suitable. In our work, we try to estimate the optimum value for the workload of each resource initially to avoid idle machines in the case of heavy-tailed tasks. However, we admit that our idea cannot cover all scenarios, too. This area of research needs further investigation and study, we assume.

Authors of [174] presented SpeQuloS framework to enhances the Quality of

Service of running BoT applications on cest effort distributed computing infrastructures[28] with the main objectives of reducing the total execution time as well as improving the stability of scheduling process. The main idea behind the work is to monitor continuously the execution of tasks and dynamically get fast and reliable resources from public cloud whenever the situation becomes critical. Several strategies have been proposed for decision of how and when resources should be provisioned from cloud, too. There are some issues remained unanswered in this report. Characterize of tail effect is among most challenging one. The metrics that have been defined should be examined thoroughly in more platforms. In addition, applying the idea presented in SpeQuloS framework on hybrid cloud must be carefully adjusted with the characteristics of new environment. Finally, the idea must be enhanced to comply with the multi-objective schema, for example by using similar methods suggested in our framework.

Authors of [175] investigate several leasing strategies from a broker's perspective, and propose a framework to minimize the rental cost of cloud resources by by finding a good combination of both on-demand and reserved instances for executing of a set of independent jobs with the assumption of knowing all characteristics of tasks in advance. They have examined performance of their framework against many job or VM selection strategies such as FCFS, round-robin, LJF, and so on to evaluate the best reservation plan. The idea of making use of both on-demand and reserved instances to reduce cost is pretty good and we consider to apply this idea in our future plan. Their work differs from us in other aspects such as reaching to a trade-off between money and time from users point of view, and the process of suggesting multiple optimal points to the user to choose from. Our work covers the non-clairvoyant case as well.

2.10 Summary

This chapter provided a brief background knowledge for readers to follow the rest of the thesis. The background contains cloud computing and virtualization concepts, features of scheduling theories, and introducing different methods of resource allocation in traditional and modern HPC systems. In particular, we emphasized

[28] BE-DCI is an infrastructure that provides not being used computing units without any warrant that the computing resources could be available during the application execution, BE-DCI may include Desktop Grids or Cloud Spot Instances.

the Bag-of-Tasks (BoT) scheduling because it has become more common in our research topic. The concepts of these features will help to enrich the understandings of the system, the modleing and the problem we are tackling in future chapters.

Table 2.2: **Some well known Grid resource allocation strategies**

Strategy	*Description*
`Random`	Allocates job to randomly the admissible site
`WQR`	Being unaware about the length of tasks, WorkQueue with Replication policy schedules the tasks of in an arbitrary order, to execute in the available processors
`Max-Min`	The task with maximum earliest completion time among all of the unscheduled tasks with least completion time over all of the hosts is allocated fairly to the host on which the minimum earliest completion time is expected
`Min-Min`	Selects the task that has the shortest earliest completion time and allocates it the host on which the minimum earliest completion time is expected
`Sufferage`	Schedules a task with longest value of difference between its earliest completion time and its second earliest completion time. In other words, it chooses the task that will be most penalized if is not allocated to the most favorable machine but on its second most favorable box
`XSufferage`	Allocates a task with the largest value of earliest site-level completion time and its second earliest site-level completion time
`Storage Affinity`	Aiming at minimizing data transfer, it determines assignments based on the the amount of the input data already stored in the site
`ML`	(Min-Load) Assign job to node with the least load per processor
`MPL`	(Min-Parallel-Load) Assign job to node with the lowest parallel load per processor
`MLB`	(Min-Lower-Bound) Strategy chooses the node with the least possible lower bound of completion time of previously assigned jobs, that is the node with the lowest work per processor. The value normally provided by the user at job submission, or estimated execution time is used.
`MCT`	(Min-Completion-Time) the earliest possible completion time is determined based on a partial schedule of already assigned jobs
`MWT`	Allocates a job to the node with minimum average job waiting time
`MST`	Allocates a job to the node with earliest start time for this job
$\texttt{LBal}_S$, $\texttt{LBal}_T$, $\texttt{LBal}_W$	Allocates job to the node with the least standard deviation of job processor/job execution time/job work requirementsD per processor (considering all nodes) after assigning job j
$\texttt{MWWT}_S$	Allocates a job to the site with minimum average job weighted waiting time
$\texttt{MSWCT}_W$	Allocates a job to the site with minimum sum of weighted (work) completion time

Chapter 3

Motivation Example and System Model

Most complex problems arising in modern environemtns involve many conflicting objectives to be optimised. Hence, mathematical decision models which represnt the actual decision context by explicitly taking distinct goals into account are more popular than models which aggregate all of them into a single indicator. Multi-objective models make it understandable to show the conflicting nature of the objectives and the possible satisfactory tradeoffs. An efficient Pareto optimal solution is a feasible solution for which no improvement in all objective functions can be achieved. In other words, an improvement in one objective function is only possible in cost of degrading one (or more) other objective function value(s). The study of multi-objective optimization problems normally requires finding the set of Pareto-optimal solutions, by doing either a full computation of Pareto solutions or by computing a representative sample, the method that we follow in this thesis.

On the other hand, uncertainty in execution time of tasks is an intrinsic characteristic of real-world applications. It is generally impracticable that decision aid models could capture all the relevant necessary information in advance to find an optimal solution. In this context, it is important to provide decision makers with a robust solution. This concept is linked to guarantee an acceptable performance even under changing conditions and uncertain input data revealed in future [176]. In this chapter, we present some of the related works on multi-objective resource management in Grid or Cloud environment. We discuss briefly the robust region of the optimal Pareto frontier of previous works whenever applicable. Then we show how uncertainty in characteristics of application can affect a robust solution

by presenting an example. We then provide the model of both Cloud and target application used in this study.

3.1 Bi-objective Analysis in Resource Assignment

Although a big portion of previous researches on task assignment in the Grid or Cloud environment relied on the hypothesis that computation is free, we strive to do literally consider the cost of execution in the target platform. In this way, we try to consider optimizing of both the monetary cost and the makespan of BoT scheduling simultaneously in our study.

Bi-objective analysis in resource assignment problems have attracted considerable attention in past researches. In a notable work reported in [177], authors develop a novel scheduling algorithm to optimize both the workflow latency as well as satisfying throughput requirements by using different techniques such as pipeline parallelism, clustering and task replication. In [178] a handful algorithms have been presented that approximates the Pareto-curve of makespan and reliability objectives to tackle the problem of task graphs' scheduling in heterogeneous parallel machines while each processor has a failure probability. They showed that the product of objective functions is a crucial factor to let the user choose a trade-off between maximizing the reliability and minimizing makespan simultaneously.

In [179], authors investigates the scheduling of workflow applications on grids by considering both completion time and resource usage. Their algorithm, called *ADOS*, incorporates an effective rescheduling method to deal with the fluctuations of unforeseen performance. In [180], authors have studied the concurrent scheduling of Parallel Task Graphs (PTG) onto heterogeneous platforms, which is closely related to the scheduling problem of a multi-threaded programs on a multi-core system. Optimizing both fairness and average global makespan was the main goal of their study.

Saule and Trystram in [181] considered the problems of scheduling of a chain of tasks on heterogeneous processors and scheduling of independent tasks on identical processors for the objective of optimizing both efficiency and reliability. They drive a safety model for the reliability of the heterogeneous processors environment and propose some dynamic programming heuristics to solve the mentioend multi-objective optimization problem. Hirales *et al.* in [182] considered the problem of non-clairvoyant multi-objective workflow allocation strategies on a computational Grid where the scheduler has no knowledge of the real execution running time

of the ready tasks. Authors of [183] tries to maximize the expected amount of workload that gets computed by the assemblage of available remote homogeneous computers. Their algorithm cope elegantly with uncertainty within an assemblage risk such as unrecoverable interruptions by using check-pointing and work replication techniques.

Authors in [184] and [185] considered the problem of non-preemptive scheduling of independent tasks on a set of unrelated machines when the processing cost (c_{ij}) is considered and only a constant number of machine exists. The objective was to find a simple FPTAS solution to approximate Pareto curve of optimal schedule which is a trade-off between the makespan and the total cost. The proposed algorithm can successfully obtain an FPTAS for constructing an ϵ-Pareto curve[1] that dominates all others solutions approximately while its running time is polynomial regarding to both n and ϵ^{-1}. Other researchers also used Pareto frontier approximation technique for scheduling problem with multi-objective optimization. We recall that because the size of Pareto set is exponential, even determining whether a given point belongs to the Pareto curve is an NP-hard problem. Therefore, the idea of $\epsilon-$approximate Pareto curve, which is presented first in [186], is quite an acceptable approach for dealing with optimization problem with multiple objectives. Interested readers are referred to [186–193] for additional insight about the technique of two ways approximation of the Pareto set.

3.2 Deploying of BoT Applications in Hybrid Cloud

When deploying an applications into Hybrid Cloud environment, users have to decide compromising between the "total cost" and the "running time" of application. On one end, users want to run the application as quick as possible. On the other end, users want to pay less for accomplishing the application. However, in a hybrid cloud platform, quenching both desires is almost impossible. The fastest deploying solution is that one single machine is assigned for each parallel task to be run, which is clearly the most expensive solution, too. There is also a cheap solution that only uses the least expensive machine for running the tasks, which often results in an extremely time-consuming accomplishment. In these types of situations, the concept of user's utility function, $U(time, cost)$, has been widely used to represent the monetary value that users are willing to attach to job completion.

[1] An ϵ-approximate Pareto curve can be described as a set of solutions that for every other solution, the set contains a solution that is at least as good approximately (within a factor of $1 + \epsilon$) in all objectives

Assuming that the information about utility function is known by the resource allocator at the time of scheduling, then a qualified scheduler is the one that can effectively maximize user satisfaction. The concept of using utility function in the field of HPC scheduling research has been recently emerged, such as the works proposed in [194–200]. However, the fact is that real user preferences and priorities are too difficult to be expressed. Furthermore, arbitrarily complex utility functions could make the scheduling problem intractable. In fact, a large portion of previous important researches on utility-based resource allocation have restricted their attention to a simple linear shape of utility function. While, It has been demonstrated that real users have such complex utility functions that could not be effectively captured by linear shape or simple format [201, 202].

In Chapter 4, we propose a p-norm model that allow us to quantify users' utility function and their preferences in a more complex way. However, the exposure of users' preferences might not always be legitimate or feasible. Users may not be willing to provide details of their preferences in a real job submission due to concern about privacy or reduction of bargaining power. In addition, in many cases, users have been already bothered to put forward other information such as job run-time estimations, and are not interested in struggling to identify additional information for setting up the platform. We investigate how to generate possible Pareto-frontier scheduling points and asks the user to select a single scheduling point without forcing him/her to expose the utility function. This idea has been implemented in a PANDA framework [203], which will be discussed in detail later in Chapter 5.

Depending very much on structural processor components, it is almost impossible to found out the exact execution time of a tasks on on a computational resource before its actual running. However, there are still some hopes to provide an upper bound (such as WCET methodology [204]) or estimation function (which is normally depends on input characteristics), or use some performance estimation methods to predict the execution time of a given application on a specific machine. Many of these methods are based on either "task profiling", "machine benchmarking", or "statistical analysis" (or a combination of them) to provide an estimation of tasks' execution time [205–209].

Code profiling is a dynamic analysis that measures the run-time behavior of the source code segments or the underlying platform to collect data such as hardware counters, register traces, operating system's kernel, or interrupt traces, to understand program behavior. ATOM [210] is one of earliest instrumentation

framework for building program monitoring routines to collect traces and measure the performance of system. Analytic machine benchmarking which was first presented by Freund [211], and has been extended by many other researchers such as [212–214], is a powerful comparative tool to determine the relative performance differences between underlying machines via defining a number of primitive code types. Then, the analytic benchmarking data and the code profiling data can be combined to build an estimation for execution time. Statistical prediction methods treat the execution time as a random variable and try to predict it by analyzing the past historical observations by means of statistical techniques such as k-nearest neighbor nonparametric regression [215, 216].

In Chapter 4 we will present a Mont-Carlo based statistical technique for predicting the execution time of the submitted Bag-of-Tasks applications. In Chapter 6 we will show how we can use the output of several Code-profiling tools to effectively schedule a given BoT application with the unknown tasks running time [2]. The big picture of the idea presented in 6 is to use a combination of several estimator tools in order to generate all (possible) Pareto frontier scheduling points which can hypothetically be achieved. At the end, we give some heuristics (or approximation algorithms) to approach the user's selection of optimized scheduling as close as possible.

3.3 Motivation Example

We presents some scenarios here to depict what is the possible challenges that a typical users faced to submit his/her applications on cloud platform. We will discuss (1) the advantages of using hybrid cloud with deadline constraints BoT applications, and (2) how available estimation tools can be used to handle the uncertainty in task running times.

In the first scenario, consider running a BoT application/job with 36 tasks in a hybrid cloud which the attributes of both tasks and the platform shown in Tables 3.1 and 3.2. If the cost of running the job is only a concern and the user lacks advance knowledge on characteristics of both tasks and the system resources, the sole use of private cloud is a likely option scheduling approaches, such as a FIFO or random scheduler. One can easily calculate that the application will complete in 6.1 hours in the best case. However, suppose the case that user is constrained by a deadline of 2 hours to finish the whole application.

[2] also known as on-clairvoyant scheduling

Table 3.1: **Task characteristics for our motivating example. The number of tasks is 36. $\hat{p_{i,j}}$ denotes the estimation of the runtime of task i (in minutes) using tool j**

No. of tasks	Actual running time	Estimated by Tool 1	Estimated by tool 2
8	37	$30 \leq \hat{p_{i,1}} \leq 48$	$26 \leq \hat{p_{i,2}} \leq 43$
8	23	$20 \leq \hat{p_{i,1}} \leq 30$	$16 \leq \hat{p_{i,2}} \leq 27$
4	15	$13 \leq \hat{p_{i,1}} \leq 20$	$10 \leq \hat{p_{i,2}} \leq 17$
16	12	$10 \leq \hat{p_{i,1}} \leq 16$	$8 \leq \hat{p_{i,2}} \leq 14$

Table 3.2: **Resource characteristics for our motivating example.**

Resource Type	#resources	Relative speed	Relative cost/hr
Private	2	1	2
Public $Type_1$	5	1	1
Public $Type_2$	5	2	4

As one straightforward suggestion, the user can make use of public cloud resources facing the tradeoff issue between minimizing total cost and makespan. If the user has an accurate prior knowledge about both tasks running time and resources, and if he/she makes scheduling decisions focusing on only makespan by allocating an equal workload to each resource/machine, then in the optimal solution, job completes its execution in 61 minutes (a makespan of 61 minutes). However, accomplishing the tasks based on the least makespan policy leads to another challenge. Since most IaaS public cloud providers charge the user based on the usage in one one hour ATU[3], even though the achieved makespan is minimal, the user is charged for the full 2 hours of usage. If the user has no idea about his/her utility function, it is essential to have some tool to assist the user in the selection of computational resources to fulfill both deadlines and budgets.

Now, consider a scenario that the user has no prior knowledge about the task running times, but instead has access to two code-analyzing estimation tools, each of which produces an estimation result within ±30% of the actual running time. Nominal results are shown in the two right-hand columns of Table 3.1. many potential advantages would be realized providing the user decides to exploit these tools. However, several questions and concerns will arise, too, such as:

[3]Accountable Time Unit

- What it is the outcome of each possible scheduling strategy in terms of performance and cost?
- What is the best way to exploit these estimation tools?
- After the completion of each task, are there any feedback techniques available to enhance the performance of our scheduling algorithm?

Next chapters aim to provide reasonable answers to these and other similar questions.

3.4 Problem Analysis

To have a comprehensive mathematical model for describing an scheduling problem, one must present the modeling of machines, jobs or tasks, and the objectives aimed by the problem. This section presents the main models and terms used in the remainder of this thesis. We exploit a hybrid approach of using both public and private resources if the private capacity are not enough to guarantee the given level of performance. This can result in a reduction of total economical costs to run the application.

Public Cloud Model: Throughout this thesis, we use notation of Γ_u to refer the set of all available resources in public IaaS cloud. We also assume that provider offers different k instance types. Machines in each instance types differs from other type because of difference either in type or cost. So, we can assume that the computing capacity of a resource of type $i \in \{1 \dots k\}$ is guaranteed at level s_i. Computing capacity here can be imagined as the average CPU speed or the amount of RAM assigned to the machine during the application execution, and is referred as resource capacity, resource performance or resource speed throughout this thesis interchangeably. Machines of each type of the public cloud is connected with monetary costs c_i, too. We assume the amount of c_i does not change during application life-cycle phase. Today's cloud provider charges users for renting resources based on their usage during an specific period of time known as **a**ccountable **t**ime **u**nit or **ATU**. We adopt *one hour* as ATU in this study as many cloud providers (including Amazon) calculate the usage hourly. Additionally, there might be a limitation on the total number of resources that a user can rent from a particular type. If such a limitation exists, we denote it with L_i. For example, L_i is limited to be 20 in the current Amazon EC2 environment for a regular user.

Private Cloud Model: The organization's infrastructures and/or data-center resources can be defined as the private cloud, whether hosted on either inside or outside of the organization. We use Γ_v to show the pool of resources within the private cloud, like CPU, Memory, *etc.* Assuming that there are different k' resource types in the private cloud, the capacity of private cloud resource is shown by $s_{i'}$ for $i' = 1 \dots k'$ (or $i' = k \dots k + k' - 1$). Because the organization has to buy and manage the private resources by itself, it sounds reasonable that a cost can be associated with each of these resources, which might be even higher than the cost of public resource with the same capacity. In Section 5.8.2 we will discuss famous techniques can be used to analyze the total cost of ownership of private resources. We use $c_{i'}$ to refer to the cost of the private resource i' based on ATU period, for $i' = 1 \dots k'$. It is worth to mention that despite its high initial cost and unpredictable running cost, private cloud provide normally more flexibility in terms of both customization and privacy. The public cloud resources can be added to or removed from the private system at anytime according to the decision of scheduling algorithm. Byzantine failures of public or private resources are not considered in this thesis, meaning that machines follow their running properly and tasks' execution is not at risk.

Bag-of-Task Application: We consider a BoT application, B, to consist of n independent tasks and be CPU-intensive. Here, the size of BoT application (n) is so large that overwhelm the capacity of resources provided by the private cloud. Each task $j; 1 \leq j \leq n$, required the running time of P_j to accomplish, providing that it is assigned to run on a resource with unitary speed, $s_i = 1$. Further, we assume that each P_j value is a random variable comes from an unknown distribution density F. We denote the mean and variance of this random variable by μ and σ^2, respectively. The processing times are considered to be non-negative and mutually independent. In our thesis, task preemption is not allowed, so once a task is assigned to run on a machine, the whole part of it must be executed without any interruption. If task j is assigned to run on a resource of speed s_i, and cost of c_i, then the processing time and the cost of executing task j is equal to $\frac{P_j}{s_i}$ and $\frac{c_i \times P_j}{s_i}$, respectively. As a user hires a public cloud resource in whole hour duration, the total cost to finish a set of tasks in a specific public cloud resource is $c_i \lceil \frac{\sum_{j \in J_i} P_j}{s_i} \rceil$, where J_i is the set of all tasks assigned to resource i. We use $|B|_R$ to represent the amount of time (in seconds) taken by a BoT application to complete its execution in a reference machine R.

The makespan of a BoT application is the time elapsed from BoT submission

until its completion. The performance metrics which are targeted in this thesis are mainly the monetary cost of all BoT tasks and makespan. The utility function of a user is a function of the above-mentioned performance metrics that quantifies the happiness of the user by using a particular strategy for running the BoT. Normally, users try to optimize their utility function. For example, if a user does not care about the makespan and solely interested in the cheapest strategy, she should provide the system a utility function that only includes the total cost of scheduling.

The problem to be solved is effective scheduling of the submitted application on hybrid cloud infrastructure respecting the user's utility function. To gain the maximum effectiveness of the hybrid cloud resources, this research aims to propose mechanisms to automatically determine the amount of resources of each type to be used in an optimal fashion. The second goal of this research is to provide the user with a set of Pareto-efficient scheduling points with dual objectives of minimizing total cost as well as make-span, and determine strategies to guarantee achieving the user's selection in practical situation. If the performance of an strategy is worse than or identical to another strategy for both total cost and makespan metrics, and strictly worse for at least one metric, then we said that the first strategy is dominated by the second strategy. The set of all strategies that is not dominated by any other strategy forms Pareto-frontier set. In other words, any approach that optimizes the utility function of user must be in Pareto-efficient set. We recall that previous researches in different contexts showed that even determining Pareto-efficient strategies are computationally intractable problems.

We introduce PANDA scheduling framework in Chapter 5 which answers what mixture of resources in public or private cloud must be used to achieve the optimum allocation schema respecting user's preference for different makespan-cost trade-off. This solution uses an FPTAS methods, means that it uses an approximate algorithm to reach a close optimal answer in the polynomial time in terms of both approximation factor and input size. An extended version of PNADA will be introduced in Chapter 6 for handling uncertainty in tasks running time with the similar approaches and aims discussed for original version of PANDA. We evaluate our framework through both simulations and real environments experiments, and show that it can performs better in comparison to scheduling strategies commonly used for cloud environment.

3.5 Summary

One of the main challenge that scientists face when try to execute a large scale Bags-of-Tasks application in hybrid computational cloud environments is the shortage of tools that offer Pareto-efficient strategies for different (even unknown) utility functions of users. In this chapter, we showed the importance of bi-objective analysis in resource assignment of hybrid cloud model regarding the user-defined utility function. Then, through a motivation example, we showed the main topics that our thesis is trying to deal with. We also provided a formal model for the environment we target in our thesis, including models for both the available resources as well as the submitted application. Finally, this chapter discussed briefly how our framework provides a clear understanding of the possible trade-offs of total cost versus makespan (Pareto-optimal solution) to execute user's application in the hybrid cloud environment.

Chapter 4

A p-norm model for Optimized Bot Application Deployment and Cloud Resource Allocation

4.1 Introduction

To dynamically expand the capacity of in-house computing systems, public cloud computing can be considered as a promising solution. Particularly, flexibility of public cloud resources with a pay-as-you-go pricing model enables tightly budgeted users, such as small organizations and individuals to "cost effectively" access massive computing resources. However, the reality is that the degree of performance gain is often not strongly correlated with the usage cost of these resources; this is particularly true when CPU-intensive applications run on cloud resources with the non-proportional cost to performance ratios (e.g., Amazon EC2 m1.small and c1.medium with such ratios of 1 and 5, respectively). This issue in scaling out is a major obstacle that must be resolved for the cost effective deployment of application into multiple cloud environment.

In this chapter, we present some resource allocation algorithms to enable the execution of BoT applications spanning beyond the private system/cloud (i.e., hybrid-cloud) by explicitly taking into account the cost efficiency—the cost to performance ratio. We give solutions of proposed objective function in both known and unknown running time of the given application. We also assume that both private and public cloud can possess several types of resource (resource heterogeneity). This complies with the practical situation which an in-house system can

evolve over time expanding it with more powerful machines and keeping legacy systems (delaying their decommissioning).

To cope with the "unknown running time" case, we develop a fully polynomial time randomized approximation scheme (FPRAS). Such an FPRAS algorithm takes two parameters of (1) $\epsilon > 0$ as an approximation factor, and (2) $\delta > 0$ as a confidence interval, and produces a solution that lies within a factor $(1+\epsilon)$ of being optimal with high probability (exact definition will be reported later in this section). The running time of an FPRAS algorithm is guaranteed to be polynomial in terms of job size (n or the number of tasks in a given job), the approximation factor (ϵ), and the confidence interval ($log(1/\delta)$). To address the resource allocation problem without any advanced knowledge about the processing time, we incorporate a Monte Carlo sampling method inspired by the work in [217] to estimate the average of tasks' running time.

In Section 4.2 we present our model for resource allocation problem. Section 4.4 details both our near optimal solution and resource allocation algorithm. Section 4.5 presents the results obtained from experiments as part of our algorithm evaluation followed by our conclusion in Section 4.6.

4.2 Problem Formulation

We describe our research problem by formulating the resource allocation problem as an optimization problem. The models for public cloud, private cloud, and Bag of Task application follow the model presented in Section 3.4.

- **Use Case Scenario**: A sample use case scenario of our proposed system (as depicted in Figure 4.1) can be described as follows: the user submits a Bag of Task job along with several other parameters (the details of these parameters, such as r and desirable weighting of each resource, will be described in the next section). If the tasks running time is not known in advance, the framework will perform an estimation phase, which will be discussed in detail in Sec. 4.4.3, to estimate the total workload submitted to the scheduler. Otherwise, the system jumps up to the next step of finding the optimal resource allocation.

  In the next step, the framework tries to solve a system of equations (given by Equations 4.14 and 4.16), in order to figure out the optimal workload value that must be assigned to each resource. The next step is answering the question of how to assign properly a set of tasks that does not exceed the optimal workload

which calculated in the previous step. To overcome this, we apply a FFD-based allocation algorithm which is given in details by Algorithm 2 and 3. Finally, every tasks will be deployed to run on the proper resource located either in private or public cloud. During the execution phase, some sort of performance monitoring actions can be done to report the user the overall system performance time to time.

4.3 L^p-form Cost Model and Cloud Resource Allocation Problem

When a user submits an application to be run using the cloud resources, there are two conflicting objectives of "minimizing the total cost" and "maximizing the performance" (or minimizing the makespan) that the user had to reach a compromise between them in a rational way. Let σ denote a particular resource allocation schema for a given application. Further, let $C(\sigma)$ and $T(\sigma)$ denote the total cost and the makespan reached by this resource allocation scheme, respectively. Comparably, assume $C_i(\sigma)$, $T_i(\sigma)$, and $\psi_i(\sigma)$ denote the corresponding cost, time (in terms of hour or ATU) and the assigned workload of resource i reached by σ. Here, resource i may belong to either public or private cloud.

By using preliminary calculus, the accuracy of the following equations is easy to be checked.

$$C(\sigma) = \sum_{i \in \Gamma_u \cup \Gamma_v} C_i(\sigma) \tag{4.1}$$

$$T(\sigma) = \max_{i \in \Gamma_u \cup \Gamma_v} (T_i(\sigma)) = \lim_{r \to \infty} \Big(\sum_{i \in \Gamma_u \cup \Gamma_v} T_i(\sigma)^r \Big)^{\frac{1}{r}} \tag{4.2}$$

$$C_i(\sigma) = \begin{cases} c_i \lceil T_i(\sigma) \rceil & \text{if } i \in \Gamma_u \\ c_i T_i(\sigma) & \text{if } i \in \Gamma_v \end{cases} \tag{4.3}$$

$$T_i(\sigma) = \psi_i(\sigma)/s_i \tag{4.4}$$

Equation 4.2 comes from the fact that in the L^p-norm space (Lebesgue spaces) of a vector z, L^∞-norm is equal to the maximum norm of that vector [218]. [1]

[1] L^p-norm (p-norm) of a vector z can be defined by $(\forall p \geq 1 \in \Re)$: $\|z\|_p = (|z_1|^p + |z_2|^p + \cdots + |z_n|^p)^{\frac{1}{p}}$. The L^∞-norm or Chebyshev distance is the limit of the L^p-norms when $p \to \infty$ which has the same definition as: $\|z\|_\infty = \max\{|z_1|, |z_2|, \ldots, |z_n|\}$

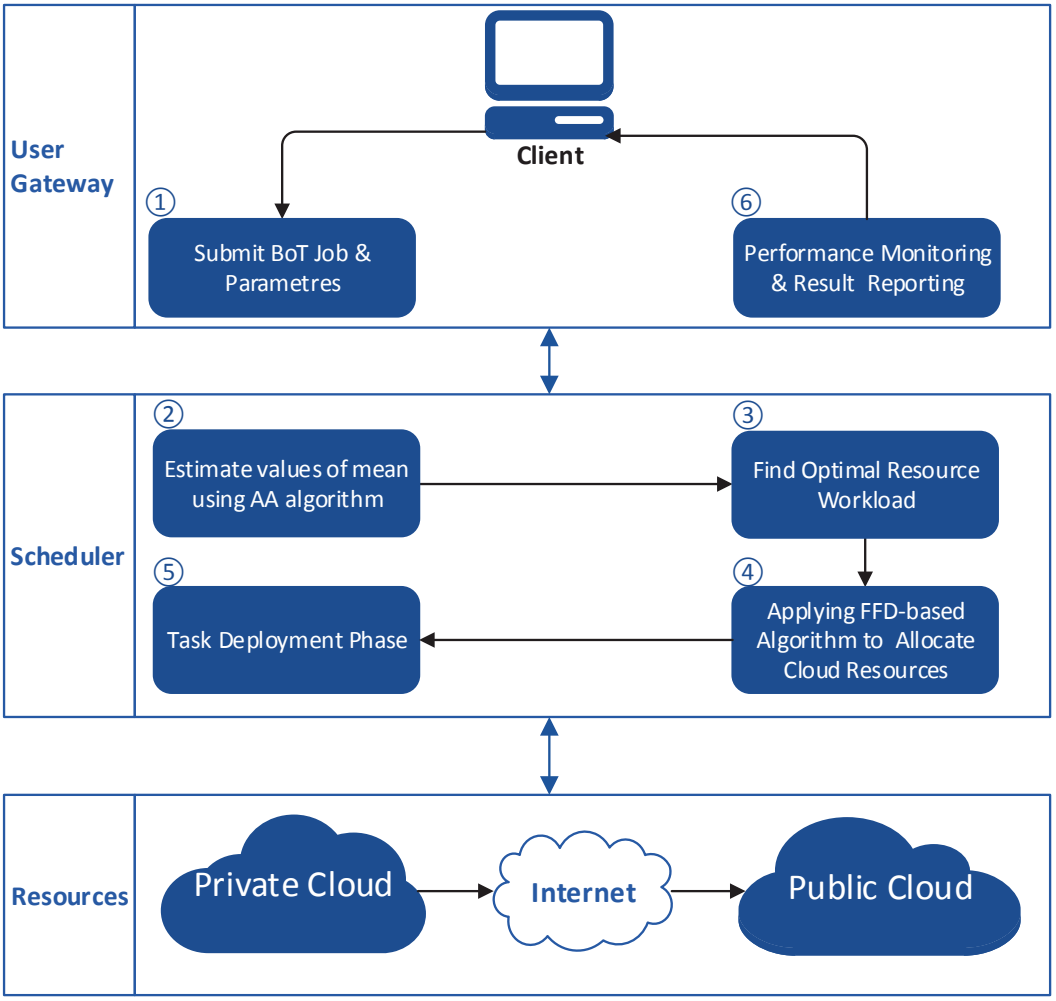


Fig. 4.1: **The storyline of process flow through our proposed framework. In this scenario, we assume that user submits a BoT job without any knowledge about tasks' running times.**

Inspiring from the definition of L^p-norm, we introduce the following user-tunable objective function to correlate two goals of minimizing the total cost and makespan together.

$$\min Z(\sigma) = \sum_{i \in \Gamma_u \cup \Gamma_v} \alpha_i \times (C_i(\sigma))^r ; r \geq 1 \tag{4.5}$$

In the above, α_i can be considered as a restriction weight that user assigns to each resource based on his/her previous knowledge or assumption. When α_i is large, it means that user does not have a desire for spending too much money to rent resources of type i. One reason can be imagined as the user has some

previous experience about the unexpected breakdown of that resource. Parameter r, on the other hand, enables user to express his/her preference for each criterion. Two values of r have been already extensively studied.

- $r = 1$: In this case, a large makespan scheduling with minimum value of total cost is achieved. The scheduling scheme normally comprise of a solution that assign all tasks to the most efficient resource types; i.e., the resource type that has the highest value of $\frac{s_i}{\alpha_i c_i}$ (turns Eq. 4.5 to be similar to Eq. 4.1).
- $r = \infty$: This case turns basically Eq. 4.5 to have an exact behavior as Eq. 4.1 has (means that they both have same minimum point). So, this leads to a solution that make the entire tasks' load equally (or wighted) balanced between all available resources. In this case, the result normally has the least value of makespan with a large total cost value.

In the next section, we reformulate Equation 4.5 as a binary non-linear optimization problem and then in Section 4.4.1, we provide solutions of different r values.

4.3.1 Formulation as Binary Nonlinear Programming

A Nonlinear Program (NLP) is an minimization (or maximization) problem such that a nonlinear "objective" function, $Z(\mathbf{x})$, must be minimized (or maximized) subject to some other (nonlinear) "constraint" functions that define limitations on the values of vector $\mathbf{x}$. In binary problems, each variable can only take on the value of 0 or 1. Optimization problem of Equation 4.5 can be restated as a binary nonlinear programming as follows:

$$\min Z(\sigma) = \sum_{i\in\Gamma_u, 1\leq j\leq n} \alpha_i c_i \lceil \frac{x_{i,j} P_j}{s_i} \rceil^r + \sum_{i\in\Gamma_v, 1\leq j\leq n} \alpha_i c_i (\frac{x_{i,j} P_j}{s_i})^r \tag{4.6}$$

$$s.t. \quad \sum_{i\in\Gamma_u\cup\Gamma_v, j=1\cdots n} x_{i,j} = n \tag{4.7}$$

$$x_{i,j} \in \{0,1\}; \quad \forall i \in \Gamma_u \cup \Gamma_v, j = 1\cdots n \tag{4.8}$$

In the above equations, the binary variable $x_{i,j}$ represents whether task j is assigned to resource i or not.

Integer programming problems are well known to be NP-hard. So, it is usually impossible to find any method to reach even a close optimal solution. Relaxation

is a traditional technique to deal with IP problems. However, a solution for the relaxed version of above mentioned "Binary Integer" problem (which obtained by replacing Constraint 4.8 with $0 \leq x_{i,j} \leq 1$) is most likely far from the optimal value.

To overcome this issue, we reformulate Equations 4.6–4.8 in order to remove the binary constraint. Let us define ψ_i as the total workload assigned to resource i. Then, we substitute the value of $\sum x_{i,j}P_j$ with the ψ_i to reach a relaxed version of the original problem as follows:

$$\min \sum_{i \in \Gamma_u} \alpha_i c_i \lceil \frac{\psi_i}{s_i} \rceil^r + \sum_{i \in \Gamma_v} \alpha_i c_i (\frac{\psi_i}{s_i})^r \tag{4.9}$$

$$s.t. \qquad \sum_{i \in \Gamma_u \cup \Gamma_v} \psi_i = \sum_{j=1 \cdots n} P_j \tag{4.10}$$

A solution for Equations 4.9 and 4.10 can be considered as a close solution of the optimal value for the original problem. In the following section, we present a solution for Equations 4.9 and 4.10 to approximate the optimal value of Equation 4.6.

4.4 Near Optimal Task Assignment

In this section we provide a solution for optimization problem stated by Equation 4.9 in two separate cases. First, we discuss if all tasks' running times are known in advance, and then we extend our solution in case that there is no beforehand information about the tasks' running time.

4.4.1 Solution for Known Tasks' Running Time

To solve the optimization problem stated by Equation 4.9, we first relax it by removing the integral condition. It can be seen that the relaxed version satisfies Karush-Kuhn-Tucker (KKT) conditions, so, we can apply Lagrange multipliers method to reach a system of equations to find the minimum value. Let us show the total number of available resources in both cloud by m, *i.e.*, $m = |\Gamma_u \cup \Gamma_v|$, and define β_i as $\alpha_i \times c_i / s_i^r$. Equation 4.9 can be relaxed as follows:

$$\begin{aligned} & \min \sum_{i=1}^{m} \alpha_i c_i \left(\frac{\psi_i}{s_i}\right)^r \\ s.t. \quad & \sum_{i=1}^{m} \psi_i = \sum_{j=1}^{n} P_j \end{aligned} \tag{4.11}$$

By introducing Lagrange multiplier (λ), the Lagrange function can be written as:

$$\Lambda(\psi_i, \lambda) = \sum_{i=1}^{m} \beta_i \psi_i^r - \lambda(\sum_{i=1}^{m} \psi_i - \sum_{j=1}^{n} P_j) \tag{4.12}$$

One can find the optimal solution of relaxed version by solving the following system of equations.

$$\nabla \Lambda_{\psi_i, \lambda} = \mathbf{0} \tag{4.13}$$

or equivalently:

$$\begin{aligned} \frac{\partial \Lambda(\psi_i, \lambda)}{\partial \psi_j} &= r\beta_j \psi_j^{r-1} + \sum_{i=1..m; i \neq j} \beta_i \psi_i^r - \lambda = 0 \; ; \forall j = 1 \dots m \\ \frac{\partial \Lambda(\psi_i, \lambda)}{\partial \lambda} &= \sum_{i=1}^{m} \psi_i - \sum_{j=1}^{n} P_j = 0 \end{aligned} \tag{4.14}$$

Let $\psi^* = \langle \psi_1^*, \psi_2^* \cdots \psi_m^* \rangle$ be the optimal solution of Equation 4.13 (or Equation 4.14). Each ψ_i^* shows the amount of workload in resource i in optimal case. The next step is to find a subset of tasks which add up exactly or as close as possible to the value of ψ_i^*. Such task assignment will be presented later in Section 4.4.4, after discussing our method of finding optimal value in the non-clairvoyant case.

4.4.2 Solution for Unknown Tasks' Running Time (Non-Clairvoyant Case)

In this section, we deal with the task assignment problem if all P_j's in Equation 4.10 are not known *a priori*. We present an FPRAS algorithm which consists of

two steps: estimation and task assignment. For a given optimization problem, an FPRAS algorithm can produce a solution within a factor of $(1+\epsilon)$ of the optimal value with a confidence interval of δ. The running time of such algorithm is polynomial in terms of input size, ϵ^{-1}, and $log(\delta^{-1})$. Naturally, an FPRAS scheme can be considered as the most efficient randomized algorithm for tackling NP-hard problem in the stochastic manner.

4.4.3 Estimation

To solve the system of equations given by Equation 4.14, we need to estimate the unknown value of $\sum_{j=1}^{n} P_j$. To this end, an estimation procedure based on a well known Monte Carlo sampling method called $\mathcal{AA}$ algorithm [217] can be employed. This sampling algorithm uses the minimum possible number of experiments to predict the average task running time, $\hat{\mu}$, (which is equivalent to $\sum P_j/n$) and satisfies the following condition.

$$\mathbf{Pr}[\mu(1-\epsilon) \leq \hat{\mu} \leq \mu(1+\epsilon)] \geq 1-\delta \tag{4.15}$$

Here, μ is the actual average value, $\hat{\mu}$ is the estimated value of μ, ϵ is the the approximation factor and δ is called the confidence factor.

The estimation procedure is presented formally in Algorithm 1. It consists of three main steps. Each step can be recognized by a *while* loop. In the first step, it produces an initial estimation of $\tilde{\mu_Z}$. In the second step, it determines the number of experiments needed to produce $\hat{\rho_Z}$ as an estimation for unknown value ρ with a probability of at least $1-\delta'$. The last step (the third *while*) takes both of previous outputs, $\tilde{\mu_Z}$ and $\hat{\rho_Z}$, to set the minimum number of experiments needed to be run on the private cloud to produce $\hat{\mu}$ which is an (ϵ, δ)-estimate of μ_Z.

After finding an estimation of μ, we replace the value of $\sum_{j=1}^{n} P_j$ (in either Equation 4.10 or 4.14) by the estimation value of $\hat{\mu} \times n$ as follows:

$$\begin{aligned} \frac{\partial \Lambda(\hat{\psi}_i, \lambda)}{\partial \hat{\psi}_j} &= r\beta_j \hat{\psi}_j^{r-1} + \sum_{i=1..m; i\neq j} \beta_i \hat{\psi}_i^r - \lambda = 0 \; ; \forall j = 1..m \\ \frac{\partial \Lambda(\hat{\psi}_i, \lambda)}{\partial \lambda} &= \sum_{i=1}^{m} \hat{\psi}_i - \hat{\mu} \times n = 0 \end{aligned} \tag{4.16}$$

Let us show the solution of the Equation 4.16 by vector $\hat{\psi} = \langle \hat{\psi}_1, \hat{\psi}_2 \cdots \hat{\psi}_m \rangle$. We will continue by giving our task assignment in the next section.

Algorithm 1: Estimation of μ based on $\mathcal{AA}$ algorithm

input : ϵ, δ, n, $\Gamma_v \cup \Gamma_u$, B;
output: $\hat{\mu}_Z$; //an estimation of average tasks' running time in B
begin
  $S \leftarrow 0$; $c \leftarrow 0$; $\epsilon' \leftarrow min\{\frac{1}{2}, \sqrt{\epsilon}\}$; $\delta' \leftarrow \delta/3$;
  $\Upsilon_1 = 1 + (1+\epsilon')\frac{2.87ln(2/\delta')}{\epsilon'^2}$;
  while $S < \Upsilon_1$ **do**
    Pick a random integer number, h, in range [1,n];
    Run task indexed by h, T_h, in private cloud;
    $S \leftarrow S + P_h$; //P_h: running time of T_h
    $c \leftarrow c + 1$;
    $B \leftarrow B - T_h$;
  end
  $\tilde{\mu}_Z \leftarrow S/c$;
  $\Upsilon_2 = \frac{5.75ln(2/\delta')}{\epsilon'\tilde{\mu}_Z^2}$;
  $S \leftarrow 0$; $c \leftarrow 0$;
  while $c < \Upsilon_2$ **do**
    Pick two random integer numbers, h_1 and h_2, in range [1,n];
    Run tasks T_{h_1} and T_{h_2} in private cloud;
    $S \leftarrow S + (P_{h_1} - P_{h_2})^2/2$;
    $c \leftarrow c + 1$;
    $B \leftarrow B - \{T_{h_1}, T_{h_2}\}$;
  end
  $\hat{\rho_Z} = \max\{S/\Upsilon_2, \epsilon.\tilde{\mu}_Z\}$
  $\Upsilon_3 = \frac{5.75ln(2/\delta').\hat{\rho_Z}}{\epsilon'^2\tilde{\mu}_Z^2}$;
  $S \leftarrow 0$; $c \leftarrow 0$;
  while $c < \Upsilon_3$ **do**
    Pick a random integer number, h, in [1,n];
    Run task T_h in private cloud;
    $S \leftarrow S + P_h$;
    $c \leftarrow c + 1$;
    $B \leftarrow B - T_h$;
  end
  $\hat{\mu}_Z = S/\Upsilon_3$
end

4.4.4 Task Assignment

In Sections 4.4.1 and 4.4.2, we discussed how to find the optimal workload received by each resource. In this section, we present two algorithms to find a subset of tasks which adds up as close as possible to the value of ψ^* (in case of known tasks' running time) or $\hat{\psi}$ (in case of unknown tasks' running time). This problem can be considered as a general case of bin packing, knapsack, or subset sum problem. For example, in the standard version of subset sum problem a set of integers and

an integer target t are given, and aim is to find a non-empty subset sum to t. All of the above-mentioned problems belong to $NP-complete$ class [219]. There are many well-known approximation algorithms in the literature [220–223] to solve these basic problems efficiently. They are often too difficult to understand or implement, however. Therefore, for the purpose of this research, we take advantage of a simple and fast approach based on First Fit Decreasing (FFD) algorithm [224], which generates reasonably a good outcome. FFD solution for bin packing problem can be described as follows: In the first step, we must sort both the items into their decreasing sizes and the bins according their capacities. Then the next item is put into the first bin where it fits. Slightly modified versions of this approach are given in Algorithms 2 and 3, where our aim is to decrease the amount of wasted capacity of resources as much as possible.

In the case of known task's running time, i.e., Algorithm 2, we do as follows: Knowing that each resource i should not receive a total workload more than ψ_i^* (in the optimal solution), we assign a task j to an already allocated resource i if its current workload plus P_j (the task's running time) does not exceed ψ_i^*. Applying this rule in addition to the rule of "selecting the most cost-effective resource first" can produce a near optimal solution.

Similarly, in the case of unknown task's running time, i.e., Algorithm 3, we pick a task from tasks' queue and add it to the next efficient resource if the expected value of this task's running time, i.e., $\hat{\mu}$, plus the current assigned workload of that resource does not exceed the value of $\hat{\psi}_{\mathbf{i}}$.

Variable $S[i]$ in these two algorithms holds the current assigned workload of each resource i. With these simple controls, we are able to assign a task to the most cost efficient resource provided there is sufficient residual capacity in it. By applying the similar approach given in [224], one can expect that both Algorithms 2 and 3 are constant approximation algorithms with approximation ratio of 11/9.

4.5 Experimental Evaluation

In this section, the performance evaluation results of the proposed approach using running BoT application on a hybrid-cloud infrastructure are represented. It follows the model we already discussed in Section 3.4.

The private cloud used in our study was composed of our in-house 4x10 core cluster with 2.4 GHz Intel Xeon processors, with a total of 256GB RAM.

Algorithm 2: Hybrid-Cloud resource allocation algorithm in known tasks' running time case

input : α_i's, r, n, $\Gamma_u \cup \Gamma_v$, and B with known T_j's;
output: resource allocation scheme
begin
  Sort resources by cost efficiency, i.e., $\frac{s_i}{\alpha_i c_i}$
  $S[i] \leftarrow 0; \quad \forall 1 \leq i \leq |\Gamma_u \cup \Gamma_v|$
  Calculate ψ_i^* from Equation 4.14;
  Sort tasks in B into their decreasing running time;
  while *B is not empty* **do**
    Pick the next available task T_j in B;
    if *there exists a resource i such that* $S[i] < \psi_i^* - T_j$ **then**
      Assign task T_j to resource i;
    else
      Assign all remaining tasks in B to resources in Γ_v using *List* scheduling;
    end
  end
  After detection of completion of task T_j on resource i:
    $S[i] \leftarrow S[i] + P_j$;
    $B \leftarrow B - T_j$;
end

Public cloud consists of instance types from Amazon EC2 cloud. We use four different instant types from EC2 which called "M1 Small Instance", "M1 Large Instance", "High-CPU Medium Instance", and "High-CPU Extra Large Instance", respectivley. Standard instances normally supply a balanced set of RAM and CPU capacity, suitable for a variety of application types. On the other hand, high-CPU instances possess more CPU capacity than RAM which make them best fit for CPU-intensive applications. Details of processing capacity and the cost of each resource are shown in Table 4.1. In this table, processing capacities are relative speeds based on the reference resource considered as public "m1.small" isntance. To mimic a real BoT workload, we created several sets of tasks which their characteristics are similar to the patterns reported by Iosup *et al.* [225]. Task characteristics are summarized in Table 4.2. The running time of each task is defined in terms of how much CPU time is required for its execution if we assign it to a standard small instance (m1.small) in Amazon EC2 public cloud. In Table 4.2, we categorize BoT applications into Short, Long and Mixture of both based on task running time. Symbols of $U(a, b)$ and $N(\mu, \sigma)$ show a uniform distribution with the minimum and maximum values of a and b, and a normal distribution with mean μ and standard deviation of σ, respectively.

Algorithm 3: Hybrid-Cloud resource allocation algorithm in unknown tasks' running time case

input : ϵ, δ, α_i's, r, n, $\Gamma_u \cup \Gamma_v$, B with unknown T_j's;
output: resource allocation scheme
begin
  Sort resources by cost efficiency, i.e., $\frac{s_i}{\alpha_i c_i}$
  $S[i] \leftarrow 0; \quad \forall 1 \leq i \leq |\Gamma_u \cup \Gamma_v|$
  Run Algorithm 1 to find $\hat{\mu}$;
  Calculate $\hat{\psi}_i$ from Equation 4.16;
  while *B is not empty* **do**
    if *there exists a resource i such that* $S[i] < \hat{\psi}_i - \hat{\mu}$ **then**
      Pick an integer number, h, from $[1..|B|]$;
      Assign task h, T_h, to resource i;
    else
      Assign all remaining tasks in B to resources in Γ_v using *List* scheduling;
    end
  end
  After detection of completion of task T_j on resource i:
    $S[i] \leftarrow S[i] + P_j$;
    $B \leftarrow B - T_j$;
end

Table 4.1: **Hybrid cloud settings with different per hour cost to performance ratios.**

Cloud	Instance Type	Normalized Proc. Capacity	Cost
	m1.small	1	$0.080
Amazon EC2	c1.medium	5	$0.165
US East (VA)	m1.large	4	$0.320
	c1.xlarge	20	$0.660
Private	4x10 Xeon Processors	10	$0.320

Table 4.2: **Different BoT workload characteristics used here**

Type	Task's Running Time (minute)
Short	U(0,8)
Long	N(36,6)
Mixture	N(12,3)

4.5.1 Hybrid-Cloud and BoT Application Setting

We compared our proposed algorithm with a heuristic algorithm which chooses tasks with some order (if possible) and assigns them to the next most cost efficient resource. A resource i is more cost efficient than another resource j if and only if $\frac{s_i}{\alpha_i c_i} > \frac{s_j}{\alpha_j c_j}$. This heuristic acts similar to the traditional **List** scheduling algorithm.

For different values of parameter r, we run the heuristic algorithm as follows: For $r = 1$, the heuristic algorithm tries to balance the tasks' load among all available resources in both private and public clouds. For $r = \infty$, it only chooses most cost efficient resources for load balancing. For other values of r, the greedy approach chooses $m - r + 1$ most cost efficient resource types for load balancing, which m shows the total resource types.

4.5.2 Result

Experimental results are analyzed and discussed based on three performance metrics: makespan, total cost, and objective function value (z). In all of the experiments, we used different values for ϵ ranging from 0.1 to 0.4 with an step of 0.1 as well as different values for δ as 5%, 10%, 15% and 20%.

In Figure 4.2, we compare performance of our algorithm with the heuristic algorithm[2]. Due to the similar patterns, only those graphs concerning two extreme values of ϵ and δ are presented. The results confirm that our approach can successfully reduce both time and cost (hence, the objective value) of using hybrid cloud resources. The average reductions in makespan and total cost in each scenarios are given in Table 4.3. The average reductions of makespan and cost usage are approximately 7.5% and 4.8%, respectively. It can be seen that though the greedy algorithm can reach to an acceptable solution when the BoT comprised only short tasks, it fails to reach a good result in the other situations. One reason for this phenomena is that greedy heuristic assigns blindly the long tasks to resources without any per-calculation, while our approach considers a limitation on the maximum workload of each resource, based on the solution for Equation 4.14 or 4.16. Therefore, we can conclude that the performance of greedy algorithm degrades significantly when the granularity of task, in terms of running time, is increased.

Another observation is that performance of our algorithm is not too sensitive to the large values of either ϵ or δ. One may expect that when either ϵ and δ gets larger, the quality of solution must decrease. In practice, however, this does not happen and Algorithm 1 produces a good estimation of μ. Interestingly, one anomaly has been detected when the values of ϵ and/or δ are chosen very small, for example below 5%. While one expect that small values of ϵ and/or δ may result

[2]Each point in Figure 4.2 shows a particular value of r starting from 1 and being incremented by 1.

in higher accuracy of both prediction of μ and the performance of algorithm, in reality, the algorithm's performance decreases significantly.

In fact, by choosing a small value for either ϵ or δ, the number of tasks used for Algorithm 1 increases; this leaves little room to optimize the assignment of rest of tasks. As an example, to reach an estimation with $\epsilon = 1\%$ and $\delta = 1\%$, nearly half of the available jobs should be used for prediction process; which leads clearly to a poor performance. We have observed that the values of ϵ and δ around $10 \approx 30\%$ and 5 15%, respectively are most appropriate.

The overhead of our scheduling algorithm can be negligible comparing to the actual value of BoT application running time. For example, for acceptable values of $\epsilon = 10\%$ and $\delta = 5\%$, the overhead is less than 1% of total BoT application running time.

4.6 Summary

We have addressed the problem of BoT task assignment on a hybrid-cloud environment. When a large-scale BoT application runs with support of the public cloud resources, the assignment of tasks should explicitly take into account the cost efficiency. Introducing a novel objective function, we dealt with the problem by formulating it as an optimization problem. We have used an *FPRAS* algorithm in case of unknown running time of tasks, where our algorithm combines a Monte Carlo sampling method to estimate the unknown values. Our method provides an effective means for the user to run large-scale CPU-intensive BoT applications with or without prior knowledge of task processing time. The quality of the task assignment algorithm has been evaluated by running experiments using the in-house cluster as the private cloud, and *Amazon EC2* instances as the public cloud.

Table 4.3: **Differences between the total cost and makespan of the proposed algorithm versus the greedy approach for different BoT types and sizes. Total number of leased machines is equal to 100**

BoT size (n)	BoT Type	Diff. Cost(%)	Diff. Makespan(%)	Sum of Diff.(%)
3*10k	Short	4.0	6.3	10.3
	Mixture	4.0	6.8	10.8
	Long	4.1	10.3	14.4
3*100k	Short	4.2	5.6	9.8
	Mixture	5.1	8.5	13.6
	Long	5	9.3	14.3

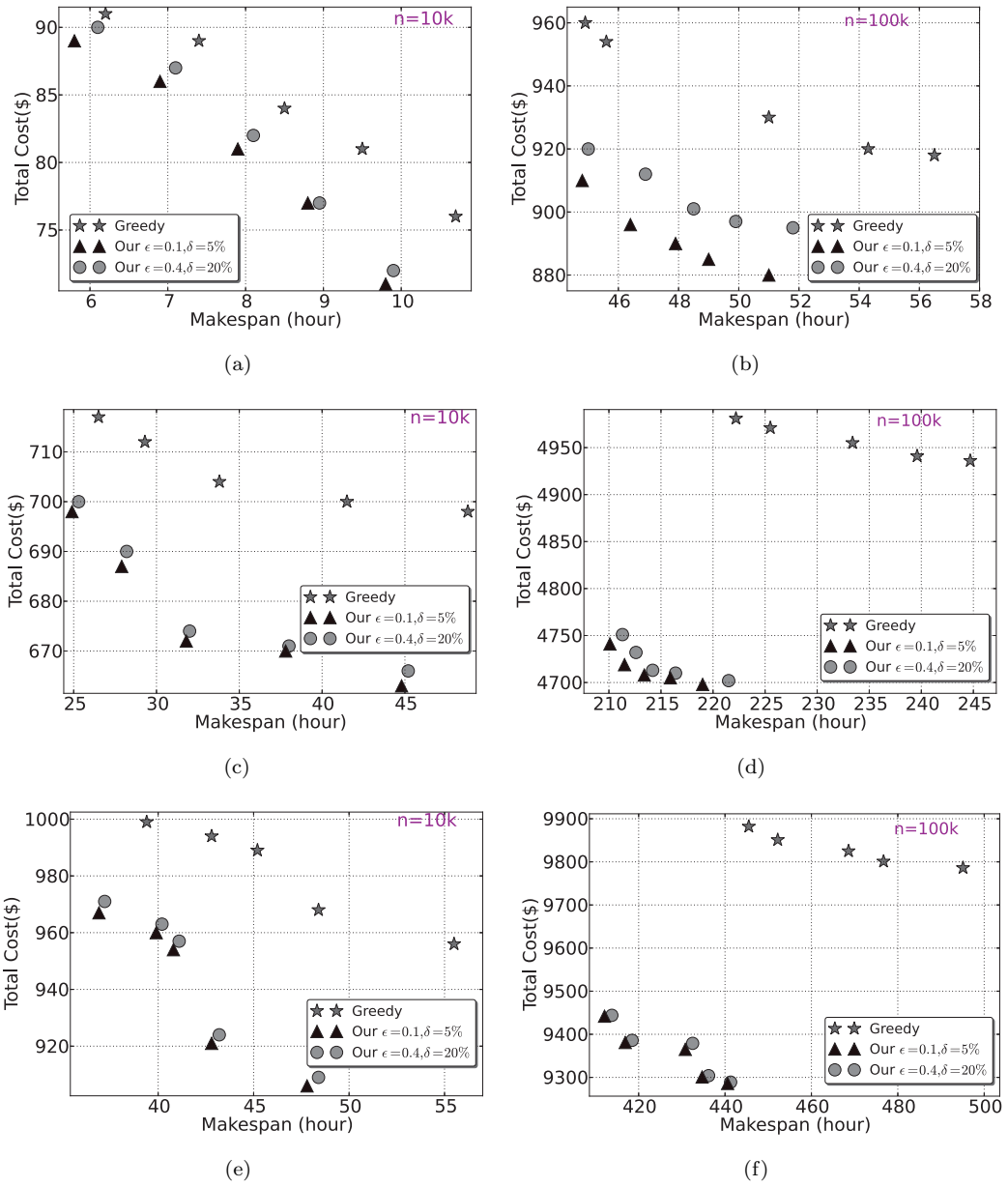


Fig. 4.2: **Performance of our proposed algorithm (with two different values of ϵ and δ) versus that of greedy approach with different sizes of BoT (n = 10k in Figures (a), (c), and (e) and n = 100k in Figures (b), (d), and (f)). Scenarios correspond to tasks with (a) and (b) short running times, (c) and (d) mixture of both short and long running times, and (e) and (f) long running times. The number of machines taken from each instance type is set to 20 (100 total machines).**

Chapter 5

PANDA: a Framework for Pareto-Optimal Cloud Bursting

Fast, Cheap, Reliable, Choose two.
–Engineering tagline

5.1 Introduction

The massive horizontal parallelism of BoT applications often overwhelms the capacity of private clouds. This capacity limitation can be easily overcome with the adoption of virtually 'unlimited' resource capacity of public clouds. However, the degree of performance gain is often not strongly correlated with the usage cost. This cost efficiency issue in scaling out is a major obstacle. Cost efficiency can be translated to the performance to cost ratio from the user's perspective; and is subjective to the user (user's utility).

In this chapter, we address the cloud bursting of large-scale BoT application with the main objective of optimizing the performance to cost ratio. The idea is the user is enabled to optimize the user's utility by choosing the best tradeoff point between application completion time and cost. We have designed the PANDA (PAreto Near-optimal Deterministic Approximation) scheduling framework with a fully polynomial-time approximation scheme (FPTAS) as a novel static scheduling algorithm.

5.2 Pareto Optimality of Cloud Bursting

In the context of cloud bursting, the user often has two conflicting objectives of minimizing cost (c) and maximizing performance (or minimizing makespan (t)). Normally, there is no solution that achieves the best value of both criteria simultaneously. If so, the concept of Pareto optimality is useful. A schedule σ' is said to be *dominated* by another one σ if the values of both metrics of cost and time are worse than or equal to that of the latter, and strictly worse in at least one criterion. A schedule σ^* which is not dominated by any other schedules is Pareto-optimal. The *Pareto frontier* is the set of all feasible Pareto-optimal points. In practice, the user's attention can be restricted to Pareto frontier points. Each user possesses a utility function, $\mathcal{U}(c,t)$, to represent the value of a particular schedule from the user's interest. However, actual user preferences are difficult to obtain; and perhaps a complex utility function could make the scheduling problem intractable.

The use case scenario of PANDA is as follows: the user submits a BoT job specifying application information (e.g., input parameters, data) and/or resource requirements. Task lengths of the BoT application are estimated (or obtained). The Pareto frontier generator produces all scheduling candidates in Pareto-frontier and sends them back to the user. The user selects a scheduling point to realize the maximum utility. Then, PANDA finds a near-optimal schedule based on the selected point and the actual execution is carried out. A schematic diagram of PANDA processes with specific components is given in Figure 5.1. PANDA enables the user to realize the best performance to cost ratio (based on user's particular interest or utility) when running BoT application in a multi-cloud environment. We have much reduced the gap between theory and practice with a proof-of-concept and its thorough evaluation using a real-world application (ISOMAP [226]). PANDA is also capable of effectively dealing with uncertainties in application or resource performance. The remainder of this section is organized as follows. Section 5.3 gives the problem formulation of cloud bursting. Section 5.4 presents our approach to schedule BoT application, and provide analyses on the algorithmic complexity of our algorithm. Section 5.5 presents a proof of concept of the PANDA framework. We also presents a discussion on costs associated with private clouds, and describe how the typical *List* scheduling heuristic is extended for fair comparisons. In Section 5.8, our evaluation methods are described. And lastly, Section 5.9 presents the experimental results.

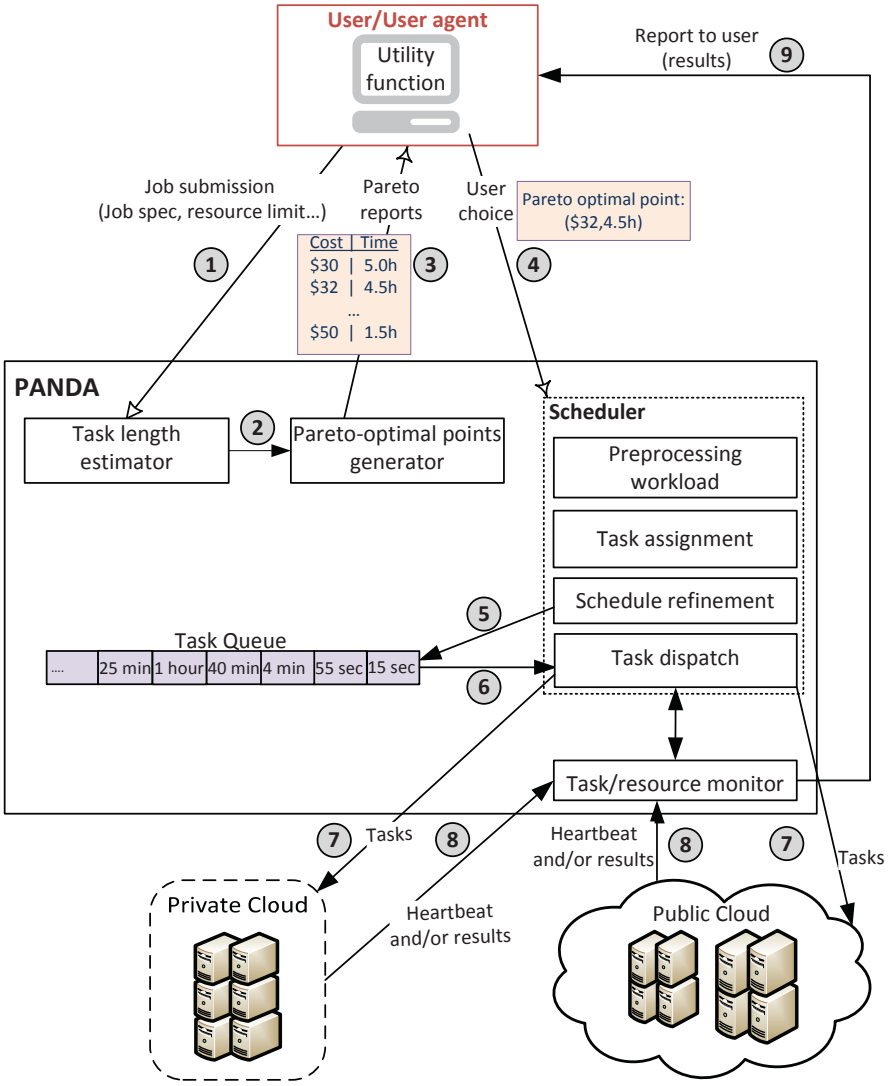


Fig. 5.1: **PANDA framework**

5.3 Problem Formulation

In the following, we formulate the cloud bursting of BoT applications as a binary integer programming problem. The models for public cloud, private cloud, and Bag of Task application follow the model presented in Section 3.4.

5.3.1 Performance versus Cost

The set of solutions for the following objective function is equal to the set of Pareto-efficient frontier (see proof in [193]). *Time* and *Cost* refer to the makespan and the total expenses incurred of job completion, respectively.

$$\min z = \theta \text{ Cost} + (1-\theta)\text{Time}; \quad \forall 0 \leq \theta \leq 1 \tag{5.1}$$

The value of θ helps the user make better realization of the optimization goal. When θ is close to zero, the user is willing to pay less money, which normally leads to a schedule with larger makespan. On the contrary, when θ approaches to one, the user tends to opt for an expensive schedule with a shorter makespan. Two values of θ studied widely before:

- $\theta = 1$: This generates a solution that all tasks run in resources of the most cost effective type, i.e., the resource type with the highest values of $\frac{s_i}{c_i}$.
- $\theta = 0$: This leads to a load balanced solution among available resources. Specifically, each resource of type i receives a load proportional to the value of $s_i / \sum_{\kappa \in \Gamma \cup \{v\}} L_\kappa s_\kappa$.

In spite of these two cases, identifying the true value of θ is generally difficult. Indeed, the user requires all knowledge about the relative importance of all objectives *in priori* to supply the true value of θ. To overcome, we offer a posteriori approach by delivering a (sufficiently) large representative set of Pareto options among which the user chooses the preferred one. When effect of ATU-based pricing is considered, scheduling schema with integer completion times and minimum costs are good candidates to tackle Equation 5.1.

Figure 5.2 shows 150 points of different schedules for running the ISOMAP application on Amazon EC2. There are clearly no Pareto points within some subranges, e.g., within interval (21,21.5) as all of points are dominated by a Pareto point annotated by T_2. This fact helps us identify a set of Pareto candidates to deliver to the user, i.e., schedules with integer completion times and minimum costs. Let E denote the set of Pareto points being delivered to the user. Let T_{best} and T_{worst} show the best and the worst makespan achievable for the execution of a given BoT application. We define $T_0 = T_{best}$, $T_{|E|} = T_{worst}$ and $T_r = \lfloor T_{best} \rfloor + r\Delta T - \varphi$, for $0 < r < |E|$, where ΔT is a positive integer number (the time step) and φ called gap factor is a small positive number to maintain a gap, e.g., φ =5 minutes. T_r's are the end-points of subrange when the interval of $[T_{best}, T_{worst}]$ is divided into $|E|$ subranges.

We are able to restate Equation 5.1 as a binary optimization problem, and its solution produces set E based on Equation 5.2. We set a binary variable $x_{i,j}$ to 1 iff task j is chosen to run on the resource i and 0 otherwise.

$$\begin{aligned} \min \text{Cost} &= \sum_{i\in\Gamma, 1\le j\le n} c_i \lceil \frac{x_{i,j}P_j}{s_i} \rceil + c_v \left(\frac{x_{v,j}P_j}{s_v} \right) \\ s.t. : \text{Time} &= \max_{i\in\Gamma\cup\{v\}} (\sum_{j=1}^{n} \frac{x_{i,j}P_j}{s_i}) < T_r \\ \sum x_{i,j} &= n \quad \text{AND} \quad x_{i,j} \in \{0,1\} \end{aligned} \tag{5.2}$$

where Equation 5.2 must be held $\forall i \in \Gamma \cup \{v\}, j = 1 \cdots n$. A solution to the minimization problem stated by Equation 5.2 gives exactly $|E|$ Pareto optimal points while each of them has a makespan at most up to T_r, for $0 \le r < |E|$.

5.4 Pareto-Optimal Scheduling BOT Applications

In this section, we present an FPTAS solution for the optimization problem stated by Equation 5.2.

An FPTAS algorithm takes both the problem specifications and a value $\epsilon > 0$ as its input, and approximates the optimal solution within a ratio bound of $(1+\epsilon)$. For any choice of ϵ, it has a running time that is polynomial in terms of both n and $1/\epsilon$. Our FPTAS solution algorithm (Algorithm 6) consists of three steps: pre-processing, task assignment, and solution refinement.

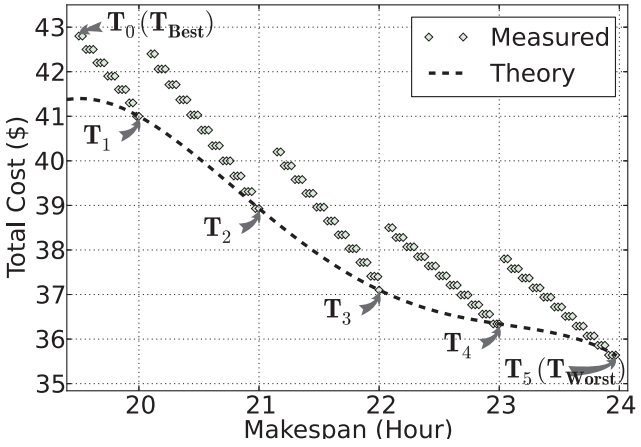


Fig. 5.2: **Pareto frontier for a BoT app (ISOMAP).**

5.4.1 Preprocessing

In the first step of our algorithm, tasks are preprocessed for their lengths to be equalized. This preprocessing enables us to find the optimal workload for each resource. To this end, we take a small real number, called μ, and replace each task of length P_j as a set of new tasks each of them with its length equal to μ (choosing $\mu = 1$ is normally appropriate for real application). The number of these equal length tasks is $n^* = \sum_{i=1}^{n} \lfloor \frac{P_i}{\mu} \rfloor$ instead of n in the original problem. Now, we deal with a new profile of the problem where there are n^* tasks of identical length $P = \mu$. It can be easily shown that in an optimal solution for this new problem, each resource of a particular type receives the same amount of workload. Let x_i show the number of tasks assigned to resource of type i. The scheduling problem for the new version can be restated as an integer (but not necessarily binary) optimization problem as follows:

$$\begin{aligned} \min \text{Cost} &= \sum_{i \in \Gamma} L_i c_i \lceil \frac{x_i P}{s_i} \rceil + L_v c_v (\frac{x_v P}{s_v}) \\ s.t. : \text{Time} &= \max_{i \in \Gamma \cup \{v\}} (\frac{x_i P}{s_i}) < T_r \\ &\sum_{i \in \Gamma} L_i x_i + L_v x_v = n^* \end{aligned} \tag{5.3}$$

where both x_i and x_v belong to $Z^{\geq 0}$. As we target BoT applications with a large number of tasks, we can use a relaxation technique (by removing the integral constraints) to transform the above problem into a nonlinear version solvable in polynomial time.

We sort resources in descending order by their cost efficiency (s_i/c_i), such that for any two resources i and j, if $i < j$ then $s_i/c_i \geq s_j/c_j$. Then, the solution for the relaxed problem can be stated as:

$$x_i = \frac{T_r s_i}{P} n^* - \sum_{k=1}^{i-1} L_k x_k \geq \frac{L_i T_r s_i}{P} \frac{n^* - \sum_{k=1}^{i-1} L_k x_k}{L_i} \tag{5.4}$$

Note that changing the profile of original problem, by introducing μ, does not impose any extra cost in terms of the running time of algorithm.

Algorithm 4: Trimming procedure

input : ℓ, δ: a sorted list of real numbers and a trimming factor
output: ℓ': a trimmed sorted list of real numbers
begin
  Let $\ell = \langle z_1 \cdots z_{|\ell|} \rangle$
  $\ell' = \langle z_1 \rangle$
  $last = z_1$
  for $i = 2 \cdots |\ell|$ **do**
    if $z_i > \frac{last}{1-\delta}$ **then**
      $\ell' = \ell' || z_i$
      $last = z_i$
    end
  end
  return ℓ'
end

5.4.2 Task Assignment: Subset Sum Algorithm

Let $\mathbf{x}^*$=$\langle x_1^*, x_2^* \cdots x_k^*, x_v^* \rangle$ represent the optimal solution based on Equation 5.4. Now that we find a subset of tasks that exactly add up to the value of x_i^* or get as close as possible. This can be considered as a general case of the subset-sum problem which is already shown to be NP-complete (see e.g., [227]). The problem can be stated formally as follows. Given n numbers as $\pi = \{P_1, P_2 \cdots P_n\}$, and a set of real numbers $x_1^*, x_2^* \cdots x_k^*, x_v^*$. We seek a subset of π, namely $\pi_1, \pi_2, \cdots, \pi_k$, whose sum is as close as possible to each value of x_i^*. These subsets must form a partition in the sense that they are disjoint and cover π. Our proposed approach to do this is summarized formally in the outer **for** loop of Algorithm 6. Here, for each target value of x_i^*, we compute sums of all subsets $\{P_1, P_2 ... P_j\}$ (the inner **for** loop). However, to reduce the size of such big sets, the sum of the first j numbers in π' is trimmed by a proper parameter.

The trimming process takes place in the inner **for** loop that constructs a sorted list ℓ_j based on $\{P_1, P_2 \cdots P_n\}$. Specifically, auxiliary procedure $Merge(\ell, \ell')$ returns the sorted list that is the merger of its two sorted input lists ℓ and ℓ' in which duplicate values are removed. The merged list (ℓ_j) is then trimmed using Algorithm 4 for all elements in the output list to satisfy the trimming property.

Given a parameter δ, where $0 < \delta < 1$, real number z approximates real number y by factor δ, if $\frac{y}{1+\delta} \leq z \leq \frac{y}{1-\delta}$. Let ℓ indicate a list of some real numbers, *i.e.*, $\ell = \langle z_1 \cdots z_{|\ell|} \rangle$. List ℓ is trimmed by parameter δ, if we remove as many elements as possible from ℓ such that every element removed is approximated by some remaining element in the list by factor δ.

The trimming procedure can dramatically reduce the number of items in a list while keeping a close representative value in the list for each deleted item [219]. Algorithm 4 trims the input list ℓ in time $\Theta(|\ell|)$, assuming that ℓ is sorted in increasing order, and outputs a trimmed and sorted list ℓ'. In Algorithm 6, trimmed elements larger than $\frac{x_i^*}{1-\frac{\epsilon}{2|\pi'|}}$ are removed as we only look for a subset whose sum is x_i^*, (or close to it by the factor of $\frac{\epsilon}{2|\pi'|}$). Once the nearest value to the target value, x_i^*, is found, numbers that create x_i^* are removed from set π', and the procedure is repeated for the next target value.

5.4.3 Refinement

Lastly, we attempt to further reduce z value using task rearrangement (Algorithm 5). The optimal solution found by Equation 5.4 is not necessarily integer. Rather, it provides a lower bound on the optimal value. The feasible set of original integer problem is given by the set of all integer-valued points for x_i. One may simply round the fractional solution given by Equation 5.4 to obtain an integral feasible solution. However, the rounded solution is most likely not optimal for the original problem due to the simplification we made by replacing the value of $\lceil \frac{x_i P}{s_i} \rceil$ with $\frac{x_i P}{s_i}$.

A simple and yet effective solution for this discrepancy is *task rearrangement*. Besides making schedule feasible, we attempt to rearrange tasks in the way cost and/or makespan is reduced. For example, a task currently assigned to a slow resource is moved to a faster resource such that the time required by the faster resource to run all assigned tasks including the task being moved does not incur any extra cost. Yet, the completion time with the moved task still remains within the original multiple of ATUs for tasks without the moved task.

In each iteration of the *while* loop of Algorithm 5, two resources i and j are selected and a certain number of tasks in one resource (i) are moved to the other resource (j) if this rearrangement reduces $\zeta = Cost(x)$. Specifically, the amount of time that resource i is leased (in whole hours) but not utilized (partial hour) is identified by r_i; and this is used to calculate values of l and u for each resource. These values indicate the numbers of tasks that can be removed from and added to resource i, respectively, without change to resource rental time (in full hours). Then, the algorithm finds a combination of tasks (δx_i) currently running on the other resource, j, for which the sum of these tasks is equal to either l_j or u_j. For every possible combination of tasks, the value of $\zeta(x)$ is checked to see if it

Algorithm 5: Rearrange task assignment

input : $n, P, L_i, c_i, s_i, T_r, \mathbf{x}$ (workloads vector);
output: $\langle \mathbf{x}^* \rangle$
begin
  $S = \Phi$;
  while *true* **do**
    Let $r_i = frac(\frac{x_i P}{s_i})$; // get fractional part;
    $l_i = L_i \lceil \frac{r_i s_i}{P} \rceil$;
    $u_i = L_i \lfloor \frac{(1-r_i)s_i}{P} \rfloor$;
    $\mathbf{\Delta} = \{(\delta x_1, \cdots, \delta x_k) | \exists j : \sum L_i \delta x_i = l_j \text{ or } \sum L_i \delta x_i = u_j;\ \delta x_j = 0;\ \delta x_i \geq 0;$
    $\{i, j\} \notin S;\ i, j \in \Gamma\}$
    Find $\min_{\Delta} \zeta(\mathbf{x} \pm \mathbf{\Delta x})$ where

$$\mathbf{x} \pm \mathbf{\Delta x} = \begin{cases} (\mathbf{x} + \mathbf{\Delta x}) & \text{if } \sum L_i \delta x_i = l_j \\ (\mathbf{x} - \mathbf{\Delta x}) & \text{if } \sum L_i \delta x_i = u_j \end{cases}$$

    and
    $\mathbf{\Delta x} = (\delta x_1, \delta x_2 \cdots - \frac{1}{L_j} \sum_{i \neq j} L_i \delta x_i \cdots \delta x_k)$,
    if $\zeta(\mathbf{x} \pm \mathbf{\Delta x}) < \zeta(\mathbf{x})$ **then**
      $S = S \cup \{j\}$;
      Update $\mathbf{x}$ with $\mathbf{\Delta x}$
    else
      break
    end
  end
end

is reduced. $\zeta(\mathbf{x})$ denotes the total cost value of a specific configuration $\langle \mathbf{x} \rangle$, as described by Equation 5.3. Once the best possible reduction of cost is found, we remove/add δx_i tasks from/to each resource indexed by i and add/remove $\sum \delta x_i$ tasks—equal to either l_j or u_j—to/from resource j. We add resource j to a holding set (S) to exclude it in the following rounds.

5.5 Proof of Concept Implementation

In this section, we demonstrate the feasibility of our framework.

5.5.1 Runtime Prediction Models

Due to the fact that HPC users need to provide run-time estimates for submitted jobs and this estimation significantly impacts the performance of parallel and distributed systems, this topic have been the focus of several researches. As mentioned

Algorithm 6: Approximate task assignment

input : $n, \epsilon, \pi=\{P_1, P_2 \cdots P_n\}$, L_i, c_i, s_i, T_r; $\forall i \in \Gamma \cup \{v\}$
output: π_i; a partition scheme of π while the sum of numbers in π_i approximates x_i^*;
begin
  Set μ to a small real number, e.g., 1;
  Let $n^* = \sum_{i=1}^{n} \frac{P_i}{\mu}$;
  Find optimal task assignment, x^* // Equation 5.4
  $Sort(x^*, descending)$;
  $\pi' \leftarrow \pi$;
  for $i = 1 \cdots k+1$ **do**
    $\ell_0 \leftarrow \langle 0 \rangle$
    for $j = 1 \cdots |\pi'|$ **do**
      Let $\ell_j = Merge(\ell_{j-1}, \ell_{j-1} + P_j)$ $\ell_j \leftarrow Trim(\ell_j, \frac{\epsilon}{2|\pi'|})$ // Algorithm 4
      Remove elements from ℓ_j for which the size is greater than $\frac{x_i^*}{1-\frac{\epsilon}{2|\pi'|}}$
    end
    Let π_i^* be the nearest value to x_i^* in ℓ_j;
    $\pi_i \leftarrow$ set of numbers whose sum equals to π_i^*;
    $\pi' \leftarrow \pi'$ - π_i;
  end
  Rearrange // Algorithm 5
end

in [228], if a good user or system estimator does not include, any improvement in scheduling algorithm could not work nicely (in other words, seems to be wasting of effort). In a comprehensive study leaded by a group of researchers in Hebrew University [228], authors analyzed workload logs collected from parallel machines in production use, and showed that user runtime estimates are highly inaccurate even for a single machine. The situation becomes worse when several geographically distributed computers with unknown characteristics are used as the user have no real information on application behavior on such systems. A metric called "user run time estimate accuracy" of each job can be defined as the ratio of the real runtime to the estimate value, $\left(p_j/p'_j\right)$. In the case that there exist some automatic tools that can generate a runtime prediction of the submitted job, we can similarly define "system prediction accuracy" as the ratio of the real runtime to the system generated prediction $\left(p_j/p''_j\right)$.

There exist several models that a user can apply for job running time estimation. Feitelson [84] suggested that user can simply assume a job's running time is distributed uniformly within the range of $[R, R(1+f)]$, where R is an initial guess about running time and $f > 0$ is a factor of uncertainly. The main problem of this model is that estimation is correlated with the real value of running time,

which gives the scheduler too amount of valuable information. The assumption of existing this information is not true in practical situation, however. An improved version of this model can be constructed by attempting to model the observation of histograms obtained from real environment. Analyzing such histograms implies that $p_j/p'_j = u$ where u is a random variable distributed uniformly within $[0, 1]$. So, one can find the artificial estimated value by changing the sides of mentioned equality (also called ϕ-model [229]).

The problem of ϕ-model is that there exists a hidden connection between longer runnig times and increased accuracy indeed. It means that the distribution of jobs in the accuracy histogram is not uniform. Hence, another model was proposed by Cirne and Berman [230], which took the opposite direction to the ϕ-model. In their model, they use gamma distribution for modeling accuracies, while estimates were modeled using a log-uniform distribution. However the model suffers from the same problem as ϕ-model, because accuracy and actual runtime are still independent. The work in [228] tries to improve previous models by estimating modality. By considering the estimates distribution as a sequence of modes, they could deliberately investigate main characteristics of user estimations. Their model is available to download from the parallel workload archive [231]. The outcome of using above-mentioned models are unrealistically better than those obtained with real estimates, however. In [232, 233], authors argued that the correct approach for workload modeling is user- and session-based modeling, instead of analyzing jobs directly. They also provide some techniques to identify the useful user and session features by using a method based on k-means clustering. In [234], Tsafrir *et al.* showed that how historical data can improve estimates because users tend to execute same program several times. Their simple runtime predictor calculates the average of runtimes of last two jobs submitted by the same user. Quiet interestingly, they showed that using most recent data can be more effective than mining the whole history. Work in [124] used the history of system generated prediction models and incorporated them into different job allocation policies in order to improve the overall grid performance. It seems that all currently available models for generating user estimates are lacking in some respect. While the current scheduling policies are mainly based on the the runtime estimates and assume that users are able to provide accurate estimates of job runtimes, which is not the true case often, the grid or cloud resource assignment may suffer severely from inadequate accuracy of estimation tools.

5.5.2 Task Length Estimator

The information on task length in our experiments is obtained by running ISOMAP with various settings in our private system. Although there are several techniques—such as Monte Carlo sampling methods (e.g., [217])—to estimate the task length, our preference was to use the actual task length values to avoid any unknown or severe side effects that an estimation might have on evaluating of our system.

5.5.3 Scheduler

The actual scheduler has been developed using Python. The key functionalities are: (1) it launches either binary executables or shell scripts automatically in a remote machine. Tasks can be executed independently from one another. Depending on the number of cores, tasks are executed in parallel in each machine (using package `multiprocessing`); (2) it triggers events for starting tasks, such as API for monitoring and acquisition of Amazon EC2 resources (using package `boto` [235]). Such an event initiates the execution of the next available task awaiting in a shared task queue; (3) it allows the user to configure job parameters. Task parameter control is carried out by command line or by a built-in graphical user interface. An XML-based configuration scheme for job management is also developed; and, (4) some built-in log facility collects performance metrics, generates some file reports, and plots charts.

There is an option to run different scheduling strategies, such as $List_n$ algorithm for performance comparisons. The SSH protocol has been used for communication and remote execution.

5.6 Running Time of the Rearranging Algorithm

In this section, we prove that Algorithm 5 runs in a reasonable amount of time with the following lemma.

Lemma 5.1. *In Algorithm 5, if a resource has been selected for task rearrangement, it does not need to be considered in the future rounds. Hence, we add it to the exclusion set S.*

Proof. Let R_n be some round in the refinement process of Algorithm 5. In round R_n, some tasks are added to resource j from resource i. In a later round R_m

($R_m > R_n$), tasks in resource j are moved to resource j'. Because, in both situations, values of objective function must be decreased by the maximum amount, moving tasks from resource i to resource j' should decrease the value of objective function more than moving tasks from resource i to resource j in round R_n, which contradicts the assumption of maximum reduction in objective function value. □

With the above lemma, it is concluded that once a resource is selected as a candidate for task rearrangement (addition or removal), it must be excluded from the refinement process in other rounds. This fact is checked in Algorithm 5 with set S. This exclusion ensures the number of rounds in the **while** loop is not greater than the number of different resources rented from the public cloud, i.e., $|\Gamma|$. In each iteration of the while loop, all possible combinations of number δx_i—sum of them is equal to either l_j or u_j—must be checked to find the minimum value of objective function, shown by $z(\mathbf{x} \pm \mathbf{\Delta x})$ in the **if** statement. Let $|\Gamma| = k$, then the running time to check all of different combination is less than

$$\binom{l_j + k - 2}{k - 2} + \binom{u_j + k - 2}{k - 2} < 2\frac{(\frac{s_j}{P} + k - 2)^{k-2}}{(k-2)!}. \tag{5.5}$$

, so the running time of Algorithm 5 is bounded by $O((\frac{s_j}{P})^{k-2})$. For a typical configuration of current public cloud environment, values of these parameters ($\frac{s_j}{P}$ and k) can be estimated to be less than 10 and 5, respectively, resulting in the running time of the algorithm being almost thousands statements which is quite acceptable. The algorithm finishes when no resource can be found to reduce the value of objective function, i.e., the global minimum point is found.

5.7 Complexity of the Scheduling Algorithm

In this section, we analyze the complexity of Algorithm 6 with respect to problem size (n) and approximation factor (ϵ).

Theorem 5.2. *Algorithm 6 is a fully polynomial-time approximation scheme for the task assignment problem in case of varying length tasks.*

Proof. To show that Algorithm 6 is an FPTAS approximation scheme for the task assignment problem of varying-length tasks we need to show that return value of the algorithm, z^*, satisfies the following condition: $z^* \leq z_{OPT}(1 + \epsilon)$, where z_{OPT} denotes the optimal solution to Equation 5.1. We must also show that the running time of this algorithm is polynomial in terms both $1/\epsilon$ and the size of the input.

Based on a similar approach proposed in [236], one can show that the outer **for** loop yields a value of π_i^* which satisfies $\pi_i^* \leq x_i^*(1+\epsilon)$. Let $\zeta = \sum f(\boldsymbol{y_i})$ be the associated value of the objective function to the vector input $\boldsymbol{y}$ according to Equation 5.3, which $f(\boldsymbol{y_i}) = \omega_i \lceil \gamma_i y_i \rceil$ for some known coefficient of ω_i and γ_i. So, the following inequality must hold for any input of increasing function f:

$$\begin{aligned} \pi_i^* \leq x_i^*(1+\epsilon) \Rightarrow f(\pi_i^*) \leq f(x_i^*(1+\epsilon)) \\ \leq \omega_i \lceil \gamma_i x_i^*(1+\epsilon) \rceil \leq (1+\epsilon)\omega_i \lceil \gamma_i x_i^* \rceil \leq (1+\epsilon) f(x_i^*) \Rightarrow \\ \zeta(\pi^*) < (1+\epsilon)\zeta_{OPT} \end{aligned} \tag{5.6}$$

we use a version of Hermite's identity in the above proof which implies that if real number x is large enough, the following approximately equality is true:

$$\lceil x(1+\epsilon) \rceil \simeq (1+\epsilon)\lceil x \rceil, \forall \epsilon > 0$$

On the other hand, the following inequality holds true:

$$\frac{x_i^* P}{s_i} < T_r \Rightarrow \frac{\pi_i^* P}{s_i} < \frac{x_i^*(1+\epsilon)P}{s_i} < T_r(1+\epsilon) \tag{5.7}$$

Let z^* and $z(\pi^*)$ show the value of Equation 5.1 for the optimum solution (x^*) and π^*, respectively. From (5.6) and (5.7) we can conclude that:

$$z(\pi^*) = \theta \sum \zeta(\pi^*) + (1-\theta) \max T(\pi^*) \leq$$

$$(1+\epsilon)\left(\theta \sum \zeta(x^*) + (1-\theta) \max T(x^*)\right) \leq (1+\epsilon) z^*$$

To show that Algorithm 6 is an FPTAS, we derive a bound on the running time of the algorithm with the size of input. It has already been shown that the inner **for** loop runtime is $\theta(\frac{n}{\epsilon})$ [236]. Thus, the running time of Algorithm 6 is bounded by $O(kn/\epsilon)$, which completes our proof. □

5.8 Experiments

In this section, we give details of our experimental setup and evaluation metrics.

5.8.1 Experimental Setup

The multi-cloud environment in our experiments was modeled as described in Section 5.3 with the use of our 4x10-core cluster and Amazon EC2. Experimental settings are summarized in Table 5.1. The communication overhead can be considered zero as the size of task is small and tasks are CPU-intensive.

In Table 5.1, the processing capacity is defined as how much work a resource performs in a given amount of time, e.g., an hour, comparing to a reference resource (m1.small). To determine this parameter, we constructed a simple and fast-running benchmark to measure how fast a resource solves the ISOMAP algorithm with specific data sets. This benchmark comprised of four data sets with sizes of 1,000 and 5,000 points taken from either Swiss-roll or S-curve manifold. A set of fixed cardinality values of reduced dimension $k \in \{5, 20, 100\}$ and reduced dimension $d = 2$ were chosen for this benchmark. We used the resource limit of 20 regardless of resource type.

BoT jobs used in our experiments are based on ISOMAP. We populated 5,000 different combinations of tasks using ISOMAP with two data sets, namely, Swiss-roll and Swiss-roll manifold. The task lengths lay between range of tens of seconds to several hours. Both time and cost overheads of data transfer are negligible.

5.8.2 Costs in Private Clouds

Total Cost of Ownership[1] of private cloud is a financial estimate for the costs of private cloud usage over its lifetime. For effective usage, TCO must account for the real costs of all items, such as cost of resource purchasing, the cost of maintenance, cost of physical space within a data-center or site, the cost of consumed power and cooling, and the cost of hardware and software used for management and training

[1] TCO

Table 5.1: **Multi-cloud settings**

Cloud	Res. Type	Proc. Capacity	Cost
	m1.small	1	$0.080
Amazon EC2	c1.medium	5	$0.165
US East (VA)	m1.large	4	$0.320
	c1.xlarge	20	$0.660
Private	4x10 Xeon Processors	10	$0.320

purpose, and so on. Each of these items associates with an overhead cost should not be overlooked.

To estimate the meaningful cost of ownership of the private cloud, researchers have proposed several methods for calculating the amortized cost [237–240]. Normally, those models take into account the amortized cost as the sum of infrastructure costs, total equipment energy consumption, cooling system, real estate, administration and maintenance cost, and so on. In each model, user must specify several parameters' values, such as energy price per KWh, scalability factor, operational life span, expected server utilization, rate of interest, etc. As calculating the exact amount of amortized cost is not in our scope, we provide an estimation of this amount for a typical private cloud. Different estimated prices per core have been reported ranging from \$0.08/h [239] to \$0.11/h [240] (we used the exchange rate of \$1.3 for one EURO). We adopt the value of $\$0.32/h$ for the cost of a node in this work (equals to the hourly rate of m1.large instance).

5.8.3 Evaluation Metrics

In this section, we describe three widely used performance metrics adopted for evaluating the quality of scheduling in terms particularly of cost efficiency.

Speedup is defined as $speedup(p) = time(1)/time(p)$ (p is the number of machine used). Although one can introduce $costup(p) = cost(p)/cost(1)$—analogous to speedup, we prefer to determine the cost efficiency of scheduling by combining the cost and speedup (performance), and define the speedup per cost as:

$$perfcost(p) = speedup(p)/cost(p) \tag{5.8}$$

Then, the cost efficiency of a schedule with multiple resources is determined by $perfcost$, *i.e.,* the higher the better compared with that of single resource ($perfcost(p) > perfcost(1)$).

A schedule is considered more cost efficient if its $speedup(p) > costup(p)$.

We define another metric to compare the desirability of different scheduling strategies, $Farness_A$, that measures how far the solution of algorithm A is from a corresponding Pareto point. More formally,

$$F_A = \frac{\text{Time}_A - \text{Time}_{Opt}}{\text{Time}_{Opt}} + \frac{\text{Cost}_A - \text{Cost}_{Opt}}{\text{Cost}_{Opt}} \tag{5.9}$$

The closer the value of F_A to 0, the more effective algorithm A. Scheduling overhead (Section 5.9.3) is also used for evaluating our scheduling algorithm.

5.8.4 Extended List Scheduling Heuristic

The idea behind *List* scheduling, one of the widely used policies in distributed systems, is creating an ordered (with some types of priorities) list of available tasks, and continuously running the following actions: (1) select the highest prioritized task, and (2) select a free machine to run this task. The priorities procedure can be decided in various ways such as longest running time or random. In fact, the standard version of *List* can not be fairly compared with our algorithm, as it only takes the speed of resources into account (and not their costs). To overcome, we extend it by a simple rule. Assume that there are m machines available and are sorted into their cost-effectiveness decreasingly. We define $List_\eta$ to represent a *List* schedule which uses only the first η most efficient ($\frac{speed}{cost}$) machines ($\eta \leq m$).

5.9 Results

Our evaluation study was conducted primarily with comparisons between our scheduling algorithm and an extended version of List scheduling heuristic. The extended List heuristic, called $List_\eta$, considers not only the processing capacity of resource, but also the cost of resource.

5.9.1 Impact of Resource Limit

We have conducted experiments with a BoT application of its size (completion time) being 10Ms (million seconds) to show the effect of resource rental limit on the scheduling performance. Whenever ϵ value is not mentioned explicitly, we set it to 0.1 to conduct the experiment in a reasonable time and quality.

Figure 5.3 compares the achieved performance of PANDA with both extended List algorithm, $List_\eta$, and the optimal values. As can be clearly seen while $List_\eta$ fails to produce an acceptable solution, PANDA can reach closely to the optimal value.

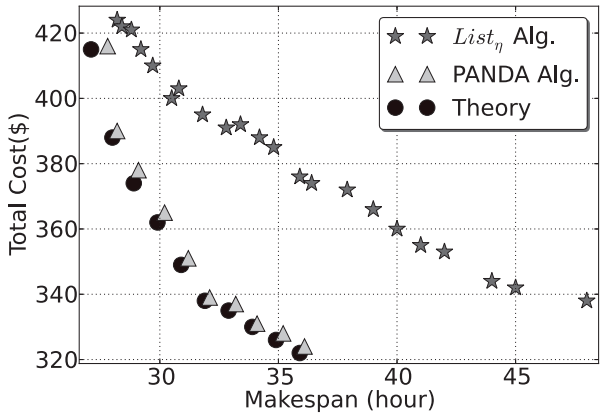


Fig. 5.3: **Pareto frontier reached by three different approaches.** $L_i = 5$**,** $\epsilon = 0.1$**, and** $|B| = 10Ms$ **(on m1.small).**

5.9.2 Impact of Application Characteristics

An application may consist of tasks with different lengths. The mean and the variance of task lengths can heavily affect the performance of any scheduling strategy. This section presents a set of experimental results designed to show the impact of BoT application size, $|B|$, and task lengths.

In Table 5.2, the achieved total cost and makespan of different algorithms for various BoT application sizes are summarized. The solution quality of $List_\eta$ heuristic is comparable to ours for small BoT applications; however, the discrepancy increases when the BoT size grows (total 30% difference with our algorithm when $|B| = 17M.s.$)

Another important factor affecting the performance is task granularity (length characteristics). We distinguish three different BoT application based on their task length characteristics: short tasks (an application consists solely of tasks with short computational requirements), long tasks, and mixture of both short and long tasks.

We define BoT mixture ratio, r, as the percentage of long tasks in a given application, *i.e.*, $r = \frac{|long\ tasks|}{|B|}$. This ratio can be interpreted as the BoT granularity. For a given application, r=1 indicates it consists of long tasks whereas r=0 shows it is composed of short tasks. The higher value of r an application has, the more performance (speedup or Speedup Efficiency defined as $\frac{speedup(p)}{p}$) our algorithm can reach. Table 5.3 depicts the achieved Speedup Efficiency of PANDA and $List_\eta$ heuristic. Speedup efficiency which is a value between zero and one, represents how well utilized the resources in a given solution. The general inference is that when the variance of task lengths grows, the $List_\eta$ algorithm exhibits poor performance.

Figure 5.4 depicts the average speedup and $perfcost$ of two algorithms with respect to different numbers of leased resources in three different BoT application types. We recall that the optimal value is not achievable by any algorithm, as it has been calculated if the application is considered either as a fully parallel application or possessing the ability of preemption.

Figure 5.4(a) depicts the speedup and the $perfcost$ for BoT application with small tasks. The size of BoT application is equal to $|B| = 10^6$ sec. (on $m1.small$), and each task length is in a range of [10, 500] seconds. There is the maximum $perfcost$ value at 10 machines which shows hiring more than 10 resources is not cost efficient. The $List_\eta$ heuristic fails to reach a good solution when the mixture ratio grows (As shown in Figures 5.4(b) and 5.4(c) with $r = 0.7$ and $r = 1$, respectively). Here, size of BoT application is equal to $|B| = 10^7$ sec. (on $m1.small$), and task lengths are in a range of [500 sec., 5 hours].

Table 5.2: **Average values of makespan and total cost with respect to different sizes of BoT applications. The reference is m1.small,** $L_i = 20$ **and** $\epsilon = 0.1$**.**

$\|B\|$	$List_\eta$		PANDA		Optimal	
	ms(h)	cost	ms(h)	cost	ms(h)	cost
1M.s.	2.2	66.0	1.8	58.5	1.5	58.5
5M.s.	4.2	118.4	3.6	117.2	3.3	117.2
10M.s.	5.5	153.5	4.9	146.0	4.5	146.0
17M.s.	9.7	241.7	8.2	215.6	7.9	192.8

Table 5.3: *Speedup Efficiency* **of PANDA and** $List_\eta$ **with respect to different values of BoT mixture ratio parameter (**r**) from 0 to 1 with a step of 0.2 and different numbers of machines (**$m = \sum L_i$**).** $|B| = 17M.s.$**, and** $\epsilon = 0.1$**.**

r	m	PANDA	$List_\eta$	r	m	PANDA	$List_\eta$
3*1.0	10	.74	.60	3*0.8	10	.67	.55
	40	.72	.50		40	.60	.43
	80	.70	.32		80	.58	.25
3*0.6	10	.59	.53	3*0.4	10	.55	.51
	40	.55	.38		40	.51	.35
	80	.50	.19		80	.44	.17
3*0.2	10	.52	.47	3*0.0	10	.50	.45
	40	.50	.30		40	.25	.24
	80	.32	.15		80	.13	.12

While PANDA can achieve the appropriate workload for each value of T_r (by using the subset sum procedure), the $List_\eta$ heuristic takes as many tasks as possible while the pre-determined workload of each of η resources is not exceeded. This selection may either leave a large empty workload on some resources, or result in overwhelming other resources with extra workload. It can be seen that $perfcost$ value of PANDA is maximized when 70 resources are leased, and the value is more than two times larger than that of the $List_\eta$ heuristic. Note that $perfcost$ is a descending function after 70 resources.

Table 5.4 shows comparisons between PANDA and $List_\eta$ heuristic in terms of $Farness$ (F_A), for BoT applications of different characteristics (task lengths). The results show that the performance of $List_\eta$ heuristic degrades significantly as the BoT mixture ratio increases.

5.9.3 Scheduling Overhead

In Section 5.7, we showed that the running time of PANDA is polynomial in terms of size of BoT application, n, as well as $1/\epsilon$. Clearly, approximation ratio (ϵ) trades accuracy for running time; and thus, ϵ can be used as a design parameter for scheduling algorithm.

Table 5.5 shows the actual running times of our algorithm on a desktop machine with a 3.0GHz Intel Core2 Quad Processor with 4GB of RAM for different settings and different values of ϵ. It also shows both the achieved farness and scheduling overhead (H) of PANDA. The scheduling overhead of algorithm A is defined as $H(A) = (runtime(A) + H_{net})/runtime(\text{BoT})$. Here, $runtime(A)$ is the running time of algorithm A, H_{net} is the additional overhead, such as boot-up time or the time required to send the data/application to the remote machine, and $runtime(\text{BoT})$ shows the actual processing time needed to execute the BoT application. One can conclude that the upper bound inequality of $z_{PANDA} < z_{opt}(1+\epsilon)$ holds true for all different settings, and consequently it reaches to the pragmatic optimal solution very effectively with reasonable computational overhead (less than 5% on average).

5.9.4 Robustness and Applicability

Optimization solutions are often affected by uncertainty in input parameters; hence, they can show notable sensitivity to even small perturbations in the problem parameters. Two main sources of perturbations are estimation errors on task

Table 5.4: ***Farness* of two approaches. $|B|$=17 M.s., $L_i = 20$ and $\epsilon = 0.1$. BoT applications are differentiated by their task lengths.**

BoT type	$List_\eta$	PANDA
Short	11%	10%
Mixture	33%	12%
Long	41%	13%

Table 5.5: **Running time (RT), *Farness*, and scheduling overhead (H) of PANDA for different ϵ and task lengths. $r = 0.7$ in the mixture case. Sizes of BoT applications, $|B|$, in three different types of short, mixture, and long tasks are 8.2M.s., 11M.s., and 11M.s., respectively.**

	Short			Mixture			Long		
ϵ	RT(h)	F_A	H	RT(s)	F_A	H	RT(s)	F_A	H
0.1	1.8	7	47	30	8	≤ 1	15	8	≤ 1
0.2	1.7	9	42	10	12	≤ 1	7	12	≤ 1
0.3	1.6	10	40	7	14	≤ 1	4	14	≤ 1
0.4	1.0	12	30	7	14	≤ 1	4	16	≤ 1
0.5	0.9	14	25	6	17	≤ 1	4	18	≤ 1
0.6	0.6	17	20	5	20	≤ 1	4	22	≤ 1
0.7	0.5	20	15	5	23	≤ 1	3	24	≤ 1
0.8	0.4	25	12	4	25	≤ 1	3	28	≤ 1
0.9	0.3	30	10	4	29	≤ 1	3	30	≤ 1

length (*EstErr*) and resource performance (speed) variations (*PrfDeg*). PANDA is capable of dealing with these issues using gap factor, φ, and approximation factor, ϵ.

As discussed in Section 5.3.1, Pareto-optimal points have integer completion times, e.g., 3 hours. However, to overcome two aforementioned issues of *EstErr* and *PrfDeg*, we use a gap factor, φ, which is a small positive number, such as 5 minutes, to keep the resources empty during that interval. Thus, if the resource performance degraded unpredictably, the gap factor would allow us to meet the assigned Pareto-optimal point. For example, if φ=5 min, then we would assign the resource workload to be run toward 2 hours and 55 minutes instead of full 3 hours. Selecting a proper φ value depends on intensity of both *EstErr* and *PrfDeg* values. Apparently, if the scheduling system is working on resources with frequent performance degradations, the φ value must be considered higher. The same argument is true in the case of low accuracy of estimation tool.

Approximation factor, ϵ, provides additional controlling parameter to encounter with the mentioned issues. The trimming procedure, Algorithm 4, uses a trimming factor, δ, which is related to ϵ, to keep only one of the adjacent numbers. For example, assume that all numbers within the interval $\mathcal{I} = (x/(1-\delta), x/(1+\delta))$ are trimmed by x value. So, for any number $z \in \mathcal{I}$, if the estimation method calculated another number of $z' \in \mathcal{I}$ instead of z, the output performance of our algorithm does not change. The reason for this is simply both z and z' have been trimmed by the same value of x. Therefore, by selecting a suitable value for approximation factor, the system tuner can reduce the negative effects of *EstErr*. To measure the quantitative sensitivity of PANDA in regards to the variability in both *EstErr* and *PrfDeg*, more experiments were conducted. To resemble the behavior of a real BoT application, we take advantage of BoT application characteristic patterns reported by [241] and [225] derived from the analysis of several real BoT applications. The task lengths in this model are expressed by 2^b (in minutes), which b follows either a (1) uniform or (2) normal distribution with attributes given in Table 5.6. The task length is defined as how much CPU time is required to run on a single core machine ($m1$.small).

Figure 5.5 presents a comparison of the $Farness$ achieved by PANDA with different values of ϵ and φ with respect to the different values of *EstErr*. We only show experimental results for BoT application with long task lengths as the same pattern was observed in two other types. In the curves related to the smaller values of ϵ ($\leq$10%) and/or φ (<10 min), the $Farness$ values grow gradually at the beginning, e.g., error is less than 20%. After this, there is a sharp increase in $Farness$ which shows a poor performance. Conversely, for larger values of ϵ (>10%) and φ ($\geq$10 min), the $Farness$ values are higher than before at the start, and then increases slowly at the end. It concludes the larger values of ϵ and φ can cope better with bigger error values of *EstErr*, while reach poor performance at less-erroneous environment. In other words, to achieve a better performance in an erroneousness condition, values of ϵ and φ must be carefully chosen higher than normal.

To study the impact of *PrfDeg*, we first introduce the average aberration parameter. Suppose that the performance level (speed) of resource i degrades to $\hat{s}_i$ from the predetermined value of s_i, during the execution of application. An assigned task with length P_j would take $P_j/\hat{s}_i$ time units to finish, instead of presumed time of P_j/s_i. *Average aberration* (in percentage) over all available resources is defined as: ($k' = |\Gamma \cup \{v\}|$)

Table 5.6: **BoT task length characteristic. n=100,000.**

BoT type	Task length $=2^x$(minute)
Short	$x \sim$U(0,3)
Long	$x \sim$N(3.5,3)
Mixture	$x \sim$N(1.8,3)

$$\bar{\nu} = (\sum_{i \in \Gamma \cup \{v\}} |\hat{s}_i - s_i|/s_i)/k' \tag{5.10}$$

Table 5.7 presents the achieved *Farness* of PANDA for different values of $\bar{\nu}$ when tasks of BoT application are all long. To create a working condition with a target degraded performance in a resource, we ran a configurable `process` to consume CPU clocks deliberately.

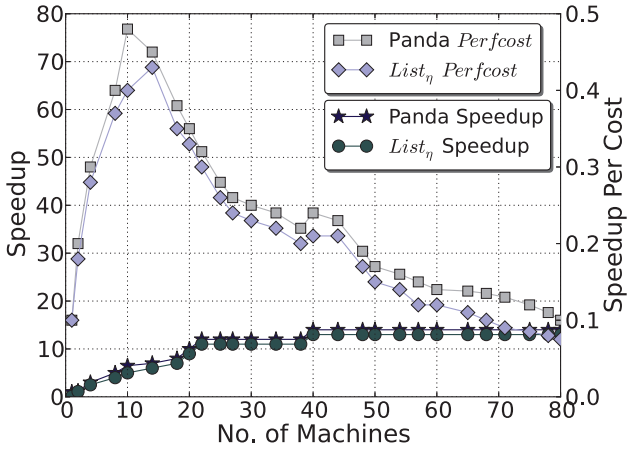
Our results show that two parameters of ϵ and φ can effectively cope with application or resource performance uncertainties. When $\bar{\nu}$ value is small ($< 10\%$), small chosen values for ϵ and φ can effectively lead to an acceptable performance. A large value for two parameters need to be chosen in an environment with higher level of $\bar{\nu}$.

Table 5.7: **Sensitivity of PANDA Farness to the aberration percentage ($\bar{\nu}$).**

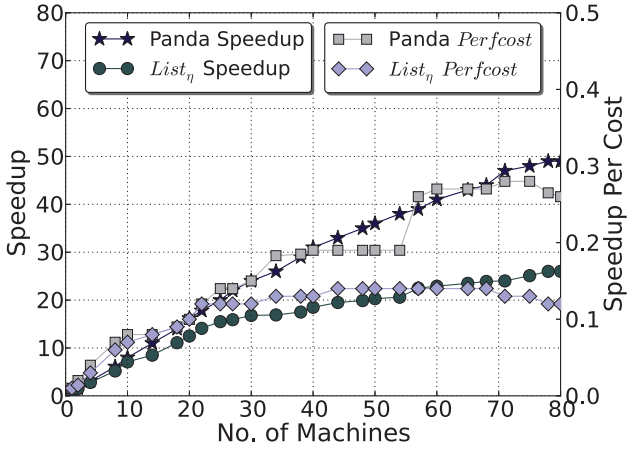
$\bar{\nu}$(%)	ϵ(%)	φ(min)	F(%)	$\bar{\nu}$(%)	ϵ(%)	φ(min)	F(%)
3*5	5	5	12	3*10	5	5	16
	10	10	15		10	10	19
	20	10	19		20	10	22
3*15	5	5	23	3*20	5	5	30
	10	10	23		10	10	30
	20	10	22		20	10	26

5.10 Summary

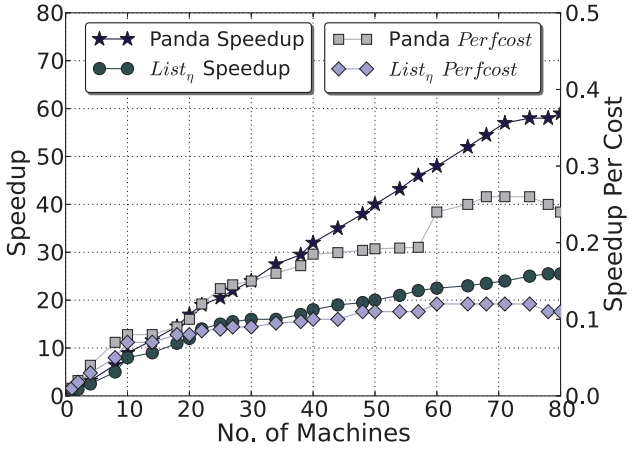
The massively and horizontally scalable nature of large-scale BoT applications in many science, engineering and business analytic applications often requires computing capacity beyond the capacity of a single computer system. Expanding the system is not a practically viable solution due to both capital and operating costs. Here, cloud computing comes into picture with elasticity and pay-as-you-go utility model. In this Chapter, we have presented PANDA framework for cloud bursting, and developed an FPTAS (for scheduling) as the essential component of the framework. We have shown that cloud bursting can be greatly leveraged by optimizing the performance to cost ratio. Our scheduling algorithm guarantees the deviation of such cost efficiency from the optimum to be bounded by a tunable parameter (approximation factor, ϵ). The identification of Pareto-frontier plays a crucial role in capturing tradeoff between performance and cost. Our framework with its scheduling algorithm has been thoroughly evaluated with extensive experiments using ISOMAP in real cloud settings. Experimental results prove that our solution can effectively leverage cost efficiency in clouds. We believe that with the increasing adoption of cloud computing, cost efficiency solutions like PANDA for cloud bursting have great practical importance.



(a)



(b)



(c)

Fig. 5.4: **Scheduling performance in terms of *Speedup* and *perfCost* metrics with respect to different BoT applications comprising of tasks with (a) Short, (b) Mixture, and (c) Long lengths**

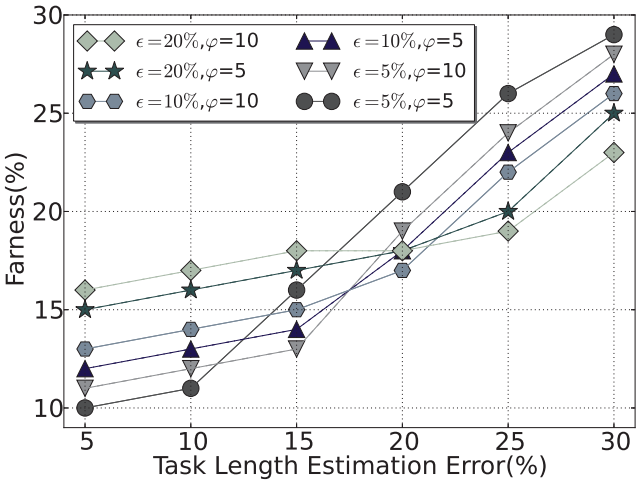


Fig. 5.5: **Farness vs.** *EstErr* **for different** φ **(min.) and** ϵ**(%). Task lengths are long, *i.e.*,** $r = 1$**.** $n = 100,000$**, and** m**=100.**

Chapter 6

Non-clairvoyant Pareto-Efficient BoT Scheduling

"Success in life is the result of good judgment. Good judgment is usually the result of experience. Experience is usually the result of bad judgment which people called failure."
–Anthony Robbins

6.1 Introduction

Research on resource management in such an hybrid environment often relies heavily on having complete and accurate information regarding relevant parameters, e.g., tasks, resources, etc. In practice, however, these pieces of information are not readily available. Three main issues of resource management in hybrid clouds are described as follows.

Coping with Parameter Uncertainty: Characteristics of tasks, such as the running time, are not given *a priori*, and therefore need to be estimated using other sources, such as historical data, profiling and analysis tools, [242–246], and [247]. Despite these developments, achieving the optimal resource allocation under uncertainty remains a challenging problem.

Satisfying Conflicting Objectives: Different interpretations of optimality from the user's perspective add more complexity to task scheduling in the cloud. On one side, the user has two conflicting objectives (makespan and total cost), subject to certain deadline or budget constraints. Such a conflict often makes finding

a solution that simultaneously optimizes both objectives impossible resulting in heuristic approaches (e.g., [248, 249]). On the other side, a compromise between these two conflicting objects is often not available to the user in advance. Although it is normally assumed that the user has a utility function, $\mathcal{U}(time, cost)$, to represent numerically his/her preference on the time and cost, deriving such a utility function is often a tedious and complex process for a normal user, and therefore not always possible.

Achieving Allocation and Scheduling Optimality: For a large scale application, allocating hybrid cloud resources respecting a predefined service level agreement is a complex, costly and time consuming matter. As estimated in [250] and [251], more than 70% of the workload in parallel systems and 96% of that in grid systems are comprised of BoT applications. Such applications are composed of a vast number of very loosely-coupled tasks and thus demonstrate massive parallelism. As the number of resources in a given in-house system is often limited, it is imaginable that the performance of task execution can be greatly enhanced by dynamically renting resources from the virtually unlimited capacity of public cloud providers.

To address these three issues, in this section we develop and evaluate a dynamic task scheduling and resource allocation algorithm for Bag-of-Task (BoT) applications running on a hybrid cloud by providing the following novel contributions.

1. **A hybrid approach for running time estimation**, accompanying a feedback control system, that is designed to cope with the uncertainty in the estimating of running time via an analysis of multiple estimation tools.
2. **An *a posteriori* possible-schedule technique** that presents to the user possible Pareto-optimal schedules and allows the user to make an informed decision about his/her preferred assignment by selecting among them.
3. **A resource allocation process** that utilizes our hybrid running time estimation process and dynamically rents public cloud resources in order to approximate the user's chosen Pareto-optimal solution.

Experiments conducted on a testbed hybrid cloud (using Amazon EC2 as a public cloud) show that the final scheduling output delivered by our resource allocation process achieves a solution with 2% of the user's chosen Pareto-optimal point. Moreover, our approach outperforms an extended *List* scheduling approach by reducing both makespan and total cost by around 20% and 5%, respectively.

The rest of this section is organized as follows. We introduce the model used in this chapter as well as a formal definition of the resource allocation problem in Sections 6.2 and 6.3, correspondingly. Our resource allocation solution algorithm is presented in Section 6.4. Finally, experimental results are presented and discussed in Section 6.5.

6.2 Models

The benefit of resource allocation under uncertainty is widely recognized within the literature. Exact information about the characteristics of applications or resources is rarely available, and thus much effort is required to properly address this problem. The models for public cloud, private cloud, and Bag of Task application follow the model presented in Section 3.4. In this work, we assume that the user has a budget, $\mathcal{B}$, and a deadline to finish the application, $\mathcal{D}$. Both of these parameters are known in advance. Since we deal with an application comprised of CPU-intensive tasks, network latency between the scheduler (or private machines) and the public cloud instances is assumed to be negligible compared to the actual running time of the application. For example, when running ISOMAP during our experiments, transferring data between instances in our hybrid cloud took only 5 minutes, while the total running time of the application is almost 20 hours. Table 6.1 summarizes a list of notations and abbreviations that used in this section. In the following we describe the model for estimation tools.

6.2.1 Estimation Tools Model

Numerous studies have addressed the topic of task running time estimation in a given application. The tools presented in these studies mainly use either analytical benchmarking or code profiling techniques which analyze the source code of the given tasks to obtain the behavior of tasks in terms of a set of parameters. There are also several methods based on statistical and probabilistic techniques to estimate the execution time of tasks [252]. We assume that our system has access to a set of different estimation tools, denoted by $\mathcal{I} = \{\mathcal{I}_1, \mathcal{I}_2, \cdots, \mathcal{I}_\tau\}$, and the estimated execution time of task j by using a tool of type t will be denoted by $\hat{p}_{j,t}$.

Table 6.1: **Frequently used symbols and notations**

Γ_u (Γ_v)	Set of resources in p**u**blic (pri**v**ate) cloud
k (k')	Number of resource types in public (private) cloud
c_i	Related cost of resource type i, (in one ATU)
s_i	Related speed of resource type i
L_i	Resource limit of type i, i.e., #resources
$\mathcal{D}$, $\mathcal{B}$, $\mathcal{U}$	User's Deadline, Budget, Utility function
n, m, τ	Numbers of tasks, resources, estimation tools
p_j	Actual running time of task j
$\mathcal{I}_t$	Estimation tool of type t
T_r ($\mathbb{E}$)	r^{th} (set of all) Pareto-point submitted to user
$b_{m,t,j}$	Est. time of m^{th} estimation point by tool t for task j
$b^*_{m,t,j}$	Actual time of m^{th} estimation point by tool t for task j
H	Number of scheduling intervals
ψ^h_t	Accuracy weight of estimation tool t at time interval h
$\hat{p}^h_i$	Estimated running time of task i at time interval h
f	Number of most cost efficient resources
w^h_i	Assigned workload to machine i at interval h

6.3 Problem Formulation

In the context of cloud bursting, the user faces two objectives: minimizing total costs (c) and maximizing performance (or minimizing makespan). Normally, there is no unique solution that simultaneously reaches the optimum value of both conflicting criteria, and so the user must sacrifice the quality of at least one of those two criteria. In such a situation, the concept of Pareto optimality becomes helpful.

Selection of one scheduling point in a Pareto frontier is subjective to a user's particular interest. An accurate preference function for a user is normally difficult to obtain, however. Furthermore, an arbitrarily complex utility function can make the scheduling problem intractable. In this paper, we propose a more convenient and practical way to deal with the issue, as discussed below.

Our system provides a list of all possible Pareto-optimal points and lets the user choose a desirable resource allocation based on his/her desired utility function. To align our idea with reality, we argue that Pareto-optimal points with an integer completion time are a good candidate set to deliver to the user. Such a statement is true when the effect of an ATU-based pricing scheme exists. Figure

5.2 shows several scheduling points when a real ISOMAP application runs in a hybrid cloud (using Amazon EC2 as public cloud). Each of these points yields different makespans and total costs, and it can be observed there are no Pareto points within some subranges, such as the interval (21,21.5), as all of the scheduling options are dominated by the Pareto point annotated by T_2. This fact helps us identify a good candidate set of Pareto schedules to deliver to the user, i.e., points with integer completion times.

Let $\mathbb{E}$ denote the set of Pareto-optimal points that should be delivered to the user, and let T_{lower} and T_{upper} represent the best and worst makespan achievable for the execution of a given BoT application. We define $T_0 = T_{lower}, T_{|\mathbb{E}|} = T_{upper}$ and $T_r = \lfloor T_{lower} \rfloor + r\Delta T - \varepsilon$, for $0 < r < |\mathbb{E}|$. Here, r and ΔT are positive integer numbers (to keep the time step) and ε is a small positive number used to maintain a gap, e.g., $\varepsilon = 0.05$. T_r points are the end-segments of each subrange when the interval $[T_{lower}, T_{upper}]$ is divided to $|\mathbb{E}|$ subranges (see Figure 5.2). Therefore, we strive to find the set of T_r points and deliver them to the user as the best possible Pareto-frontier scheduling. In Section 6.4.2, we discuss in detail how to calculate T_{lower}, T_{upper}, and the other T_r points based on the available information.

6.4 Resource Allocation and Task Assignment Strategy

In this section, we will construct and present our proposed solution. Our resource allocation process begins with the estimation of each task's execution time, and thus facilitates the location of possible Pareto points. The estimation of task running time is made by employing estimation tools. To improve the measurement of each estimation tool, we devise a method that adds several breakpoints to each task. Following this, we can monitor and compare the actual running time of breaking points with the estimation values given by the profiler, and use the breakpoints to assign an accurate weight to each tool. To increase the weight accuracy of each profiler, we divide the whole time horizon into equal intervals. At the beginning of each interval, a monitoring phase happens, during which the actual revealed running time and the estimated running time are compared to evaluate the accuracy of each estimation tool. To assign a weight to each profiler tool, we bring in a simple yet effective equation which will be discussed in detail in Section 6.4.1.

The next step of the proposed algorithm is to allocate the available resources. In this phase, a schema of Pareto-efficient resource allocation is generated. Then, the workload of each machine is calculated based on the previously generated information, followed by generating the possible Pareto scheduling points. Jobs are then assigned to resources by engaging a technique inspired by the First Fit Decreasing (FFD) technique. Section 6.4.2 provides the details of this process.

6.4.1 Running Time Estimation

One of the riskiest aspects of any provisioning system lies in the estimation phase. This is due not only to the inaccuracy of many estimators, but also to the increasing complexity of software applications over time. To make it more challenging, input parameters that form the running time estimation tools are not unique for all type of applications, making them more difficult to calculate for new ones. To cope with this growing challenge, we propose a hybrid approach that uses existing estimation techniques in an iterative fashion. At the heart of this technique lies previous analytical-based procedures that endeavor to estimate the running time of each task, based on the observed data and available prior experience (such as ATOM [253], Pin [254], and Valgrind [255]).

While sophisticated methods can be applied in a more complex system, we introduce a simple technique to describe our hybrid method for the scope of this paper: we assume that each tool inserts extra code, called an estimation point (or instrumentation point), into the target task to observe its behavior and collect some information about its running time. For each tool of type t, we define a set of points $b_{m,t,i}$ to represent the running time of task i up to the m-th estimation point estimated by this tool. Let $b^*_{m,t,i}$ denote the actual running time of task i at this point. Here, $b^*_{m,t,i}$ is set to zero at system startup, and the concrete values of $b^*_{m,t,i}$ are observed and collected throughout the task running phase.

We use $b_{m,t,i}$, $b^*_{m,t,i}$ and their difference values to evaluate the accuracy of tool t when estimating the running time of task i. To do this, the time horizon is divided into H intervals of equal duration. At the beginning of each interval h, $0 \leq h \leq H$, we assign a weight value, $0 \leq \psi_t^h \leq 1$, to each tool t that measures the accuracy of this tool relative to the others. A larger weight indicates a tool that has estimation closer to the actual value, while a smaller weight refers to a tool that has estimation far from the actual value. In addition, the values of ψ_t^h are updated based on the values of $b_{m,t,i}$ and $b^*_{m,t,i}$, according to the following equation:

$$\psi_t^h = \frac{\sum_{b^*>0} \frac{b^*_{m,t,i}}{|b^*_{m,t,i} - b_{m,t,i}|}}{\sum_{t'=1,\cdots,\tau} \sum_{b^*>0} \frac{b^*_{m,t',i}}{|b^*_{m,t',i} - b_{m,t',i}|}} \tag{6.1}$$

The values of ψ_t^h are set initially to $\psi_t^0 = \frac{1}{\tau}$. Based on the calculated ψ_t^h, the following weighted sum model is used to estimate the execution time of each task i at the beginning of each interval h, $0 \leq h \leq H$:

$$\hat{p}_i^h = \sum_{t=1,\cdots,\tau} \psi_t^h \times \hat{p}_{i,t} \tag{6.2}$$

Here, $\hat{p}_{i,t}$ denotes the estimated value of running time of task i by using tool of type t.

As modern well-known estimation tools heavily rely on quick-enough techniques based on some kind of pattern detection, historical data analysis, and source code profiling, we assume that the required time to estimate the running time, which happens once at system startup, is negligible compare to the running time of the application. For example, in our experiments, a group of estimation tools, including ATOM [253], was employed in the estimation phase of an ISOMAP application comprised of 100k tasks. The overhead of this phase was between 8∼20 minutes, compared to the total application running time of 20 hours.

6.4.2 Pareto-Efficient Point Generation

In this section, we describe how to generate Pareto-efficient resource allocation points based on the data generated during the estimation phase in Section 6.4.1. As we have discussed in Section 6.3, finding both T_{upper} and T_{lower} are the first steps towards generating the set of all Pareto-optimal points. It can be proven that if there is no limitation on either user's budget or deadline, the T_{upper} solution can be achieved when all tasks run only in resources of the most cost effective type, i.e., the resource type that has the highest value of $\frac{s_i}{c_i}$. Furthermore, such a solution normally results in a least-total-cost schedule.

Let f denote the number of the most cost effective available resources in the hybrid cloud. T_{upper} can then be calculated as follows:

$$T_{upper} = \frac{\sum_{j=1,\ldots,n} p_j}{\sum_{i'=1,\ldots,f} s_{i'}}$$

However, the exact running time p_j is not known *a priori*. Instead we use the following formula to calculate the workload of machines at the beginning of each interval h:

$$w_i^h(\sigma_{upper}) = s_i \min(\mathcal{D}, \frac{\sum_{j=1...,n} \hat{p}_j^h}{\sum_{i'=1,...,f} s_{i'}})$$

where $\mathcal{D}$ represents the user's deadline, if given.

In a similar fashion, when a load balanced solution among all available resources is employed, T_{lower} is achievable; that is, each resource type i receives a load proportional to the value of $s_i / \sum_{\kappa \in \Gamma_u \cup \Gamma_v} L_\kappa s_\kappa$. This schedule normally yields the lowest makespan with the largest total cost. In other words, the following formula can be seen as a good lower bound of T_{lower}:

$$T_{lower} \geq \frac{\sum_{j=1,...,n} p_j}{\sum_{\kappa \in \Gamma_u \cup \Gamma_v} s_\kappa}$$

6.4.3 Resource Allocation

With the values of T_{lower} and T_{upper} defined in the preceding section, we can calculate each T_r, $0 < r < |\mathbb{E}|$, using the method addressed in Section 6.3. As discussed in that section, these T_r points are the possible Pareto scheduling points based on our estimation and the aim of this paper is to find a schedule, σ_r, which closely approximates a user chosen T_r from those submitted. To determine σ_r, which optimally has time not exceeding T_r while achieving the minimum total cost, we calculate the maximum allotted workload of each resource i in such a schedule, using the following formula. This calculation is performed at the beginning of each interval h:

$$w_i^h = T_r \times s_i \sum_{j=1}^{n} \hat{p}_j^h - \sum_{i'=1}^{i-1} T_r s_{i'} \geq T_r s_i \sum_{j=1}^{n} \hat{p}_j^h - \sum_{i'=1}^{i-1} T_r s_{i'} \tag{6.3}$$

We assume that resources are sorted as a list, $\langle M_1, M_2, \cdots, M_m \rangle$, by their cost efficiency (s_i/c_i) in descending order.

The next step of resource allocation is to find a subset of tasks that add up as close as possible to the value of w_i^h defined by Equation 6.3. This can be considered as a general case of either bin packing, knapsack, subset sum, or partition problem, which are all shown to be NP-complete (see e.g., [256]).

While there are many good approximation algorithms to solve these fundamental problems, these algorithms are typically coupled with high complexity and computational overhead. Therefore, we will instead use a simple approach which is fast, easy to implement, and generates fairly good results in many cases. This approach, inspired by the well known First Fit Decreasing technique [257] is summarized formally in Algorithm 7.

Algorithm 1 aims to decrease the amount of wasted processing capacity by those machines which are already allocated. At the beginning of each interval, the algorithm detects those machines which have already run some tasks. The variable ρ_i keeps track of the current amount of workload for each resource i. The algorithm then assigns each task j to the first available resource which has sufficient residual capacity workload according to Equation 6.3. To improve the efficiency of our algorithm, we first begin by assigning larger tasks to the most cost efficient resources. One can show that this algorithm is a Constant Factor Approximation algorithm close to 11/9 [257], which carries a reasonable efficiency for the purpose of this work.

Our proposed schema for resource allocation of a hybrid cloud environment is formally presented in Algorithm 8. Here, we first find an estimation for the running time of all tasks using results available via an analysis of each estimation tool. Based on these estimation results, the algorithm generates a set of possible Pareto points accounting for both the user's deadline and budget. After the user chooses a desirable Pareto point, the algorithm tries to approximate this point as closely as possible. To do so, the time horizon is divided into several equal intervals. At the beginning of each interval, the algorithm assigns a weight to every estimation tool, based on the correctness of each tool's estimation values. Afterwards, the amount of workload for each resource is recalculated. Additionally, at the start of each interval, tasks are assigned to the proper resources using Algorithm 7.

It can be shown that running time of Algorithm 7 is bounded by $O(n^2)$ in the worst case scenario. Hence, the running time of Algorithm 8 is $O(n^2h)$, which is very fast (a fraction of minute) compared to the running time of an actual BoT application (several hundred hours).

6.5 Evaluation

In this section, we provide the performance evaluation report of our approach using a BoT application on a testbed infrastructure composed of real, mixed cloud

Algorithm 7: Resource allocation algorithm

input : $h, n, m, \hat{p}_j^h, \Gamma_u, \Gamma_v, T_r, w_i^h$
output: $x_{i,j} \in \{0,1\}; \forall 1 \leq i \leq m, 1 \leq j \leq n$
// x_i, j indicates the tasks j is assigned to be run on resource of i.
begin
  sort not yet run tasks by $\hat{p}_j^h$ in *descending* order
  //(obtained sequence: $\langle J_1, J_2, \cdots, J_{j'} \rangle$)
  sort resources by efficiency in *descending* order
  //(obtained list: $\langle M_1, M_2, \cdots, M_m \rangle$)
  ρ_i: current workload on resource i
  for $j \in \{1, ..., j'\}$ **do**
    for $i \in \{1, ..., m\}$ **do**
      if $\rho_i + \hat{p}_j^h \leq w_i^h$ **then**
        run task J_j on resource i
        $\rho_i \longleftarrow \rho_i + \hat{p}_j^h$
        set $x_{i,j} \longleftarrow 1$
        break;
      end
    end
  end
end

infrastructure, following the hybrid cloud model described previously in Section 3.4.

6.5.1 Experimental Setup

The private cloud used in our experiments consists of a 40-core (80 logical cores) cluster composed of four 2.40GHz Intel(R) Xeon(R) E7-8870 processors, with a total of 256GB of RAM. The communication overhead between these processors can be considered zero in our experiments. Amazon EC2 is used as the public cloud. The instance types rented, and the characteristics of these instances, are summarized in Table 6.2. In this table, the processing capacity of a resource (third column) is defined as how much work a resource performs in a given amount of time, e.g., an hour, relative to a reference machine (*m1.small*). To determine this parameter, we constructed a small and fast benchmark that measures how fast a resource completes a given algorithm on specific sets of data.

We create several different sets of BoT workloads to run on our hybrid cloud. To mimic the behavior of real BoT applications, we use different sets of BoT characteristic patterns as reported by Iosup et al. [258] and Tran [259], which was derived from the analysis of seven real BoT applications. Table 6.3 details

Algorithm 8: Proposed algorithm

input : n, m, Γ_u, Γ_v
output: scheduling schema
begin
  user submitted BoT, $\mathcal{D}$, $\mathcal{B}$ (if any)
  for *each estimation tool* $t \in \{1, ..., \tau\}$ **do**
    $\psi_t^0 = 1/\tau$
    for *each task* $j \in \{1, ..., n\}$ **do**
      calculate $\hat{p}_{j,t}$
      calculate $b_{m,t,j}$ points
    end
  end
  calculate T_r's and submit to the user
  T_{r^*}: user's selection
  divide interval $[0, T_{r^*}]$ to H equal parts
  for *each* $h \in \{0, \cdots, H\}$ **do**
    record $b^*_{m,t,j}$
    for *each tool* $t \in \{1, ..., \tau\}$ **do**
      update ψ_t^h from Equation 6.1
    end
    for *each task* $j \in \{1, ..., n\}$ **do**
      update $\hat{p}_j^h$ from Equation 6.2
    end
    run Algorithm 7
    if *a task is completed on resource* i
    then
      if $x_{i,j} = 1$ **then**
        dispatch largest estimated task j
      end
    end
  end
end

a summary of this model. For example, the number of tasks in a BoT can be expressed by a^b, where a is often equal to 2 and b follows a Weibull distribution with parameters given in the table. Likewise, the running time of tasks is a^b minutes, where a is similarly equal to 2 and b follows either an uniform or a normal distribution with average and standard deviation given in Table 6.3. The running time of each task is defined in terms of how much CPU time is required for its execution on a single core machine in a small instance (as a reference). Depending on the number of tasks (Small, Medium, Large), and task length characteristics (Short, Long, Mixture), we can define nine classes of application: *SS, MS, LS*, and so on. However, BoT workloads must contain a large number of tasks with order

of thousands up to hundred of thousands tasks for a realistic and representative evaluation result. So, we focus more on examining results in a large-sized BoT environment, i.e., LS, LL and LM from Table 6.3.

6.5.2 Scheduling Policies

We compare our proposed algorithm with an extended version of *List* scheduling. *List* scheduling is one of the most common scheduling policies currently used in parallel systems. It is based on the idea of making an (ordered) list of ready tasks (that may have assigned priorities), and then repeatedly executing the following two steps: (1) choose the (highest prioritized) available task from the list, (2) choose a free machine to accommodate and run this task. The priorities can be determined in different ways. Two of the most widely used are Longest running time and Random (or without priorities), both of which were applied in this study.

The basic version of *List* scheduling is not truly comparable with our algorithm, as it only produces one point in the time-cost plane while ours produces several Pareto points. To overcome this issue, we simply extend the *List* policy as follows. Assume that there are m available machines for service in the hybrid cloud. We define $List_\eta$ to represent a schedule based on the *List* policy that uses

Table 6.2: **Hybrid cloud settings with different cost to performance ratios (Relative costs are calculated based on the cost of m1.small).**

Cloud	Res. Type	Proc. Capacity	Hourly Cost
	m1.small	1	\$0.080
Amazon EC2	c1.medium	5	\$0.165
US East (VA)	m1.large	4	\$0.320
	c1.xlarge	20	\$0.660
Private	4x10-core Xeon	10	\$0.320

Table 6.3: **BoT Workload characteristic used in this study.**

Type (BoT size, Task Running Time)	No. Tasks $=10k \times 2^x$	Task Length $=2^x$(minitue)
LS (Large, Short)	x ~Wbl(1.7,2)	x ~U(0,3)
LL (Large, Long)	x ~Wbl(1.7,2)	x ~N(3.5,3)
LM (Large, Mixture)	x ~Wbl(1.7,2)	x ~N(1.8,3)

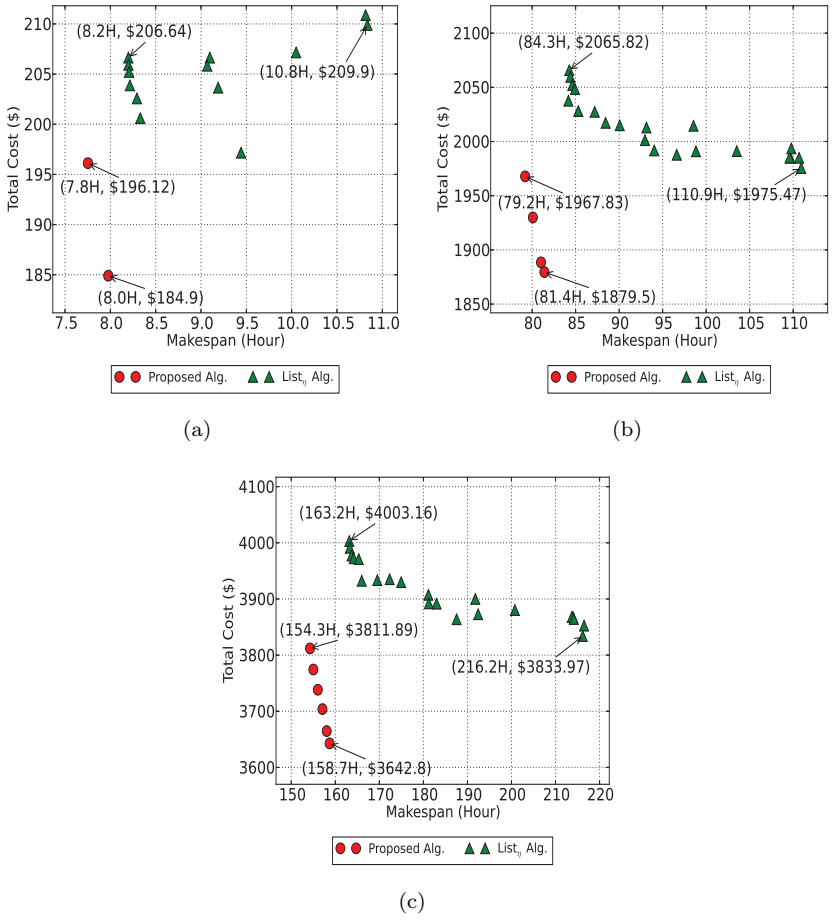


Fig. 6.1: **Performance of our proposed algorithm vs. that of $List_\eta$ scheduling with a large-size BoT job (n = 100,000). Scenarios corresponding to tasks with (a) short running time (LS), (b) a mixture of both short and long running time (LM), and (c) long running time (LL) are included. For all experiments, $\tau = 2$ and the maximum error of each estimation tools is equal to %10. The number of machines taken from each instance type is set to 20 (100 total machines).**

only the η most efficient machines for resource allocation ($\eta \leq m$). This extended version of $List_m$ is clearly similar to traditional *List* scheduling.

6.5.3 Results

In this section, we present a comparative performance evaluation of both our method and extended *List* scheduling, in terms of both total cost and makespan. As previously mentioned, this comparison is performed for a large-size BoT with several hundred thousands tasks, each of them having either short or long running time. Figure 6.1 shows a comparison of the makespan and cost of both scheduling

algorithms. The Pareto points found by our proposed algorithm (red circle points) dominate all those found by extended *List* scheduling (green points). For example, in scenario LL_{20}, our algorithm recommends and finds a schedule with cost and speed equal to 154.3 hours and \$3812, respectively. This schedule uses all available machines (equal to 20 machines here) in a load-balanced manner. $List_m$, however, finds a comparable scheduling point with cost of \$4003 and makespan of 163.2 hours. i.e., a decrease of 5%∼6% in both time and cost. In another extreme, our scheduling algorithm finds a strategy with total cost equal to \$3642.8 and makespan equal to 158.7 hours, while the *List* algorithms can achieve at best a schedule with cost of \$3834 and makespan of 216.2 hours. In this case, both algorithms try to use only the most efficient machines, but our algorithm shows 36% and 5% improvement in make-span and cost, respectively.

Table 6.4: **Achieved makespan, total cost, their differences (Diff.) and percentage differences (Diff.%) of the proposed algorithm versus $List_\eta$ for the extreme Pareto-point of T_{lower}. Subscripts denote number of machines. For all experiments, $n = 100,000$, $\tau = 2$, and the maximum tool's error percentage is 10%.**

Scenario	T_{lower}: Best Makespan Scheduling			
	Our(Hr/\$)	$List_\eta$(Hr/\$)	Diff.(Hr/\$)	Diff.(%/%)
LS_{20}	7.7/196.1	8.2/206.6	0.5/10.5	6.5/5.4
LS_{60}	3.8/196.8	4.1/205.9	0.3/9.1	7.9/4.6
LM_{20}	79.2/1967.8	84.2/2037.6	5.0/69.8	6.3/3.5
LM_{60}	39.7/1968.0	42.4/2067.5	2.7/99.5	6.8/5.1
LL_{20}	154.3/3811.9	163.2/4003.2	8.9/191.3	5.8/5.0
LL_{60}	77.2/3834.8	82.2/4024.8	5.0/190	6.5/5.0

The difference between the cost and makespan of *List* schedules and those found by our algorithm are calculated and summarized in Tables 6.4 and 6.5. The subscript in this table indicates the number of machines (rented from each instance group) used to run the scenario. Results verify that our approach can successfully reduce both the time and cost of using hybrid cloud resources as a whole. The average reduction of makespan and cost usage in cloud machines is approximately 17% and 6%, respectively. The performance of the two scheduling policies is only close if the user asks for the best-makespan point (although our approach performs better by up to 11%). It is a predictable situation as one needs to balance the whole workload among all resources, which both algorithms can do well.

Table 6.5: **Achieved makespan, total cost, their differences (Diff.) and percentage differences (Diff.%) of the proposed algorithm versus $List_\eta$ for an extreme Pareto-point of T_{upper}. Subscripts denote number of machines. For all experiments, $n = 100,000$, $\tau = 2$, and the maximum tool's error percentage is 10%.**

Scenario	T_{upper}: Best Cost Scheduling			
	Our(Hr/$)	$List_\eta$(Hr/$)	Diff.(Hr/$)	Diff.(%/%)
LS_{20}	7.9/184.9	9.4/197.2	1.5/12.3	19.0/6.7
LS_{60}	4.0/186.6	4.2/201.7	0.2/15.2	5.0/8.1
LM_{20}	81.4/1879.5	110.9/1975.5	29.5/96.0	36.2/5.1
LM_{60}	40.7/1881.2	53.3/1997.6	12.6/116.4	31.0/6.2
LL_{20}	158.7/3642.8	216.2/3833.9	57.5/191.1	36.2/5.2
LL_{60}	79.4/3667.4	110.0/3862.0	30.6/194.6	38.5/5.3

Alternatively, any scheduling algorithm which seeks to reach other Pareto points with less cost must calculate and keep track of both the current and future assigned workload of each resources. Because the *List* algorithms do not possess such a mechanism, they are often unable to achieve such Pareto points (see Figure 6.1 and Tables 6.4 and 6.5). For instance, if the best-cost schedule is desired, our approach achieves 27% lower makespan and 6% lower cost.

Another notable observation is that the performance of *List* algorithms degrades when the ratio of long tasks increase within the workload. Let us define *workload mixture ratio* r as the percentage of long-size tasks in a given BoT, i.e., $r = \frac{|Longtasks|}{|totaltasks|}$. This ratio can be considered as the granularity of a BoT application. For a given BoT application, $r = 1$ indicates that it solely consists of long tasks whereas $r = 0$ indicates that it is composed solely of short tasks. In cases where the ratio is close to zero (LS scenario), the *List* algorithms perform satisfactorily (a percentage difference of around 9% in makespan and 5% in total cost), while in cases where this ratio is close to one (scenarios LM and LL), the *List* algorithms fail to reach a good solution (with degradation of more than 5% in total cost and 22% in makespan). The reason for this phenomena can be explained as follows. Our proposed algorithm calculates the proper workload of each resource for each value of T_r, and tries to assign tasks to each resource in a way that does not put more extra load than the pre-calculated value. On the other hand, the *List* algorithms blindly take as many tasks as possible, often leaving an empty workload on some resources and crushing other resources with extra workload.

6.5.4 Evaluating the Achievement of Service Levels

As discussed earlier, the service level agreement with the user must indicate both the maximum time and the total cost of accomplishing the entire application. To evaluate whether the objectives of this agreement have been met, we run scenarios LM_{20} and LL_{20} using two estimation tools, each of them with two error modes (10% and 30%), and calculate the differences between the total time and cost spent to accomplish the user's request. Table 6.6 shows the concrete values of such an experiment. The aberration from the initial commitment is less than approximately 2%. Considering the fact that in a normal application of BoT there is neither a hard deadline for the time or the budget, we believe that this imperfection in quite acceptable from the user's perspective.

6.6 Summary

In this chapter, we have presented a novel scheduling algorithm under uncertainty for the purpose of cloud bursting a large-scale BoT application. We have discussed how the identification of a Pareto frontier plays a crucial role in capturing the tradeoff between performance (makespan) and cost from the user's perspective. In our approach, we allow the user to choose a desired resource allocation from a delivered list of all possible Pareto-optimal points without expressing his/her utility function to the system. Our algorithm then tries to achieve the user's selection as closely as possible. Our approach has been thoroughly evaluated through extensive experiments running realistic BoT applications in real hybrid cloud settings. For future work, we plan to extend our algorithm to cover the arrival pattern of BoT, as well as to consider network latency between the scheduler (or private network) and the public cloud network. We have also left the development of data-intensive workloads for future work. We believe that with the increasing adoption of cloud computing, cost-efficiency solutions like those presented in this work have great practical importance.

Table 6.6: **Quality of achievement of the user's Service Level Agreement illustrating, for specific T_i, the percentage difference (Diff. %) between the makespan and cost of the initially proposed Pareto-point, and that finally delivered. Here $n = 100,000$, $m = 20$ and $\tau = 2$. Er denotes the maximum error percentage of each estimation tool.**

T_0 Pareto-point			
Scenario	T_0		
	Proposed (Hr/$)	Delivered (Hr/$)	Diff. (%/%)
$LM_{20,Er=10\%}$	78.9/1957	79.2/1967.8	0.4/0.6
$LM_{20,Er=30\%}$	78.9/1958	79.9/1967.8	1.3/0.5
$LL_{20,Er=10\%}$	153.9/3802	154.3/3811.9	0.3/0.3
$LL_{20,Er=30\%}$	153.9/3803	155.8/3812.7	1.2/0.3

T_2 Pareto-point			
Scenario	T_2		
	Proposed (Hr/$)	Delivered (Hr/$)	Diff. (%/%)
$LM_{20,Er=10\%}$	80.9/1880	81.1/1890.7	0.2/0.6
$LM_{20,Er=30\%}$	80.9/1881	81.1/1890.9	0.2/0.5
$LL_{20,Er=10\%}$	157.9/3726	158.3/3737.9	0.3/0.3
$LL_{20,Er=30\%}$	157.9/3727	159.2/3738.5	0.8/0.3

$T_{\lvert E\rvert}$ Pareto-point			
Scenario	$T_{\lvert E\rvert}$		
	Proposed (Hr/$)	Delivered (Hr/$)	Diff. (%/%)
$LM_{20,Er=10\%}$	81.2/1868	81.4/1879.5	0.3/0.6
$LM_{20,Er=30\%}$	81.2/1869	81.4/1879.5	0.3/0.6
$LL_{20,Er=10\%}$	157.9/3630	158.7/3642.8	0.5/0.4
$LL_{20,Er=30\%}$	157.9/3632	159.2/3665.7	0.8/0.9

Chapter 7

Conclusion and Future Work

7.1 Summary of Thesis

Infrastructure-as-a-Service (IaaS) cloud is typically referred to a type of virtual machine that bundles a certain amount of computing, memory, and I/O bandwidth capacity with an unchanging price schema throughout its lifetime. This new model of computing, such as Amazon EC2, allows anyone worldwide to access almost unlimited source of computing power by having an internet access and a credit card in a pay-as-you-go manner. Before cloud computing, users have to either purchase their own servers (which its useful lifetime was around three years), or rent a remote hosting server on a long-term basis (e.g. monthly). Advent of EC2 instances in 2006 changed dramatically the server renting time framework into one hour. This new model brought a lot of advantages for both service provider as well as clients via better time-sharing of the hardware, shutting down unneeded instances, reducing total cost of ownership or executing a particular application, and having a greener computing system. This new trend of renting server for one hour time frames has been recognized as such a huge successful business schema that forecaster researches predict a two hundred billion dollar market for cloud computing by 2020 [260].

As the short rental durations and diversity in available instances types offered by today's cloud providers, clients need to automate the process of deploying their applications into cloud for reaching an economic yet effective solution. To reach a satisfactory solution from client's point of view, however, client utility function is also required to be exposed to the scheduling agent. But, this exposure is not something that users are willing to do. They may not have any information about

their preference between time and cost, or they may afraid that exposure of their utility function can have a negative impact on their chance for particular type of bargains in future. Secondly, in most practical situations users do not have advanced accurate information about the required execution time of submitted tasks, which is assumed to be known in many existing scheduling solutions, make their ability thwarted.

In this thesis, we have presented several algorithms to execute large-scale BoT scheduling in hybrid cloud environment cost-effectively. Our solutions try to satisfy conflicting users' objectives (makespan and total cost) with or without prior knowledge about the processing time of tasks. Precisely, in Chapter 4 we discussed how an optimum solution can be achieved if user's utility function follows a specific form of Lebesgue p-Norm. Our solution works perfectly for both cases of known and unknown tasks' running times. The methodology we used for the known running time case was to reformulate the scheduling problem as a binary nonlinear programming problem, and apply Lagrange multipliers method to find an optimal solution for the relaxed version of the original problem. The solution for the relaxed version actually shows the amount of workload for each cloud resource in the optimal case. For the non-clairvoyant case, we presented an FPRAS algorithm which consists of estimation and task assignment steps. The estimation procedure uses a well known *Monte Carlo* sampling method called $\mathcal{AA}$ algorithm to produce an estimated total workload needed for executing the whole submitted BoT application. In the task assignment phase, we take advantage of a fast yet efficient approach based on First Fit Decreasing (FFD) algorithm. The quality of proposed scheduling algorithms has been evaluated by running several experiments using the in-house cluster plus the Amazon EC2 instances.

Chapters 5 and 6 deals with the scheduling problem when users personal preferences might not be available for some reasons. We argued that users normally have two conflicting objectives of minimizing the total monetary cost and maximizing the performance. We presented some algorithms that approximates the Pareto-curve of both objectives in both clairvoyant and non-clairvoyant cases (regarding tasks' running time). The framework determines the achievable Pareto-points (based on the available or estimated information) and provide user with a list of these points to choose one regarding his/her preferences, budget, or deadline. By converting the original scheduling problem into an integer programing, our framework found out the amount of workloads must be assigned to each resources (public or private) in the optimal solution for the Pareto-point selected

by the user. Reaching effectively to the determined workload is also an NP-hard problem (resource allocation problem). We convert this challenge into a special case of well-known subset-sum problem and use a $FPTAS$ (or $FPRAS$) algorithm to reach an approximate solution (in non-clairvoyant case). Extensive experiments conducted on a testbed hybrid cloud infrastructure (a mixture of using Amazon EC2 and our in-house private cloud) confirm that the final scheduling output delivered by our framework achieves a solution very close to the user's chosen Pareto-optimal point, and outperforms enhanced List scheduling approach by reducing both makespan and total cost significantly.

7.2 Future Work

In addition to the research achievements summarized in previous chapters, the results of our study have opened windows for several interesting and important issues as future research work listed below.

One exciting future work for PANDA framework includes devising true mechanisms to cope with unreliability of underlying resources in both public and private cloud. Several studies have already showed that the performance of resources in cloud is not a fixed value during the life-time of an application (as what we suggested in this study). Distributing right mechanisms for handling resource performance variability in the dynamic scheduling layer or large errors in prediction of task execution time is another topic for research. It means that "system robustness" is the third objective that can be further investigated as as a future plan. Our work presented in 6 can nicely handle predication errors less than twice the actual value. So, an unanswered question is to devise dynamic solutions to find optimal strategy effectively for dealing with situations that the percentage of errors is tenfold (or even more) than the actual values. A good initial clue for conducting this research can be found in [261].

Another exciting direction for future work includes coping with some workload features missed in this study. Building scheduling frameworks that are designed by taking into account performance impacts of specific workload features such as temporal locality, burst behavior, workload periodicity (like daily cycles), or self-similar characteristics is still rare in the literature, especially for hybrid-cloud system. It can be imagined that renting resources in public cloud to handle burst, periodic, or long tail workloads is a promising solution. Applying ideas contributed in this thesis, such as finding a tradeoff solution, however, has not yet

been thoroughly studied. By adding more complex characteristics of application workload such as periodicity, temporal burst, or self-similar patterns, one can further explore the efficiency of PANDA, and adjust its scheduling algorithm to reach an optimized solution based on these workload features.

Integration of more complex data-intensive applications with PANDA framework can provide software solutions to large-scale data analysis applications (such as Map-Reduce framework) in hybrid cloud infrastructure. Currently, our proposed system copes with large-scale application which contains CPU-intensive independent tasks. However, data-intensive applications are becoming the dominant workload of cluster systems suddenly, referred to big-data challenge, too. In these new type of large-scale applications, a huge volume of data are distributed with fixed locations worldwide. In these cases, a new important scheduling goal would be to reduce the rate of data transfer as mush as possible. So, developing a more complex cost model to incorporate the data transfer cost into the current optimization goals and constraints to reach better data placement and resource allocation strategies can be further studied.

Bibliography

[1] G. R. Andrews, *Foundations of Parallel and Distributed Programming*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1st ed., 1999.

[2] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "A view of cloud computing," *Commun. ACM*, vol. 53, pp. 50–58, Apr. 2010.

[3] P. Mell and T. Grance, "The nist definition of cloud computing," Tech. Rep. 800-145, National Institute of Standards and Technology (NIST), Gaithersburg, MD, September 2011.

[4] R. Nathuji and K. Schwan, "Virtualpower: Coordinated power management in virtualized enterprise systems," in *Proceedings of Twenty-first ACM SIGOPS Symposium on Operating Systems Principles*, SOSP '07, (New York, NY, USA), pp. 265–278, ACM, 2007.

[5] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the art of virtualization," in *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*, SOSP '03, (New York, NY, USA), pp. 164–177, ACM, 2003.

[6] I. Hwang, T. Kam, and M. Pedram, "A study of the effectiveness of cpu consolidation in a virtualized multi-core server system," in *Proceedings of the 2012 ACM/IEEE International Symposium on Low Power Electronics and Design*, ISLPED '12, (New York, NY, USA), pp. 339–344, ACM, 2012.

[7] G. Chen, W. He, J. Liu, S. Nath, L. Rigas, L. Xiao, and F. Zhao, "Energy-aware server provisioning and load dispatching for connection-intensive internet services," in *Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation*, NSDI'08, (Berkeley, CA, USA), pp. 337–350, USENIX Association, 2008.

[8] S. Srikantaiah, A. Kansal, and F. Zhao, "Energy aware consolidation for cloud computing," in *Proceedings of the 2008 Conference on Power Aware Computing and Systems*, HotPower'08, (Berkeley, CA, USA), pp. 10–10, USENIX Association, 2008.

[9] F. Xu, F. Liu, H. Jin, and A. Vasilakos, "Managing performance overhead of virtual machines in cloud computing: A survey, state of the art, and future directions," *Proceedings of the IEEE*, vol. 102, pp. 11–31, Jan 2014.

[10] Y. Lee and A. Zomaya, "Energy efficient utilization of resources in cloud computing systems," *The Journal of Supercomputing*, vol. 60, no. 2, pp. 268–280, 2012.

[11] G. L. Valentini, S. U. Khan, and P. Bouvry, *Energy-Efficient Resource Utilization in Cloud Computing*, pp. 377–408. John Wiley & Sons, Inc., 2013.

[12] R. Nathuji and K. Schwan, "Virtualpower: Coordinated power management in virtualized enterprise systems," *SIGOPS Oper. Syst. Rev.*, vol. 41, pp. 265–278, Oct. 2007.

[13] B. Franklin, "Data center interconnect implementation guide for virtualized workload mobility with cisco, netapp and vmware," 2011.

[14] M. Ardica and J. Howering, "Data center interconnect design guide for virtualized workload mobility with cisco, emc and vmware," 2011.

[15] "Vmware infrastructure architecture overview," tech. rep., Palo Alto, CA, USA, 2006.

[16] G. Lovász, F. Niedermeier, and H. Meer, "Performance tradeoffs of energy-aware virtual machine consolidation," *Cluster Computing*, vol. 16, pp. 481–496, Sept. 2013.

[17] C.-H. Hsu, K. D. Slagter, S.-C. Chen, and Y.-C. Chung, "Optimizing energy consumption with task consolidation in clouds," *Inf. Sci.*, vol. 258, pp. 452–462, Feb. 2014.

[18] A. Sen, A. Garg, A. Verma, and T. Nayak, "Cloudbridge: On integrated hardware-software consolidation," *SIGMETRICS Perform. Eval. Rev.*, vol. 39, pp. 14–25, Sept. 2011.

[19] H. Goudarzi, M. Ghasemazar, and M. Pedram, "Sla-based optimization of power and migration cost in cloud computing," in *Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (Ccgrid 2012)*, CCGRID '12, (Washington, DC, USA), pp. 172–179, IEEE Computer Society, 2012.

[20] I. Rodero, J. Jaramillo, A. Quiroz, M. Parashar, and F. Guim, "Towards energy-aware autonomic provisioning for virtualized environments," in *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, HPDC '10, (New York, NY, USA), pp. 320–323, ACM, 2010.

[21] K. Tian and Y. Dong, "Power-aware i/o virtualization," in *Proceedings of the 2Nd Conference on I/O Virtualization*, WIOV'10, (Berkeley, CA, USA), pp. 8–8, USENIX Association, 2010.

[22] G. J. Popek and R. P. Goldberg, "Formal requirements for virtualizable third generation architectures," *Commun. ACM*, vol. 17, pp. 412–421, July 1974.

[23] A. Co., "Amazon elastic compute cloud (ec2)," 2013.

[24] I. Amazon Web Services, "Aws documentation," 2014.

[25] S. Sorrell, "Cloud computing - enterprise markets, saas, paas & iaas, 2014-2018," 2014.

[26] H. Casanova and F. Berman, *Parameter Sweeps on the Grid with APST*, pp. 773–787. John Wiley and Sons, Ltd, 2003.

[27] D. Abramson, J. Giddy, and L. Kotler, "High performance parametric modeling with nimrod/g: killer application for the global grid?," in *Parallel and Distributed Processing Symposium, 2000. IPDPS 2000. Proceedings. 14th International*, pp. 520–528, 2000.

[28] W. Cirne, D. Paranhos, L. Costa, E. Santos-Neto, F. Brasileiro, J. Sauve, F. A. B. Silva, C. Barros, and C. Silveira, "Running bag-of-tasks applications on computational grids: the mygrid approach," in *Parallel Processing, 2003. Proceedings. 2003 International Conference on*, pp. 407–416, Oct 2003.

[29] C. Marco, C. Fabio, D. Alvise, G. Antonia, G. Alessio, GiacominiFrancesco, M. Alessandro, M. Elisabetta, M. Salvatore, and P. Luca, "The glite workload management system," *Journal of Physics: Conference Series*, vol. 219, no. 6, p. 062039, 2010.

[30] O. Ben-Yehuda, A. Schuster, A. Sharov, M. Silberstein, and A. Iosup, "Expert: Pareto-efficient task replication on grids and a cloud," in *Parallel Distributed Processing Symposium (IPDPS), 2012 IEEE 26th International*, pp. 167–178, May 2012.

[31] A. Iosup, M. Jan, O. Sonmez, and D. Epema, "The characteristics and performance of groups of jobs in grids," in *Euro-Par 2007 Parallel Processing* (A.-M. Kermarrec, L. Bouge, and T. Priol, eds.), vol. 4641 of *Lecture Notes in Computer Science*, pp. 382–393, Springer Berlin Heidelberg, 2007.

[32] A. Iosup and D. Epema, "Grid computing workloads," *IEEE Internet Computing*, vol. 15, pp. 19–26, mar 2011.

[33] T. N. Minh, L. Wolters, and D. Epema, "A realistic integrated model of parallel system workloads," in *Cluster, Cloud and Grid Computing (CCGrid), 2010 10th IEEE/ACM International Conference on*, pp. 464–473, May 2010.

[34] E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys, "Sequencing and scheduling: Algorithms and complexity," *Handbooks in operations research and management science*, vol. 4, pp. 445–522, 1993.

[35] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New York, NY, USA: W. H. Freeman & Co., 1979.

[36] M. R. Garey and D. S. Johnson, "Strong np-completeness results: Motivation, examples, and implications," *J. ACM*, vol. 25, pp. 499–508, jul 1978.

[37] G. Ausiello, A. Marchetti-Spaccamela, and M. Protasi, "Toward a unified approach for the classification of np-complete optimization problems," *Theoretical Computer Science*, vol. 12, no. 1, pp. 83 – 96, 1980.

[38] A. Paz and S. Moran, "Non deterministic polynomial optimization problems and their approximations," *Theoretical Computer Science*, vol. 15, no. 3, pp. 251 – 277, 1981.

[39] K. S. Metaxiotis, K. Ergazakis, and J. E. Psarras, "An elaborate analysis of production systems in industry: what a consultant should know," *Industrial Management amp; Data Systems*, vol. 101, no. 4, pp. 185–193, 2001.

[40] E. Horowitz and S. Sahni, "Exact and approximate algorithms for scheduling nonidentical processors," *J. ACM*, pp. 317–327, 1976.

[41] R. L. Graham, "Bounds for certain multiprocessing anomalies," *Bell System Technical Journal*, vol. 45, no. 9, pp. 1563–1581, 1966.

[42] J. L. Bruno and P. J. Downey, "Probabilistic bounds on the performance of list scheduling.," *SIAM J. Comput.*, vol. 15, pp. 409–417, may 1986.

[43] O. H. Ibarra and C. E. Kim, "Heuristic algorithms for scheduling independent tasks on nonidentical processors," *J. ACM*, vol. 24, pp. 280–289, apr 1977.

[44] R. L. Graham, "Bounds on multiprocessing timing anomalies," *SIAM Journal on Applied Mathematics*, vol. 17, pp. 416—429, 1969.

[45] D. K. Friesen, "Tighter bounds for the multifit processor scheduling algorithm," *SIAM J. Comput.*, vol. 13, no. 1, pp. 170–181, 1984.

[46] G. Dobson, "Scheduling independent tasks on uniform processors," *SIAM J. Comput.*, vol. 13, pp. 705–716, nov 1984.

[47] M. Yue, "On the exact upper bound for the multifit processor scheduling algorithm," *Ann. Oper. Res.*, vol. 24, pp. 233–259, oct 1990.

[48] B. Chen, "Parametric bounds for lpt scheduling on uniform processors," *Acta Mathematicae Applicatae Sinica*, vol. 7, no. 1, pp. 67–73, 1991.

[49] R. E. Burkard and Y. He, "A note on multifit scheduling for uniform machines," *Computing*, vol. 61, pp. 277–283, dec 1998.

[50] D. s. Hochbaum and D. B. Shmoys, "A polynomial approximation scheme for scheduling on uniform processors: Using the dual approximation approach," *SIAM J. Comput.*, vol. 17, pp. 539–551, jun 1988.

[51] C. Chekuri and M. Bender, "An efficient approximation algorithm for minimizing makespan on uniformly related machines," in *Integer Programming and Combinatorial Optimization* (R. Bixby, E. Boyd, and R. Rios-Mercado, eds.), vol. 1412 of *Lecture Notes in Computer Science*, pp. 383–393, Springer Berlin Heidelberg, 1998.

[52] J. K. Lenstra and R. A. H. G. Kan, "Complexity of scheduling under precedence constraints," *Operations Research*, vol. 26, no. 1, pp. 22–35, 1978.

[53] G. J. Woeginger, "A comment on scheduling on uniform machines under chain-type precedence constraints," *Oper. Res. Lett.*, vol. 26, pp. 107–109, apr 2000.

[54] L. Epstein and J. Sgall, "Approximation schemes for scheduling on uniformly related and identical parallel machines," in *Algorithms - ESA' 99* (J. Nesetril, ed.), vol. 1643 of *Lecture Notes in Computer Science*, pp. 151–162, Springer Berlin Heidelberg, 1999.

[55] R. A. Cody and E. G. Coffman, Jr., "Record allocation for minimizing expected retrieval costs on drum-like storage devices," *J. ACM*, vol. 23, pp. 103–115, jan 1976.

[56] A. K. Chandra and C. K. Wong, "Worst-case analysis of a placement algorithm related to storage allocation.," *SIAM J. Comput.*, vol. 4, no. 3, pp. 249–263, 1975.

[57] S. K. Sahni, "Algorithms for scheduling independent tasks," *J. ACM*, vol. 23, pp. 116–127, jan 1976.

[58] G. J. Woeginger, "When does a dynamic programming formulation guarantee the existence of a fully polynomial time approximation scheme (fptas)?," *INFORMS J. on Computing*, vol. 12, pp. 57–74, jan 2000.

[59] J. Lenstra, *Sequencing by enumerative methods.* Mathematical Centre tracts, Mathematisch Centrum, 1977.

[60] K. Baker, *Introduction to sequencing and scheduling.* Wiley, 1974.

[61] W. E. Smith, "Various optimizers for single-stage production," *Naval Research Logistics Quarterly*, vol. 3, pp. 59–66, 1956.

[62] J. Du and J. Y.-T. Leung, "Minimizing mean flow time with release time and deadline constraints," *J. Algorithms*, vol. 14, no. 1, pp. 45–68, 1993.

[63] J. M. Jaffe, "Efficient scheduling of tasks without full use of processor resources," *Theoretical Computer Science*, vol. 12, no. 1, pp. 1 – 17, 1980.

[64] F. A. Chudak and D. B. Shmoys, "Approximation algorithms for precedence-constrained scheduling problems on parallel machines that run at different speeds," in *Proceedings of the Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '97, (Philadelphia, PA, USA), pp. 581–590, Society for Industrial and Applied Mathematics, 1997.

[65] C. Koulamas and G. J. Kyparisis, "Makespan minimization on uniform parallel machines with release times," *European Journal of Operational Research*, vol. 157, pp. 262–266, August 2004.

[66] J. Du, J. Y.-I. Leung, and G. H. Young, "Minimizing mean flow time with release time constraint," *Theor. Comput. Sci.*, vol. 75, pp. 347–355, oct 1990.

[67] J. Y, L. J., and M. Pinedo, "Minimizing total completion time on parallel machines with deadline constraints," *SIAM J. Comput.*, vol. 32, pp. 1370–1388, may 2003.

[68] T. F. Gonzalez, J. Y. Leung, and M. Pinedo, "Minimizing total completion time on uniform machines with deadline constraints," *ACM Trans. Algorithms*, vol. 2, pp. 95–115, jan 2006.

[69] S. T. McCormick and M. L. Pinedo, "Scheduling n independent jobs on m uniform machines with both flowtime and makespan objectives: A parametric analysis.," *INFORMS Journal on Computing*, vol. 7, no. 1, pp. 63–77, 1995.

[70] M. I. Dessouky, R. L. Marcellus, and L. Zhang, "Scheduling identical jobs on uniform parallel machines with random processing times," *Computers and Industrial Engineering*, vol. 35, no. 1-2, pp. 109–112, 1998.

[71] K. si Lin, "Scheduling with parallel machines to minimize total job tardiness," *Engineering Costs and Production Economics*, vol. 5, no. 34, pp. 289 – 296, 1981.

[72] A. Guinet, "Scheduling independent jobs on uniform parallel machines to minimize tardiness criteria," *Journal of Intelligent Manufacturing*, vol. 6, no. 2, pp. 95–103, 1995.

[73] M. Azizoglu and O. Kirca, "Tardiness minimization on parallel machines," *International Journal of Production Economics*, vol. 55, no. 2, pp. 163 – 168, 1998.

[74] V. A. Armentano and M. F. de FranASa Filho, "Minimizing total tardiness in parallel machine scheduling with setup times: An adaptive memory-based grasp approach," *European Journal of Operational Research*, vol. 183, no. 1, pp. 100 – 114, 2007.

[75] V. Kayvanfar, G. Komaki, A. Aalaei, and M. Zandieh, "Minimizing total tardiness and earliness on unrelated parallel machines with controllable processing times," *Computers and Operations Research*, vol. 41, no. 0, pp. 31 – 43, 2014.

[76] S. Polyakovskiy and R. M'Hallah, "A multi-agent system for the weighted earliness tardiness parallel machine problem," *Computers and Operations Research*, vol. 44, no. 0, pp. 115 – 136, 2014.

[77] V. Lauff and F. Werner, "Scheduling with common due date, earliness and tardiness penalties for multimachine problems: A survey," *Mathematical and Computer Modelling*, vol. 40, no. 5-6, pp. 637 – 655, 2004.

[78] H. Hussain, S. U. R. Malik, A. Hameed, S. U. Khan, G. Bickler, N. Min-Allah, M. B. Qureshi, L. Zhang, W. Yongji, N. Ghani, J. Kolodziej, A. Y. Zomaya, C.-Z. Xu, P. Balaji, A. Vishnu, F. Pinel, J. E. Pecero, D. Kliazovich, P. Bouvry, H. Li, L. Wang, D. Chen, and A. Rayes, "Review: A survey on resource allocation in high performance distributed computing systems," *Parallel Computing*, vol. 39, no. 11, pp. 709 – 736, 2013.

[79] J. Leung, *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*. Chapman & Hall/CRC Computer and Information Science Series, Taylor & Francis, 2004.

[80] C. N. Potts and V. A. Strusevich, "Fifty years of scheduling: a survey of milestones," *Journal of the Operational Research Society*, vol. 60, no. S1, 2009.

[81] L. Becchetti, S. Leonardi, A. Marchetti-Spaccamela, and K. Pruhs, "Semi-clairvoyant scheduling," *Theoretical Computer Science*, vol. 324, no. 2-3, pp. 325 – 335, 2004.

[82] M. A. Bender, S. Muthukrishnan, and R. Rajaraman, "Improved algorithms for stretch scheduling," in *Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA'02, (Philadelphia, PA, USA), pp. 762–771, Society for Industrial and Applied Mathematics, 2002.

[83] V. Berten, J. Goossens, and E. Jeannot, "On the distribution of sequential jobs in random brokering for heterogeneous computational grids," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 17, pp. 113–124, Feb 2006.

[84] A. Mualem and D. Feitelson, "Utilization, predictability, workloads, and user runtime estimates in scheduling the ibm sp2 with backfilling," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 12, pp. 529–543, Jun 2001.

[85] IBM, "Parallel sysplex," 2011.

[86] Microsoft, "Microsoft cluster service," 2003.

[87] O. Solaris, "Oracle solaris cluster," 2007.

[88] IBM, "World community grid," Nov. 2004.

[89] IBM, "European grid infrastructure," 2010.

[90] V. Pande, "Stanford university foldinghome project," Oct. 2000.

[91] B. University Of California, "Setihome," 1999.

[92] L. S. C. (LSC), "Einsteinhome," 2005.

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

[94] K. K. Droegemeier, D. Gannon, D. Reed, B. Plale, J. Alameda, T. Baltzer, K. Brewster, R. Clark, B. Domenico, S. Graves, E. Joseph, D. Murray, R. Ramachandran, M. Ramamurthy, L. Ramakrishnan, J. A. Rushing, D. Weber, R. Wilhelmson, A. Wilson, M. Xue, and S. Yalda, "Service-oriented environments for dynamically interacting with mesoscale weather," *Computing in Science and Engg.*, vol. 7, pp. 12–29, Nov. 2005.

[95] D. Thain, T. Tannenbaum, and M. Livny, "Distributed computing in practice: The condor experience: Research articles," *Concurr. Comput. : Pract. Exper.*, vol. 17, pp. 323–356, Feb. 2005.

[96] A. Barak and O. La'adan, "The mosix multicomputer operating system for high performance cluster computing," *Future Gener. Comput. Syst.*, vol. 13, pp. 361–372, Mar. 1998.

[97] A. Barak and A. Shiloh, "The mosix cluster operating system for high-performance computing on linux clusters, multi-clusters, gpu clusters and clouds," tech. rep., East Lansing, Michigan, 2014.

[98] D. Hildebrand, "An architectural overview of qnx," in *Proceedings of the Workshop on Micro-kernels and Other Kernel Architectures*, (Berkeley, CA, USA), pp. 113–126, USENIX Association, 1992.

[99] P. Leroux, "Qnx delivers extremely reliable microkernel for massively scalable routing system," 2004.

[100] IBM, "Platform load sharing facility," 2013.

[101] M. Litzkow, M. Livny, and M. Mutka, "Condor-a hunter of idle workstations," in *Distributed Computing Systems, 1988., 8th International Conference on*, pp. 104–111, Jun 1988.

[102] U. of Wisconsin–Madison, "Computing with htcondor," 2014.

[103] S. McClure and R. Wheeler, "Mosix: How linux clusters solve real world problems," in *Proceedings of the Annual Conference on USENIX Annual Technical Conference*, ATEC '00, (Berkeley, CA, USA), pp. 32–32, USENIX Association, 2000.

[104] M. Okun and A. Barak, "Atomic writes for data integrity and consistency in shared storage devices for clusters," *Future Generation Computer Systems*, vol. 20, no. 4, pp. 539 – 547, 2004. Advanced services for Clusters and Internet computing.

[105] L. Amar, A. Barak, and A. Shiloh, "The mosix direct file system access method for supporting scalable cluster file systems," *Cluster Computing*, vol. 7, pp. 141–150, Apr. 2004.

[106] A. Barak, A. Shiloh, and L. Amar, "An organizational grid of federated mosix clusters," in *Proceedings of the Fifth IEEE International Symposium on Cluster Computing and the Grid - Volume 01*, CCGRID '05, (Washington, DC, USA), pp. 350–357, IEEE Computer Society, 2005.

[107] A. Barak and A. Shilo, "Mosix cluster operating system user's and administrator's guides," 2014.

[108] D. Hildebrand, "An architectural overview of qnx," in *Proceedings of the Workshop on Micro-kernels and Other Kernel Architectures*, (Berkeley, CA, USA), pp. 113–126, USENIX Association, 1992.

[109] L. F. Friedrich, J. Stankovic, M. Humphrey, M. Marley, and J. Haskins, "A survey of configurable, component-based operating systems for embedded applications," *IEEE Micro*, vol. 21, pp. 54–68, May 2001.

[110] F. Armand and M. Gien, "A practical look at micro-kernels and virtual machine monitors," in *Proceedings of the 6th IEEE Conference on Consumer Communications and Networking Conference*, CCNC'09, (Piscataway, NJ, USA), pp. 395–401, IEEE Press, 2009.

[111] A. Iqbal, N. Sadeque, and R. I. Mutia, *An Overview of Microkernel, Hypervisor and Microvisor Virtualization Approaches for Embedded Systems*. 2010.

[112] D. S. Milojíčić, F. Douglis, Y. Paindaveine, R. Wheeler, and S. Zhou, "Process migration," *ACM Comput. Surv.*, vol. 32, pp. 241–299, Sept. 2000.

[113] P. C. Inc, "Running jobs with platform lsf," 2009.

[114] IBM, "Platform lsf foundations," 2013.

[115] IBM, "Using platform multicluster," 2013.

[116] D. Quintero, *IBM Technical Computing Clouds*. Poughkeepsie, NY: IBM Corp., International Technical Support Organization, 2013.

[117] F. Pascual, K. Rzadca, and D. Trystram, "Cooperation in multi-organization scheduling," *CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE*, vol. 21, pp. 905–921, May 2009.

[118] M. G. Buscemi, U. Montanari, and S. Taneja, "A game-theoretic analysis of grid job scheduling," *J. Grid Comput.*, vol. 10, pp. 501–519, Sept. 2012.

[119] P.-F. Dutot, K. Jansen, C. Robenek, and D. Trystram, "A (2+ϵ)-approximation for scheduling parallel jobs in platforms," in *Proceedings of the 19th International Conference on Parallel Processing*, Euro-Par'13, (Berlin, Heidelberg), pp. 78–89, Springer-Verlag, 2013.

[120] J. Cohen, D. Cordeiro, D. Trystram, and F. Wagner, "Analysis of multi-organization scheduling algorithms," in *Proceedings of the 16th International Euro-Par Conference on Parallel Processing: Part II*, Euro-Par'10, (Berlin, Heidelberg), pp. 367–379, Springer-Verlag, 2010.

[121] A. Bar-Noy, A. Freund, and J. S. Naor, "On-line load balancing in a hierarchical server topology," 2001.

[122] U. Schwiegelshohn, A. Tchernykh, and R. Yahyapour, "Online scheduling in grids," in *IPDPS*, pp. 1–10, 2008.

[123] A. Tchernykh, U. Schwiegelshohn, R. Yahyapour, and N. Kuzjurin, "On-line hierarchical job scheduling on grids with admissible allocation," *Journal of Scheduling*, vol. 13, no. 5, pp. 545–552, 2010.

[124] J. M. Ramirez-Alcaraz, A. Tchernykh, R. Yahyapour, U. Schwiegelshohn, A. Quezada-Pina, J. L. Gonzalez-Garcia, and A. Hirales-Carbajal, "Job allocation strategies with user run time estimates for online scheduling in hierarchical grids," *J. Grid Comput.*, vol. 9, pp. 95–116, Mar 2011.

[125] U. Schwiegelshohn, "An owner-centric metric for the evaluation of online job schedules," in *Proceedings of the 4th Multidisciplinary International Scheduling Conference: Theory and Applications (MISTA 2009), 10-12 Aug 2009, Dublin, Ireland* (J. Blazewicz, M. Drozdowski, G. Kendall, and B. McCollum, eds.), pp. 557–569, 2009.

[126] F. Kon, R. H. Campbell, M. D. Mickunas, K. Nahrstedt, and F. J. Ballesteros, "2k: A distributed operating system for dynamic heterogeneous environments," in *Proceedings of the 9th IEEE International Symposium on High Performance Distributed Computing*, HPDC '00, (Washington, DC, USA), pp. 201–, IEEE Computer Society, 2000.

[127] F. Berman and R. Wolski, "The apples project: A status report," 1997.

[128] W. Hoschek, F. J. Jaén-Martínez, A. Samar, H. Stockinger, and K. Stockinger, "Data management in an international data grid project," in *Proceedings of the First IEEE/ACM International Workshop on Grid Computing*, GRID '00, (London, UK, UK), pp. 77–90, Springer-Verlag, 2000.

[129] I. Foster and C. Kesselman, "The globus project: A status report," in *Proceedings of the Seventh Heterogeneous Computing Workshop*, HCW '98, (Washington, DC, USA), pp. 4–, IEEE Computer Society, 1998.

[130] M. eScience and G. E. Laboratory, "The nimrod toolkit," 1997.

[131] N. H. Kapadia and J. A. B. Fortes, "Punch: An architecture for web-enabled wide-area network-computing," *Cluster Computing*, vol. 2, pp. 153–164, Apr. 1999.

[132] F. Berman, R. Wolski, H. Casanova, W. Cirne, H. Dail, M. Faerman, S. Figueira, J. Hayes, G. Obertelli, J. Schopf, G. Shao, S. Smallen, N. Spring, A. Su, and D. Zagorodnov, "Adaptive computing on the grid using apples," *IEEE Trans. Parallel Distrib. Syst.*, vol. 14, pp. 369–382, Apr. 2003.

[133] M. Livny, J. Basney, R. Raman, and T. Tannenbaum, "Deploying a high throughput computing," *Cluster", High Performance Cluster Computing, Rajkumar Buyya, Editor*, 1999.

[134] I. Foster, "Globus toolkit version 4: Software for service-oriented systems," in *Proceedings of the 2005 IFIP International Conference on Network and Parallel Computing*, NPC'05, (Berlin, Heidelberg), pp. 2–13, Springer-Verlag, 2005.

[135] I. Foster, C. Kesselman, and S. Tuecke, "The anatomy of the grid: Enabling scalable virtual organizations," *Int. J. High Perform. Comput. Appl.*, vol. 15, pp. 200–222, Aug. 2001.

[136] I. Foster, "Service-oriented science: Scaling escience impact," in *Proceedings of the 2006 IEEE/WIC/ACM International Conference on Web Intelligence*, WI '06, (Washington, DC, USA), pp. 9–10, IEEE Computer Society, 2006.

[137] D. Abramson, J. Giddy, and L. Kotler, "High performance parametric modeling with nimrod/g: killer application for the global grid?," in *Parallel and Distributed Processing Symposium, 2000. IPDPS 2000. Proceedings. 14th International*, pp. 520–528, 2000.

[138] D. Abramson, I. Foster, J. Giddy, G. Laboratories, A. Lewis, R. Sosic, R. Sutherst, and N. White, "The nimrod computational workbench: A case study in desktop metacomputing," 1997.

[139] N. H. Kapadia, R. J. Figueiredo, and J. A. B. Fortes, "Punch: Web portal for running tools," *IEEE Micro*, vol. 20, pp. 38–47, May 2000.

[140] M. Wieczorek, R. Prodan, and T. Fahringer, "Scheduling of scientific workflows in the askalon grid environment," *SIGMOD Rec.*, vol. 34, pp. 56–62, Sept. 2005.

[141] L. F. Bittencourt and E. R. M. Madeira, "A dynamic approach for scheduling dependent tasks on the xavantes grid middleware," in *Proceedings of the 4th International Workshop on Middleware for Grid Computing*, MCG '06, (New York, NY, USA), pp. 10–, ACM, 2006.

[142] H. Zhao and R. Sakellariou, "Scheduling multiple dags onto heterogeneous systems," in *Proceedings of the 20th International Conference on Parallel and Distributed Processing*, IPDPS'06, (Washington, DC, USA), pp. 159–159, IEEE Computer Society, 2006.

[143] A. Ramakrishnan, G. Singh, H. Zhao, E. Deelman, R. Sakellariou, K. Vahi, K. Blackburn, D. Meyers, and M. Samidi, "Scheduling data-intensive workflows onto storage-constrained distributed resources," in *Proceedings of the Seventh IEEE International Symposium on Cluster Computing and the Grid*, CCGRID '07, (Washington, DC, USA), pp. 401–409, IEEE Computer Society, 2007.

[144] T. Szepieniec and M. Bubak, "Investigation of the dag eligible jobs maximization algorithm in a grid," in *Proceedings of the 2008 9th IEEE/ACM International Conference on Grid Computing*, GRID '08, (Washington, DC, USA), pp. 340–345, IEEE Computer Society, 2008.

[145] G. Singh, M.-H. Su, K. Vahi, E. Deelman, B. Berriman, J. Good, D. S. Katz, and G. Mehta, "Workflow task clustering for best effort systems with pegasus," in *Proceedings of the 15th ACM Mardi Gras Conference: From Lightweight Mash-ups to Lambda Grids: Understanding the Spectrum of Distributed Computing Requirements, Applications, Tools, Infrastructures, Interoperability, and the Incremental Adoption of Key Capabilities*, MG '08, (New York, NY, USA), pp. 9:1–9:8, ACM, 2008.

[146] L. Masko, G. Mounie, D. Trystram, and M. Tudruj, "Program graph structuring for execution in dynamic smp clusters using moldable tasks," in *Proceedings of the International Symposium on Parallel Computing in Electrical Engineering*, PARELEC '06, (Washington, DC, USA), pp. 95–100, IEEE Computer Society, 2006.

[147] N. Fujimoto and K. Hagihara, "Near-optimal dynamic task scheduling of precedence constrained coarse-grained tasks onto a computational grid," in *Proceedings of the Second International Conference on Parallel and Distributed Computing*, ISPDC'03, (Washington, DC, USA), pp. 80–87, IEEE Computer Society, 2003.

[148] O. H. Ibarra and C. E. Kim, "Heuristic algorithms for scheduling independent tasks on nonidentical processors," *J. ACM*, vol. 24, pp. 280–289, Apr. 1977.

[149] H. Casanova, D. Zagorodnov, F. Berman, and A. Legrand, "Heuristics for scheduling parameter sweep applications in grid environments," in *Proceedings of the 9th Heterogeneous Computing Workshop*, HCW '00, (Washington, DC, USA), pp. 349–, IEEE Computer Society, 2000.

[150] E. Santos-Neto, W. Cirne, F. Brasileiro, and A. Lima, "Exploiting replication and data reuse to efficiently schedule data-intensive applications on grids," in *Proceedings of the 10th International Conference on Job Scheduling Strategies for Parallel Processing*, JSSPP'04, (Berlin, Heidelberg), pp. 210–232, Springer-Verlag, 2005.

[151] Y. C. Lee and A. Y. Zomaya, "Practical scheduling of bag-of-tasks applications on grids with dynamic resilience," *IEEE Trans. Comput.*, vol. 56, pp. 815–825, June 2007.

[152] C. Anglano and M. Canonico, "Scheduling algorithms for multiple bag-of-task applications on desktop grids: A knowledge-free approach," in *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on*, pp. 1–8, April 2008.

[153] D. P. D. Silva, W. Cirne, F. V. Brasileiro, and C. Grande, "Trading cycles for information: Using replication to schedule bag-of-tasks applications on computational grids," in *Applications on Computational Grids, in Proc of Euro-Par 2003*, pp. 169–180, 2003.

[154] A. Benoit, L. Marchal, J.-F. Pineau, Y. Robert, and F. Vivien, "Scheduling concurrent bag-of-tasks applications on heterogeneous platforms," *Computers, IEEE Transactions on*, vol. 59, pp. 202–217, Feb 2010.

[155] A. Benoit, L. Marchal, J.-F. Pineau, Y. Robert, and F. Vivien, "Offline and online master-worker scheduling of concurrent bags-of-tasks on heterogeneous platforms," in *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on*, pp. 1–8, April 2008.

[156] F. A. B. da Silva and H. Senger, "Improving scalability of bag-of-tasks applications running on master-slave platforms," *Parallel Comput.*, vol. 35, pp. 57–71, Feb. 2009.

[157] V. Kumar and V. N. Rao, "Parallel depth first search. part ii. analysis," *Int. J. Parallel Program.*, vol. 16, pp. 501–519, Dec. 1987.

[158] M. D. Hill, "What is scalability?," *SIGARCH Comput. Archit. News*, vol. 18, pp. 18–21, Dec. 1990.

[159] M. Maheswaran, S. Ali, H. J. Siegel, D. Hensgen, and R. F. Freund, "Dynamic matching and scheduling of a class of independent tasks onto heterogeneous computing systems," in *Proceedings of the Eighth Heterogeneous Computing Workshop*, HCW '99, (Washington, DC, USA), pp. 30–, IEEE Computer Society, 1999.

[160] H. Casanova, M. Gallet, and F. Vivien, "Non-clairvoyant scheduling of multiple bag-of-tasks applications," in *Proceedings of the 16th International Euro-Par Conference on Parallel Processing: Part I*, EuroPar'10, (Berlin, Heidelberg), pp. 168–179, Springer-Verlag, 2010.

[161] B. Hong and V. K. Prasanna, "Distributed adaptive task allocation in heterogeneous computing environments to maximize throughput.," in *IPDPS*, 2004.

[162] R. Bertin, S. Hunold, A. Legrand, and C. Touati, "Fair scheduling of bag-of-tasks applications using distributed lagrangian optimization," *J. Parallel Distrib. Comput.*, vol. 74, pp. 1914–1929, Jan. 2014.

[163] S. Bleikertz, "On amazon ec2's underlying architecture," 2010.

[164] I. Eucalyptus Systems, "Elastic utility computing architecture for linking your programs to useful systems," 2008.

[165] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov, "The eucalyptus open-source cloud-computing system," in *Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*, CCGRID '09, (Washington, DC, USA), pp. 124–131, IEEE Computer Society, 2009.

[166] A. S. Foundation, "Apache cloudstack," 2010.

[167] O. Community, "Opennebula opennebula simple cloud management, developed for the enterprise private and hybrid cloud," 2008.

[168] O. Community, "Opennebula key features," 2012.

[169] O. MANUALS, "Openstack architecture," 2013.

[170] J. Rhoton, "Discover openstack: Architectures, functions, and interactions," 2013.

[171] A.-M. Oprescu and T. Kielmann, "Bag-of-tasks scheduling under budget constraints," in *Proceedings of the 2010 IEEE Second International Conference on Cloud Computing Technology and Science*, CLOUDCOM '10, (Washington, DC, USA), pp. 351–359, IEEE Computer Society, 2010.

[172] K. Deng, J. Song, K. Ren, and A. Iosup, "Exploring portfolio scheduling for long-term execution of scientific workloads in iaas clouds," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, SC '13, (New York, NY, USA), pp. 55:1–55:12, ACM, 2013.

[173] A.-M. Oprescu, T. Kielmann, and H. Leahu, "Stochastic tail-phase optimization for bag-of-tasks execution in clouds," in *Proceedings of the 2012 IEEE/ACM Fifth International Conference on Utility and Cloud Computing*, UCC '12, (Washington, DC, USA), pp. 204–208, IEEE Computer Society, 2012.

[174] S. Delamare, G. Fedak, D. Kondo, and O. Lodygensky, "Spequlos: A qos service for hybrid and elastic computing infrastructures," *Cluster Computing*, vol. 17, pp. 79–100, Mar. 2014.

[175] S. Shen, K. Deng, A. Iosup, and D. Epema, "Scheduling jobs in the cloud using on-demand and reserved instances," in *Proceedings of the 19th International Conference on Parallel Processing*, Euro-Par'13, (Berlin, Heidelberg), pp. 242–254, Springer-Verlag, 2013.

[176] C. Barrico and C. Antunes, "Robustness analysis in multi-objective optimization using a degree of robustness concept," in *Evolutionary Computation, 2006. CEC 2006. IEEE Congress on*, pp. 1887–1892, 2006.

[177] N. Vydyanathan, U. V. Catalyurek, T. M. Kurc, P. Sadayappan, and J. H. Saltz, "Toward optimizing latency under throughput constraints for application workflows on clusters," in *Proceedings of the 13th International Euro-Par Conference on Parallel Processing*, Euro-Par 07, (Berlin, Heidelberg), pp. 173–183, Springer-Verlag, 2007.

[178] J. J. Dongarra, E. Saule, E. Jeannot, and Z. Shi, "Bi-objective scheduling algorithms for optimizing makespan and reliability on heterogeneous systems," in *In Proc. of the 19th Annual ACM Symposium on Parallel Algorithms and Architectures SPAA'07*, pp. 280–288, 2007.

[179] Y. C. Lee, R. Subrata, and A. Y. Zomaya, "On the performance of a dual-objective optimization model for workflow applications on grid platforms," *IEEE Trans. Parallel Distrib. Syst.*, vol. 20, pp. 1273–1284, sep 2009.

[180] T. N'Takpe and F. Suter, "Concurrent scheduling of parallel task graphs on multi-clusters using constrained resource allocations," in *Parallel Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on*, pp. 1–8, May 2009.

[181] E. Saule and D. Trystram, "Analyzing scheduling with transient failures," *Inf. Process. Lett.*, vol. 109, pp. 539–542, May 2009.

[182] A. Hirales-Carbajal, A. Tchernykh, R. Yahyapour, J. L. González-García, T. Röblitz, and J. M. Ramírez-Alcaraz, "Multiple workflow scheduling strategies with user run time estimates on a grid," *J. Grid Comput.*, vol. 10, pp. 325–346, June 2012.

[183] A. Benoit, Y. Robert, A. Rosenberg, and F. Vivien, "Static strategies for worksharing with unrecoverable interruptions," *Theory of Computing Systems*, vol. 53, no. 3, pp. 386–423, 2013.

[184] E. Angel, E. Bampis, and A. Kononov, "A fptas for approximating the unrelated parallel machines scheduling problem with costs.," in *ESA* (F. M. auf der Heide, ed.), vol. 2161 of *Lecture Notes in Computer Science*, pp. 194–205, Springer, 2001.

[185] K. Jansen and L. Porkolab, "Improved approximation schemes for scheduling unrelated parallel machines," in *Proceedings of the Thirty-first Annual ACM Symposium on Theory of Computing*, STOC '99, (New York, NY, USA), pp. 408–417, ACM, 1999.

[186] C. H. Papadimitriou and M. Yannakakis, "On the approximability of trade-offs and optimal access of web sources," in *Proceedings of the 41st Annual Symposium on Foundations of Computer Science*, FOCS '00, (Washington, DC, USA), pp. 86–, IEEE Computer Society, 2000.

[187] E. Jeannot, E. Saule, and D. Trystram, "Bi-objective approximation scheme for makespan and reliability optimization on uniform parallel machines," in *Euro-Par 2008, Parallel Processing* (E. Luque, T. Margalef, and D. Benitez, eds.), vol. 5168 of *Lecture Notes in Computer Science*, pp. 877–886, Springer Berlin Heidelberg, 2008.

[188] C. Boeres, I. M. Sardina, and L. M. A. Drummond, "An efficient weighted bi-objective scheduling algorithm for heterogeneous systems," *Parallel Comput.*, vol. 37, pp. 349–364, aug 2011.

[189] E. Jeannot, E. Saule, and D. Trystram, "Optimizing performance and reliability on heterogeneous parallel systems: Approximation algorithms and heuristics," *J. Parallel Distrib. Comput.*, vol. 72, pp. 268–280, feb 2012.

[190] A. Agnetis, P. B. Mirchandani, D. Pacciarelli, and A. Pacifici, "Scheduling problems with two competing agents," *Oper. Res.*, vol. 52, pp. 229–242, mar 2004.

[191] I. Assayad, A. Girault, and H. Kalla, "A bi-criteria scheduling heuristic for distributed embedded systems under reliability and real-time constraints," in *Proceedings of the 2004 International Conference on Dependable Systems and Networks*, DSN'04, (Washington, DC, USA), pp. 347–, IEEE Computer Society, 2004.

[192] H. Hoogeveen, "Multicriteria scheduling," *European Journal of Operational Research*, vol. 167, no. 3, pp. 592 – 623, 2005. Multicriteria Scheduling.

[193] V. TKindt and J.-C. Billaut, "Multicriteria scheduling problems," in *Multiple Criteria Optimization: State of the Art Annotated Bibliographic Surveys* (M. Ehrgott and X. Gandibleux, eds.), vol. 52 of *International Series in Operations Research and Management Science*, pp. 445–491, Springer US, 2002.

[194] D. E. Irwin, L. E. Grit, and J. S. Chase, "Balancing risk and reward in a market-based task service," in *Proceedings of the 13th IEEE International Symposium on High Performance Distributed Computing*, HPDC '04, (Washington, DC, USA), pp. 160–169, IEEE Computer Society, 2004.

[195] J. Broberg and R. Buyya, "A multi-commodity flow approach to maximising utility in linked market-based grids," in *Proceedings of the 5th International Workshop on Middleware for Grid Computing: Held at the ACM/IFIP/USENIX 8th International Middleware Conference*, MGC '07, (New York, NY, USA), pp. 5:1–5:6, ACM, 2007.

[196] Y. Chi, H. J. Moon, and H. Hacigümüş, "icbs: Incremental cost-based scheduling under piecewise linear slas," *Proc. VLDB Endow.*, vol. 4, pp. 563–574, June 2011.

[197] C. B. Lee and A. E. Snavely, "Precise and realistic utility functions for user-centric performance analysis of schedulers," in *Proceedings of the 16th International Symposium on High Performance Distributed Computing*, HPDC '07, (New York, NY, USA), pp. 107–116, ACM, 2007.

[198] P. deGrandis and G. Valetto, "Elicitation and utilization of application-level utility functions," in *Proceedings of the 6th International Conference on Autonomic Computing*, ICAC '09, (New York, NY, USA), pp. 107–116, ACM, 2009.

[199] B. Chun, P. Buonadonna, A. AuYoung, C. Ng, D. Parkes, J. Shneidman, A. Snoeren, and A. Vahdat, "Mirage: A microeconomic resource allocation system for sensornet testbeds," in *Embedded Networked Sensors, 2005. EmNetS-II. The Second IEEE Workshop on*, pp. 19–28, May 2005.

[200] K. Lai, L. Rasmusson, E. Adar, L. Zhang, and B. A. Huberman, "Tycoon: An implementation of a distributed, market-based resource allocation system," *Multiagent Grid Syst.*, vol. 1, pp. 169–182, Aug. 2005.

[201] C. B. Lee and A. E. Snavely, "Precise and realistic utility functions for user-centric performance analysis of schedulers," in *Proceedings of the 16th International Symposium on High Performance Distributed Computing*, HPDC '07, (New York, NY, USA), pp. 107–116, ACM, 2007.

[202] S. Kalyanasundaram, E. K. P. Chong, and N. B. Shroff, "Optimal resource allocation in multi-class networks with user-specified utility functions," *Comput. Netw.*, vol. 38, pp. 613–630, Apr. 2002.

[203] M. R. HoseinyFarahabady, Y. C. Lee, and A. Y. Zomaya, "Pareto-optimal cloud bursting," *IEEE Transactions on Parallel and Distributed Systems*, vol. 99, no. PrePrints, p. 1, 2013.

[204] F. S. de Boer, M. M. Bonsangue, S. Graf, and W. P. de Roever, eds., *Timing Analysis and Timing Predictability*, vol. 3657 of *Lecture Notes in Computer Science*, Springer, 2005.

[205] M. Iverson, F. Ozguner, and L. Potter, "Statistical prediction of task execution times through analytic benchmarking for scheduling in a heterogeneous environment," in *Heterogeneous Computing Workshop, 1999. (HCW '99) Proceedings. Eighth*, pp. 99–111, 1999.

[206] M. A. Iverson, F. Ozguner, and G. J. Follen, "Run-time statistical estimation of task execution times for heterogeneous distributed computing," in *Proceedings of the 5th IEEE International Symposium on High Performance Distributed Computing*, HPDC '96, (Washington, DC, USA), pp. 263–270, IEEE Computer Society, 1996.

[207] E. Mera, P. Lopez-Garia, G. Puebla, M. Carro, and M. Hermenegildo, "Combining static analysis and profiling for estimating execution times," in *Practical Aspects of Declarative Languages* (M. Hanus, ed.), vol. 4354 of *Lecture Notes in Computer Science*, pp. 140–154, Springer Berlin Heidelberg, 2007.

[208] E.-S. Hu, A. Wellings, and G. Bernat, "Deriving java virtual machine timing models for portable worst-case execution time analysis," in *On The Move to Meaningful Internet Systems 2003: OTM 2003 Workshops* (R. Meersman and Z. Tari, eds.), vol. 2889 of *Lecture Notes in Computer Science*, pp. 411–424, Springer Berlin Heidelberg, 2003.

[209] E. Mera, P. Lopez, M. Carro, and M. Hermenegildo, "Towards execution time estimation in abstract machine-based languages," in *Proceedings of the 10th International ACM SIGPLAN Conference on Principles and Practice of Declarative Programming*, PPDP '08, (New York, NY, USA), pp. 174–184, ACM, 2008.

[210] A. Srivastava and A. Eustace, "Atom: A system for building customized program analysis tools," in *Proceedings of the ACM SIGPLAN 1994 Conference on Programming Language Design and Implementation*, PLDI '94, (New York, NY, USA), pp. 196–205, ACM, 1994.

[211] R. F. Freund, "Optimal selection theory for superconcurrency," in *Proceedings of the 1989 ACM/IEEE Conference on Supercomputing*, Supercomputing '89, (New York, NY, USA), pp. 699–703, ACM, 1989.

[212] M. A. Iverson, F. Özgüner, and L. Potter, "Statistical prediction of task execution times through analytic benchmarking for scheduling in a heterogeneous environment," *IEEE Trans. Comput.*, vol. 48, pp. 1374–1379, Dec. 1999.

[213] M. Tan and H. J. Siegel, "A stochastic model for heterogeneous computing and its application in data relocation scheme development," *IEEE Trans. Parallel Distrib. Syst.*, vol. 9, pp. 1088–1101, Nov. 1998.

[214] D. Pease, A. Ghafoor, I. Ahmad, D. L. Andrews, K. Foudil-Bey, T. E. Karpinski, M. A. Mikki, and M. Zerrouki, "Paws: A performance evaluation tool for parallel computing systems," *Computer*, vol. 24, pp. 18–29, Jan. 1991.

[215] S. Fu and C.-Z. Xu, "Stochastic modeling and analysis of hybrid mobility in reconfigurable distributed virtual machines," *J. Parallel Distrib. Comput.*, vol. 66, pp. 1442–1454, Nov. 2006.

[216] C.-Z. Xu, L. Y. Wang, and N.-T. Fong, "Stochastic prediction of execution time for dynamic bulk synchronous computations," in *Proceedings of the 15th International Parallel &Amp; Distributed Processing Symposium*, IPDPS '01, (Washington, DC, USA), p. 44, IEEE Computer Society, 2001.

[217] P. Dagum, R. Karp, M. Luby, and S. Ross, "An optimal algorithm for monte carlo estimation," in *Foundations of Computer Science*, pp. 142 –149, 1995.

[218] R. Adams and J. Fournier, *Sobolev Spaces*. Pure and Applied Mathematics, Elsevier Science, 2003.

[219] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms, Third Edition*. The MIT Press, 3rd ed., 2009.

[220] C. Chekuri and S. Khanna, "A ptas for the multiple knapsack problem," in *Proceedings of the eleventh annual ACM-SIAM symposium on Discrete algorithms*, SODA '00, (Philadelphia, PA, USA), pp. 213–222, Society for Industrial and Applied Mathematics, 2000.

[221] H. Kellerer, R. Mansini, U. Pferschy, and M. G. Speranza, "An efficient fully polynomial approximation scheme for the subset-sum problem," vol. 66, pp. 349 – 370, 2003.

[222] E. L. Lawler, "Fast approximation algorithms for knapsack problems," *Mathematics of Operations Research*, vol. 4, no. 4, pp. 339–356, 1979.

[223] S. Arora, D. Karger, and M. Karpinski, "Polynomial time approximation schemes for dense instances of np-hard problems," pp. 284–293, 1995.

[224] G. Dosa, "The tight bound of first fit decreasing bin-packing algorithm is ffd(i)=(11/9)opt(i)+6/9," in *Combinatorics, Algorithms, Probabilistic and Experimental Methodologies* (B. Chen, M. Paterson, and G. Zhang, eds.), vol. 4614 of *Lecture Notes in Computer Science*, pp. 1–11, Springer Berlin Heidelberg, 2007.

[225] A. Iosup, O. Sonmez, S. Anoep, and D. Epema, "The performance of bags-of-tasks in large-scale distributed systems," in *Intl. Symp. on High Performance Distributed Computing, HPDC '08*, (USA), pp. 97–108, 2008.

[226] J. B. Tenenbaum, V. de Silva, and J. C. Langford, "A global geometric framework for nonlinear dimensionality reduction," *Science*, vol. 290, no. 5500, p. 2319, 2000.

[227] E. Horowitz and S. Sahni, "Computing partitions with applications to the knapsack problem," *J. ACM*, vol. 21, pp. 277–292, Apr. 1974.

[228] D. Tsafrir, Y. Etsion, and D. G. Feitelson, "Modeling user runtime estimates," in *Proceedings of the 11th International Conference on Job Scheduling Strategies for Parallel Processing*, JSSPP'05, (Berlin, Heidelberg), pp. 1–35, Springer-Verlag, 2005.

[229] Y. Zhang, H. Franke, J. E. Moreira, and A. Sivasubramaniam, "An integrated approach to parallel scheduling using gang-scheduling, backfilling, and migration," in *Revised Papers from the 7th International Workshop on Job Scheduling Strategies for Parallel Processing*, JSSPP '01, (London, UK, UK), pp. 133–158, Springer-Verlag, 2001.

[230] W. Cirne and F. Berman, "A comprehensive model of the supercomputer workload," in *Proceedings of the Workload Characterization, 2001. WWC-4. 2001 IEEE International Workshop*, WWC '01, (Washington, DC, USA), pp. 140–148, IEEE Computer Society, 2001.

[231] D. G. Feitelson, "Parallel Workloads Archive."

[232] J. Zilber, O. Amit, and D. Talby, "What is worth learning from parallel workloads?: A user and session based analysis," in *Proceedings of the 19th Annual International Conference on Supercomputing*, ICS '05, (New York, NY, USA), pp. 377–386, ACM, 2005.

[233] D. Talby, *User Modeling of Parallel Workloads*. PhD thesis, Hebrew University, Dec. 2006.

[234] D. Tsafrir, Y. Etsion, and D. Feitelson, "Backfilling using system-generated predictions rather than user runtime estimates," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 18, pp. 789–803, June 2007.

[235] M. Garnaat, "boto: Python interface to amazon web services," 2013.

[236] O. H. Ibarra and C. E. Kim, "Fast approximation algorithms for the knapsack and sum of subset problems," *J. ACM*, vol. 22, pp. 463–468, Oct. 1975.

[237] J. Hamilton, "Overall data center costs." http://perspectives.mvdirona.com/2010/09/18/OverallDataCenterCosts.aspx, 2010. [Online].

[238] D. P. Chandrakant and J. S. Amip, "Cost model for planning, development and operation of a data center," tech. rep., Hewlett-Packard Development Company, Laboratories Palo Alto, 07 2005.

[239] H. Wang, Q. Jing, R. Chen, B. He, Z. Qian, and L. Zhou, "Distributed systems meet economics: pricing in the cloud," in *HotCloud '10*, p. 6, USENIX Association, June 2010.

[240] I. Konstantinou, E. Floros, and N. Koziris, "Public vs private cloud usage costs: The stratuslab case," in *Proceedings of the 2Nd International Workshop on Cloud Computing Platforms*, CloudCP '12, (New York, NY, USA), pp. 3:1–3:6, ACM, 2012.

[241] N. M. Tran and L. Wolters, "Towards a profound analysis of bags-of-tasks in parallel systems and their performance impact," in *Proceedings of the 20th International Symposium on High Performance Distributed Computing*, HPDC '11, (New York, NY, USA), pp. 111–122, ACM, 2011.

[242] U. Dayal, M. Hsu, and R. Ladin, "A transactional model for long-running activities," in *Proc. of Intl. Conf. on Very Large Data Bases, VLDB '91*, (USA), pp. 113–122, 1991.

[243] J. Tang and S.-Y. Hwang, "Handling uncertainties in workflow applications," in *Proc. of Intl. Conf. on Information & Knowledge Management, CIKM '96*, (USA), pp. 269–276, 1996.

[244] M. Berlingerio, F. Pinelli, M. Nanni, and F. Giannotti, "Temporal mining for interactive workflow data analysis," in *Proc. of Intl. Conf. on Knowledge discovery & Data Mining, KDD '09*, (France), pp. 109–118, 2009.

[245] R. B. V. Subramanyam, A. Goswami, and B. Prasad, "Mining fuzzy temporal patterns from process instances with weighted temporal graphs," *Int. J. Data Anal. Tech. Strateg.*, vol. 1, no. 1, pp. 60–77, 2008.

[246] W. Alsabhan, "Normalised support: a virtual angle of measurement of interestingness," *Int. J. Data Anal. Tech. Strateg.*, vol. 4, no. 1, pp. 101–114, 2012.

[247] P. Papapetrou, G. Kollios, S. Sclaroff, and D. Gunopulos, "Mining frequent arrangements of temporal intervals," *Knowledge Information System*, vol. 21, no. 2, pp. 133–171, 2009.

[248] S. Abrishami, M. Naghibzadeh, and D. H. Epema, "Cost-driven scheduling of grid workflows using partial critical paths," *IEEE Trans. on Parallel and Distributed Systems*, vol. 23, no. 8, pp. 1400–1414, 2012.

[249] Y. C. Lee, R. Subrata, and A. Y. Zomaya, "On the performance of a dual-objective optimization model for workflow applications on grid platforms," *IEEE Trans. on Parallel and Distributed Systems*, vol. 20, no. 9, pp. 1273–1284, 2009.

[250] A. Iosup, M. Jan, O. O. Sonmez, and D. Epema, "The characteristics and performance of groups of jobs in grids," in *In Euro-Par, volume 4641 of LNCS*, pp. 382–393, 2007.

[251] T. N. Minh, L. Wolters, and D. H. J. Epema, "A realistic integrated model of parallel system workloads," in *Proc. of Intl. Symp. on Cluster, Cloud the Grid, CCGRID '10*, (USA), pp. 464–473, 2010.

[252] C. Z. Xu, L. Y. Wang, and N.-T. Fong, "Stochastic prediction of execution time for dynamic bulk synchronous computations," *J. Supercomput.*, vol. 21, no. 1, pp. 91–103, 2002.

[253] A. Srivastava and A. Eustace, "Atom: a system for building customized program analysis tools," *SIGPLAN Not.*, vol. 29, no. 6, pp. 196–205, 1994.

[254] C.-K. Luk, R. Cohn, R. Muth, H. Patil, A. Klauser, G. Lowney, S. Wallace, V. J. Reddi, and K. Hazelwood, "Pin: building customized program analysis tools with dynamic instrumentation," in *Proc. of Intl. Conf. on Programming Language Design and Implementation, PLDI '05*, (USA), pp. 190–200, 2005.

[255] N. Nethercote and J. Seward, "Valgrind: a framework for heavyweight dynamic binary instrumentation," *SIGPLAN Not.*, vol. 42, no. 6, pp. 89–100, 2007.

[256] E. Horowitz and S. Sahni, "Computing partitions with applications to the knapsack problem," vol. 21, (New York, NY, USA), pp. 277–292, apr 1974.

[257] G. Dosa, "The tight bound of first fit decreasing bin-packing algorithm is ffd(i)=(11/9)opt(i)+6/9," in *ESCAPE, volume 4614 of LNCS*, pp. 1–11, 2007.

[258] A. Iosup, O. Sonmez, S. Anoep, and D. Epema, "The performance of bags-of-tasks in large-scale distributed systems," in *Proc. of Intl. Symp. on High Performance Distributed Computing, HPDC '08*, (USA), pp. 97–108, 2008.

[259] N. M. Tran and L. Wolters, "Towards a profound analysis of bbots in parallel systems & their performance impact," in *Proc. of Intl. Symp. on High Performance Distributed Computing, HPDC '11*, (USA), pp. 111–122, 2011.

[260] S. Ried, H. Kisker, P. Matzke, A. Bartels, and M. Lisserman, "Sizing the cloud—understanding and quantifying the future of cloud computing," tech. rep., USA, 2011.

[261] A. Iosup, N. Yigitbasi, and D. Epema, "On the performance variability of production cloud services," in *Proceedings of the 2011 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, CCGRID '11, (Washington, DC, USA), pp. 104–113, IEEE Computer Society, 2011.