EXPLORATIONS IN GRID WORKFLOW SCHEDULING

A thesis submitted to the University of Manchester
for the degree of Doctor of Philosophy
in the Faculty of Engineering and Physical Sciences

2010

By
Wei Zheng
School of Computer Science

Contents

List of Tables

List of Figures

Abstract

Aiming at aggregating numerous distributed resources to provide immense computing power, Grid computing has emerged as a promising paradigm to run complex composite applications such as workflows. However, the inherent uncertainties of grid systems as well as the structural complexity of workflow applications make it extremely challenging to schedule workflows in an efficient way, regardless of whether the objective is to minimize execution time or meet specific user and/or system Quality of Service (QoS) requirements. For both these cases, this thesis considers scheduling problems motivated by grid uncertainties and advances the state-of-the-art by developing new techniques to address these problems.

First, based on existing scheduling heuristics, a Monte-Carlo approach is developed to minimize the average makespan (i.e., the overall execution time) in the presence of task estimates exhibiting limited uncertainty in the form of (controlled) random behaviour. Next, a scenario where performance prediction is difficult to obtain and resource availability may vary over time, is considered. A low-cost efficient just-in-time heuristic is proposed to cope with grid uncertainties.

After addressing these performance-driven scheduling problems, a QoS-driven problem, which considers not only the aforementioned uncertainties but also the uncertainty caused by queue-based scheduling, is examined. In order to tackle all these uncertainties, an integrated scheduling model consisting of three supportive techniques is developed. Extensive evaluation using simulation shows that the proposed techniques can achieve substantial improvements towards the ultimate goal of providing a good solution for QoS-driven workflow scheduling on the Grid.

Declaration

No portion of the work referred to in this thesis has been submitted in support of an application for another degree or qualification of this or any other university or other institute of learning.

Copyright

Acknowledgements

This thesis has emerged with help and support from many people.

In the first place, I would like to thank my supervisor Dr. Rizos Sakellariou for his insightful guidance and continuous encouragement without that the completion of this dissertation would not have been possible. I am also grateful for his thoughtful advice and everything that I learnt from him.

I am deeply indebted to my parents, Mr. Jianshu Zheng and Mrs. Shuduan Liu, and my girlfriend, Yonghui, for their invaluable love, understanding and support all the time.

Thanks also to all members in the School of Computer Science, especially Dr. Milan D. Mihajlovic, Henan, Kwanchai, Serafeim, Syed, Viktor, and Ye, for their help during my PhD study.

This PhD project has been funded by a Dorothy Hodgkin Postgraduate Award. I feel highly appreciated for this prestigious scholarship from the UK government.

Chapter 1

Introduction

This chapter provides a brief introduction of the whole thesis. Section 1.1 illustrates scheduling in the context of parallel execution, using a simple example. Section 1.2 introduces Grid computing and workflow applications. Section 1.3 states the grid workflow scheduling problem. Section 1.4 describes the issues that motivate our work. Section 1.5 presents aims and contributions of the present research. Section 1.6 concludes the chapter with an overview of the structure of this dissertation.

1.1 Scheduling and Parallelism

According to the Cambridge Advanced Learner's Dictionary [Cam03], a *schedule* is defined as a 'timetable' or 'an official list of things' or 'a list of planned activities or things to be done showing the times or dates when they are intended to happen or be done'. In terms of the Oxford English Dictionary [SW89], *scheduling* is 'the action of entering in or drawing up a schedule'. Here, describing a scheduling procedure as 'drawing up a plan of activities or things to be done showing the times or dates when they are intended to happen or be done' comes the closest to the topic of this thesis.

Before defining scheduling more precisely and specifically, an intuitive perception is provided by introducing a simple example below, where scheduling technique is applied to a job. Note that a job here is regarded as a collection of multiple dependent or independent tasks.

Suppose there is a job to be done. The job consists of two tasks t_1 and t_2, which are independent to each other. Two workers, w_1 and w_2 are hired to do the

Required Time	w_1	w_2
t_1	3	2
t_2	4	6

Table 1.1: Execution time required by each worker to complete each task

job. One worker can only handle one task at a time, and the time each worker needs to complete each job is predictable. As enumerated in Table 1.1, the worker w_1 needs 3 and 4 time units, respectively, to complete t_1 and t_2, while w_2 needs 2 and 6. Apparently, different allocations of tasks to workers will lead to a different completion time for the job. Given that the job is ready to start at time unit 0 at the earliest, if one wants to complete the job as early as possible, he/she should plan the job by assigning t_1 to w_2 and t_2 to w_1, and letting both workers start at time 0. This results in a schedule which obtains the minimum job completion time of 4 ($= \max\{2, 4\}$).

Scheduling in the above example may be trivial. However, what this example can demonstrate is manifold and well connected with the topic of this thesis. From the example, three points can be extracted as follows.

First of all, it is illustrated that doing tasks in parallel (i.e., simultaneously by different workers) can save the time of completing the job. In the above example, if the tasks are both allocated to w_1 or w_2, correspondingly, the job completion time will be 7 (= 3+4) or 8 (= 2+6). This example can be easily associated with the principle of *parallel computing* in computer science, in which simultaneously running different parts of a computation on different CPUs can speedup the computation. Here, a job is viewed as a computation, and a worker a computational resource (e.g., CPU). For many years, parallel computing, as well as the development of the capability of CPU, has been playing an important role to satisfy the continuously growing computing demands from various areas such as science, engineering and commerce. Actually, the interest in parallel computing has grown even more quickly because of the physical constraints hindering the advance of CPU capability. Nowadays, the popularity of the Internet and the availability of high-speed networks make it possible for geographically-distributed and decentralizedly-administered resources to collaborate to solve a single large-scale problem. This encourages the emergence of a new parallel computing paradigm known as *Grid computing* [FK99, FK03]. Grid computing has been widely recognized as a promising technology to aggregate the

power of various geographically-distributed resources (e.g., personal computers, supercomputers, clusters, storage systems, data sources and specialized devices) owned by different organizations, to provide tremendous computational capability in order to solve problems that are too big to be run at any single resource. The work of this thesis is expected to contribute to the development of parallel computing, especially grid computing technology.

Secondly, the provided example also demonstrates that a carefully crafted schedule is crucial to the efficiency of doing tasks in parallel. An unwise schedule usually results in a degradation of performance. In the example, if t_1 and t_2 are allocated to w_1 and w_2, respectively, the job completion time will be 6 (=max$\{3, 6\}$), which is worse than the optimal result. Analogously, to explore the promising potential of Grid computing, effective and efficient scheduling solutions for grid systems are of fundamental importance.

Thirdly, the provided example depicts a generic formulation of a scheduling problem, which is to make 'a plan of activities or things to be done' (i.e., a schedule) for a given job in such a way that several constraints are respected and some given objective is achieved. In the example, a schedule includes the assignment of tasks and the setting of task start time. Some simple constraints are assumed, such as 'one worker can only handle one task at a time' and the objective is to 'minimize the job completion time'. Some of these specifications, such as the task assignment and the expected start time for each task, may also appear in the schedule for an application in the context of grids.

1.2 Grid Computing and Workflow Applications

Nowadays, with the popularity of high-speed Internet, Grid computing has emerged as a novel pattern of distributed computing. The main aim of Grid computing is to form a federation of computers which may consist of multiple networked resources within and/or across organizations to solve a grand problem that is too big for any single personal computer. The grid resources, usually connected by Ethernet, are often geographically distributed and from different administrative domains. Without central administration, these resources are normally coordinated by a set of open standards for pursuing common goals [Fos02]. The use of supported resource includes not only conventional computing resources like CPU, storage, and databases, but also various kinds of domain-specific services

and on-line instruments [Yu09]. In practice, a considerable number of grid platforms have been developed, such as EuroGrid [EG], Open Science Grid [OSG] and TeraGrid [TG]. According to the diversity of resources, these systems can often be categorized into Computational Grid, Data Grid, Knowledge Grid [FK99]. In this thesis, we concentrate on Computational Grid (Grid, hereafter).

With the advance of grid technologies, it has been increasingly popular for today's grid systems to provide diverse sophisticated services to support workflow applications (or, simply, workflows). Such applications have recently emerged as a paradigm for representing, managing and automating complex composite applications from diverse fields such as computational science, business and engineering [DGST09]. A workflow application is a set of tasks with dependencies between them that must be satisfied in order to achieve an overall goal. In reality, there have been quite a few examples of workflow applications running on grid platforms, such as e-Protein [EP] for biotechnology, EMAN [EM] for electron micrograph analysis, GriPhyN [GP] for experimental physics, LEAD [LE] for meteorological data analysis and weather forecasting, Montage [MO] for astronomy, and WIEN2K [WI] for quantum chemistry. All indications are that workflow applications are and will be an important use case for Grid computing.

To achieve efficient workflow execution in grid environments, a bunch of techniques are required to define, manage and execute complex workflows on the grid resources with the above-mentioned features. An integration of these techniques can be viewed as a workflow management system [YB05]. In recent years, as the popularity of workflow applications grows, many workflow management systems have been designed and developed, such as ASKALON [FJP+05], DAGMan/Condor [DA], GrADS [GR], GridFlow [CJSN03], Pegasus [PE] and Taverna [OAF+04]. In general, some of the issues addressed by a workflow management system involve how to: (i) define and represent the workflow tasks so that they are understandable and executable for grid resources; (ii) obtain resource information, including availability and capability of the grid resources; and (iii) determine the appropriate allocation of tasks to resources and the start time of executing the tasks. Although there may be diverse issues contributing to the complicated challenge for a workflow management system to achieve efficient workflow execution, at the core of this challenge is workflow scheduling.

1.3 The Grid Workflow Scheduling Problem

The workflow scheduling problem considered in this thesis is the process of finding a solution (which is often described as a scheduling algorithm) to determine the assignment of the workflow tasks to resources. This may also include the start time of these tasks on the resources so that several specified constraints are respected and one or more given objectives are met. Typically, the workflow scheduling problem involves a set of problem variables including the details of the workflow, the characteristics of the underlying system, the scheduling objective(s) etc. These variables are usually depicted by a set of parameters, which may specify the constraints on the scheduling process (e.g., dependencies between tasks) or information about the computing environment (e.g., the number of resources, the predicted execution time of each task on each resource).

The most common representation of a workflow is the directed graph, which can be classified into two categories: acyclic (DAG) and cyclic (DCG) [DGST09]. The former category is much more used than the latter. In this thesis, we concentrate on the workflows that can be represented by a DAG. DAG scheduling is recognized as an NP-hard problem [GJ79]. In mathematical terms, if a problem is NP-hard, then no polynomial time algorithm exists for its solution, unless P = NP [Uet01]. The latter condition is generally not considered to be likely. This leads to the issue that the algorithmic approach to solve such problems becomes a big challenge.

In addition, several inherent features of grid systems may render it more challenging to realize the efficient execution of a workflow. Firstly, the grid resource set is normally heterogeneous, i.e., different resources may have different hardware and/or software configuration and capability. This makes it not only important but also difficult to obtain a proper task-resource mapping for a workflow. Mainly due to the lack of centralized ownership and control and multiple users competing, grid resources are inherently dynamic and unpredictable. For instance, resources may exhibit varying availability and capability that change over time. Moreover, in spite of the development of prediction techniques, it is inevitable that the task execution time estimation is unlikely to be entirely accurate [JHSN05]. Thus, from the scheduler's point of view, the availability and capability of resource, and the performance prediction may appear to be uncertain. Such uncertainties, existing through the whole process of workflow scheduling, contribute to even greater complexity of the workflow scheduling problem.

In the context of grid computing, the users and the resource owners may have different goals. Therefore, different models may be used to capture these goals. In turn, these models will lead to different scheduling strategies and patterns. Two of the most commonly used models are *Performance-driven* and *QoS-driven*:

Performance-driven model is a traditional scheduling model that attempts to optimize performance metrics, such as the completion time of workflow or the system throughput. Performance-driven workflow management systems (e.g., DAGMan/Condor [DA], GrADS [GR], Taverna [OAF+04]) usually adopt conventional strategies, where a scheduling component decides which tasks are to be executed at which resource based on some cost functions driven by performance metrics, whereas the applications submitted by users are normally scheduled in best-effort manner. In such a scheduling manner, the scheduler attempts to optimize performance metrics whereas ignores users' various requirements on satisfaction and resource access cost (price).

QoS-driven model, in contrast, focuses on striking a tradeoff between the different and often conflicting requirements from users and service providers. Although the optimization of performance metrics might be significant for users, it is also envisaged that future fully deployed grid environments will need to guarantee a certain level of quality of service (QoS). Such QoS is usually based on some attributes that users find important, e.g., the deadline by which their jobs have to be completed. This model is often associated with the so-called market-based grids [BB07]. From the users' perspective, once their QoS requirements are satisfied, they would be happy to make the payment that is commensurate to the successful service provision and within their budget. From the service providers' perspective, they would like their owned resources, which provide services, to be sufficiently utilized so that the profit can be maximized. Like a contract in the real world, a Service Level Agreement (SLA), which is an agreement providing an explicit statement of the expectation and obligation of both sides—the user and the service providers—in their business relationship, plays a crucial role in market-based grids.

In this thesis, we consider grid workflow scheduling problems based on not only performance-driven model but also QoS-driven model. We focus on workflow completion time (i.e., makespan) as the performance metric to evaluate the

performance-driven model, and the satisfaction of QoS constraints on economic budget and time deadline in QoS-driven model. In a performance-driven model, two types of uncertainty may affect the optimization of makespan: one is the uncertainty in task performance prediction [JHSN05], and the other is the uncertainty caused by varying resource availability. In a QoS-driven model, these two uncertainties can also affect the guarantee to user's QoS requirements and the maximization of the service provider's profit. Moreover, with conventional queue-based scheduling systems, which run a task when it gets to the head of the queue, it is commonly reckoned inappropriate to provide guarantees to users' requirements, for example, when a hard deadline is specified. As a result, advance reservation [Mac03] is introduced into the scheduler so that a task can be made to run on a specific resource at a precise time. Or compensation may be given depending on how far the deadline is overshot [SCJ+05]. However, various issues may still arise from advance reservation due to the uncertainty in performance prediction [MSK+04]. Generally, the objective of this thesis is to develop efficient scheduling approaches to cope with the lack of predictability in grid environments so as to achieve different functional objectives with both performance-driven and QoS-driven models.

As a classic problem, DAG scheduling has been extensively studied in the context of heterogeneous systems. However, frequently, such computing environments are not viewed as dynamic as grids. A big challenge for grid workflow management systems is how to deal with the inherent uncertainties of grid environments in the scheduling and management for workflow execution so as to achieve high efficiency.

1.4 Motivation

Traditionally, the workflow scheduling problem that has been extensively studied considers the static case, where it is commonly assumed that the estimation of task execution times of a workflow is accurate. In this case, **full-ahead static scheduling** heuristics are trying to take into account the structure of the graph and the nature of the resources to build a good schedule. Two typical example approaches that follow this scheduling pattern are presented in [THW02] and [SZ04a]. However, accurate prediction is hardly possible in reality, especially for highly dynamic grid systems, where runtime changes may probably

degrade the expected performance of the static scheduling heuristics that rely upon performance estimation to make scheduling decision. Nevertheless, it can be argued that a well-crafted static schedule can still exhibit competitive running performance in the dynamic context with prediction uncertainty [JHSN05]. This motivates the exploration in this study of full-ahead scheduling as a scheme to cope with prediction inaccuracy so as to optimize application performance.

In the case of highly dynamic grid environments, where resource availability is not guaranteed and it is hard to obtain estimation of task execution time, a **just-in-time scheduling** scheme will be a better choice than a full-ahead that relies on stable resource availability and information about performance estimation. To avoid these uncertainties in early allocation, just-in-time scheduling approaches make a task allocation decision only when a workflow task is ready to execute, i.e., the data dependency restriction has been released. These approaches have been adopted by several workflow management systems such as DAGMan [TWML01] and Pegasus [DBG+04]. Among the efforts on developing just-in-time scheduling heuristics, the direction of seeking a deliberate execution order of tasks to acquire best possible parallelism of ready tasks seems to be promising [Ros04]. However, the existing heuristics on this direction are based on decomposition of DAG structure [MFR07], and thus have limitations of incomplete applicability and high overhead. Hence, it seems worth the effort to develop a new just-in-time scheduling heuristic, which has low cost and is useable for an arbitrary DAG structure.

The performance-driven model, in which user applications are executed in a best-effort manner, is naturally unsuitable to guarantee a certain level of user's QoS requirements. In the context of a QoS-driven model where performance estimation can usually be obtained, market-based grid systems may allow users to make advance reservations to secure resource availability and guarantee users' QoS requirements. However, due to estimation inaccuracy, problems may still arise for not only user's QoS guarantee but also service provider's benefits [MSK+04]. Although there has been increasing research interests on providing QoS in market-based grids [ABG02, ZBN+04, YB04, SY08, Qua06a, SCJ+05], few efforts so far have tried to deal with the uncertainty in performance estimation for workflow applications in a way that both meets user's hard QoS constraints and maximizes service provider's benefit. This fact also motivates a big portion of the work present in this thesis.

1.5 Aims and Contributions

The aim of this work is to investigate grid workflow scheduling techniques that can cope well with the inherent unpredictability of grid systems. For this purpose, the thesis considers both settings: (i) settings where the model followed is strictly performance-driven and (ii) settings where the model followed is QoS-driven.

The thesis starts with investigating how the existing full-ahead static scheduling heuristics behave with the stochastic deviation of task execution time in runtime, and then develops an approach aiming to optimize the average workflow makespan (i.e., the overall execution time of the whole workflow) with the prediction uncertainty in a performance-driven model. The second goal of this thesis is to design an efficient and effective just-in-time scheduling heuristic to deal with such a situation in a performance-driven model, where the information the full-ahead heuristics rely upon to make scheduling decisions (such as task execution time and/or resource availability in the future) is hard to obtain. Then, the thesis turns to the complex mission of tackling uncertainties for the QoS-driven workflow scheduling. The ultimate goal of this mission is to design an SLA-based scheduling model so as to guarantee users' QoS requirements and meanwhile maximize service providers' benefits. This challenge is broken into three sub-problems as follows: (i) how to efficiently find a plan about the allocation of tasks of the workflow to different resources to judge whether or not the user's QoS constraints can be met; (ii) how to make an advance reservation for each individual task of the workflow to strike a balance between satisfying multiple users' QoS requirements and the benefit of service providers; (iii) how to schedule the reserved tasks with a flexible start time in local resources to maximize the resource utilization and service providers' profits.

By realizing these goals, the present thesis makes the following contributions:

- A novel Monte Carlo-based approach that extends static full-ahead workflow scheduling heuristics to stochastic scheduling has been proposed to improve the average makespan over various cases where task execution time changes stochastically.

- A novel priority-based just-in-time workflow scheduling heuristic has been proposed, which maximizes the parallelism of ready tasks of workflow during runtime in order to improve the average makespan at a low cost.

- A novel SLA-based workflow scheduling model has been developed to both guarantee users' QoS constraints and maximize service providers' profit. More precisely, this SLA-based workflow scheduling model contributes on three aspects including planning, advance reservation and local scheduling as follows:

  - A novel workflow planning heuristic has been developed for the SLA-based scheduling model to determine efficiently whether or not the deadline-budget constraints specified by users can be satisfied in terms of the existing load of resources.
  - Novel advance reservation strategies have been developed for the SLA-based scheduling model to automate the advance reservation for workflow applications in order to both guarantee the user's QoS requirements under task execution time changes and maximize the overall profits earned by service providers.
  - A novel local scheduling policy has been developed for the SLA-based scheduling model to reduce the utilization fragments caused by advance reservation so as to maximize the resource utilization and service providers' profits.

1.6 Thesis Organization

To give a concise overview, this section summarizes the contents of the following individual chapters. The rest of this thesis consists of two main parts. Chapters 2 and 3 focus on the uncertainties of a performance-driven model and Chapters 4, 5, 6 and 7 contribute to addressing the uncertainties in a QoS-driven model. Each chapter is intended to be rather self-contained. Figure 1.1 illustrates how all of these chapters are organized and connected. The details of the following chapters are itemized as follows:

Chapter 2: A Monte Carlo-based approach for full-ahead scheduling heuristics. Since the majority of the existing workflow scheduling heuristics rely upon traditional queue-based (i.e., best-effort) scheduler [MSK+04], the thesis begins with an investigation aiming to make improvement on the average performance under such a scheduling mechanism. This chapter proposes a

novel Monte Carlo-based approach to minimize the average makespan in the cases where task execution times can be viewed as random variables. The simulated evaluation results suggest that the proposed approach provides significant improvement on the average makespan compared with that of static scheduling heuristics.

Chapter 3: A priority-based just-in-time scheduling heuristic. This work is motivated by a simple intuition, that is, when the tasks in a workflow are executed in such a sequence that maximize the ready tasks, the application should obtain considerable performance even if the remote resources change unpredictably over time. In this chapter, a priority-based heuristic is proposed to maximize the parallelism of ready tasks. The simulated evaluation suggests that the proposed heuristic not only outperforms comparative heuristics but also runs at a lower cost.

Chapter 4: Overview of SLA-based scheduling system. This chapter depicts the big picture of the proposed SLA-based scheduling system, reviews the state-of-art of the existing market-based scheduling and resource management systems and highlights the distinction of the present study.

Chapter 5: Budget-Deadline Constrained (BDC) workflow planning heuristics. When a user submitted a workflow request with specified budget and deadline constraints, the scheduling system must know in advance if it is feasible to meet the user's constraints in terms of current resource capacity to decide whether accepting the request or not. In this chapter, we propose and evaluate a new heuristic that can efficiently plan workflow applications with considering the user-specified budget and deadline constraints as well as the existing load of resources.

Chapter 6: SLA-based advance reservation. Based on the heuristic proposed in Chapter 5, this chapter investigates how to appropriately overestimate task execution time in advance reservation planning to guarantee user's QoS constraints under performance prediction uncertainty. To address the problem, several novel advance reservation strategies are designed and a simulator is implemented to evaluate the strategies' performance on both guaranteeing user's QoS constraints and maximizing service provider's benefit in the scenario of multiple workflows.

Chapter 7: SLA-based local scheduling. With overestimation of task execution time, the advance reservation strategies designed in Chapter 6 may probably result in low resource utilization when tasks are completed earlier than expected. This chapter presents local scheduling policies to make use of the flexibility of SLA-base reservations to improve the resource utilization. Here, 'flexibility' means that the individual reserved tasks in local resources can be adjusted freely as long as the deadline constraints and the task dependencies are satisfied. The proposed approach defines two important notions to specify the earliest possible start time and latest possible finish time for a reserved workflow task according to task dependencies, and employs a backfilling technique to adjust the start time of reserved tasks in local resources in order to improve the resource utilization and consequently increase service providers' revenue.

Chapter 8: Conclusion. This chapter reviews the thesis' contributions and discusses directions for future work.

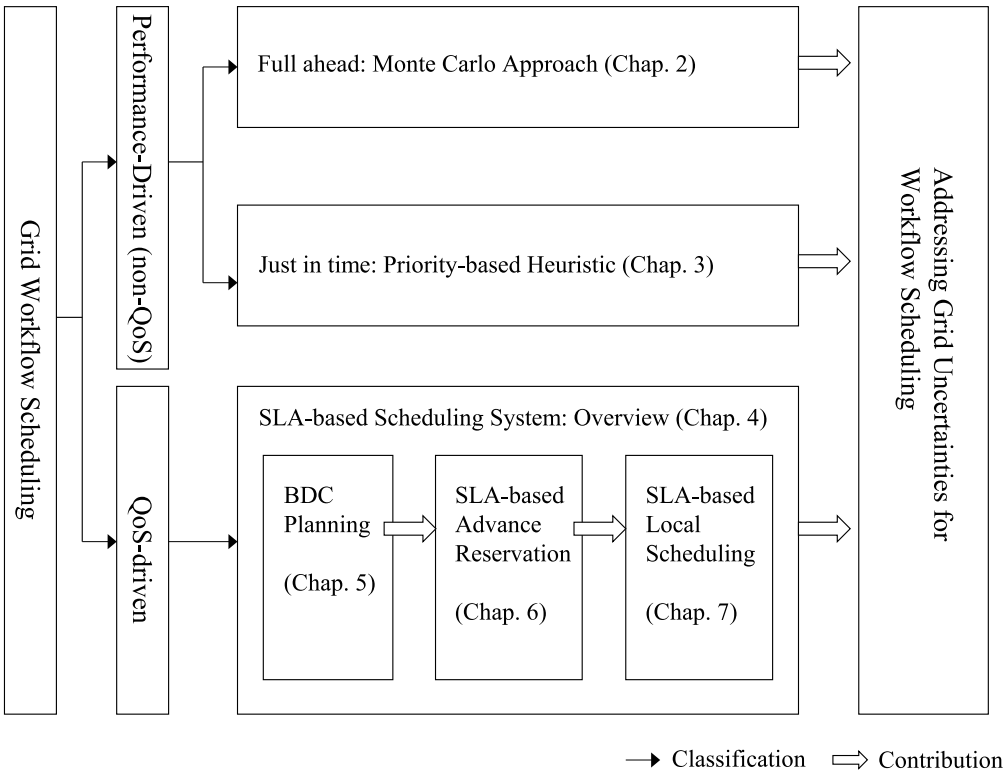


Figure 1.1: Thesis Structure

Chapter 2

A Monte-Carlo Approach for Full-ahead Scheduling

Static full-ahead scheduling schemes have been extensively studied in the context of heterogeneous computing systems and applied by several grid schedulers. However, in practice, the inaccuracy in performance prediction is always unavoidable, and to some extent, this may place an obstacle in the way of achieving performance-efficient schedules for workflow applications in grid environments. In addition, since heuristics do not provide globally optimal solutions, any performance prediction inaccuracies may affect the quality of their schedule even more.

This chapter reviews the existing static full-ahead scheduling heuristics, and extends the development of a full-ahead scheduling scheme admitting the uncertainty in performance prediction and treating each task execution time estimation as a random variable. The main contribution of this chapter is a novel full-ahead scheduling approach which exploits the existing static scheduling heuristics' talents to achieve minimum workflow completion time under performance prediction uncertainty. The scheme is implemented by combining two classic static scheduling heuristics respectively and evaluated by means of an extensive simulation at the end of this chapter. The experimental results demonstrate the great advantage the proposed approach has on minimizing workflow completion time with inaccurate task execution time estimation.

2.1 Background

Scheduling plays an important role in computer science when problems arise in the control of sequential or parallel processing on a computer, or a network of computers. An early example was scheduling in the context of multiprocessing operating system design for a single computer [Sim78, LKB77]. With the development of parallel computing, more and more research interests have been attracted to investigating how multiple tasks can be mapped across processors or machines to speedup the sequential performance of applications in parallel. During the late 1980s and early 1990s, scheduling studies mainly concentrated on multiple-processor and homogeneous environments [KA99]. Later on, as large scale distributed computing systems emerged, heterogeneous resources were encouraged to cooperate to resolve large problems. As a consequence, this motivated a great many further scheduling studies (for example, [THW99, BBR02a]) which take the heterogeneity of resources into account. Nowadays, with the popularity of grid computing paradigms, and the growing demand for workflow applications (especially those which can be presented by *directed acyclic graph* (DAG)), there has been a recent increase of efforts to research DAG scheduling in dynamic computing environments.

DAG scheduling problems have already been extensively explored in the context of heterogeneous systems, and it is well-known that the DAG scheduling problem is NP-complete for most of its variants [GJ79]. This means that the algorithmic approach to resolving such problems is a big challenge. As a result, many heuristics have been proposed, which behave well in practice but generally provide no guarantee of excluding the worst case of solution quality and computation cost. These heuristics may be highly diverse in terms of scheduling objectives and/or assumptions about the target computing environments. However, the majority of them are characterised as being deterministic full-ahead scheduling heuristics with the aim of minimizing the overall execution time of DAG application, i.e., *makespan*.

In a broad sense, the models of scheduling problems can be classified into two types: *deterministic* and *stochastic*; and the heuristic solutions of scheduling problem can be grouped into two categories: *full-ahead* and *just-in-time* [WHP08]. In the deterministic model, all problem input data (for example, the task execution times, communication costs) are assumed to be known with certainty in advance; while in stochastic model, some of the input data, which is the task

execution time in this thesis, may be subject to random fluctuation [SB99]. More precisely, it is assumed that the execution time of any task is governed by a corresponding random variable and its actual value becomes known only upon completion. Full-ahead heuristics, which deal with the problem totally before the execution of the whole application starts, are particularly suitable for a problem model where the input data is completely pre-known (for example, the deterministic model). In contrast, just-in-time heuristics make scheduling decision during runtime, namely each task is only scheduled at the time when the task is released from task dependency constraint to execution. The idea behind such an algorithmic paradigm is to mitigate the negative impact which arises in a situation in which there is hardly any effective information, either accurate or inaccurate, about the problem data. We consider only the full-ahead heuristics in this chapter and leave the just-in-time heuristics to the discussion in the next chapter.

Deterministic full-ahead scheduling heuristics, which have been extensively studied for heterogeneous systems, are not suitable in the context of grid computing. First of all, these heuristics are not directly applicable to such computing environments where the task execution times may be highly uncertain due to the following factors:

- The inherently non-dedicated grid resources are usually shared by multiple users. This means that the number of resources available for a single task may vary over time.
- It is still infeasible to accurately predict the performance of an arbitrarily-given task according to the state of art prediction techniques.
- Some tasks themselves may be inherently random (for example, simulated annealing, genetic algorithm).

Given these factors, it is argued that the stochastic model should be preferable to the deterministic model for DAG scheduling problems based-on grid resources. For instance, rather than giving the execution time of a task a constant valuation of 10 seconds, it can be assumed that the possible execution time varies from 5 to 15 seconds and a random variable can be used to depict this uncertainty. It should be noted that, in such a stochastic model, the deterministic heuristics may be applicable by using the mean values of these random variables. However, this approach, as shown in [SZ04b, KL05], does not lead to the best schedule in

most cases. And this will also be verified by the evaluation later in this chapter. It is worth mentioning that some deterministic schedules produced by full-ahead heuristics which obtain good makespan results in the deterministic model are somehow likely to exhibit a good average performance in the stochastic model. This is also indicated in [CJSZ08]. Therefore, it may be worth exploring the space of developing a novel approach based on the exhaustive deterministic full-ahead scheduling studies to resolve the scheduling problem with uncertain performance prediction in dynamic computing environments, such as grids.

Given this motivation, this research focuses on developing full-ahead DAG scheduling with the stochastic model. In this chapter, a Monte-Carlo based approach is proposed to minimize the average makespan of a given DAG application in the stochastic model. The remainder of the chapter is organised as follows. The stochastic scheduling problem and associated assumptions are introduced in Section 2.2. In Section 2.3, a taxonomy of the existing deterministic full-ahead scheduling heuristics is presented, and the research efforts on DAG scheduling aimed at minimizing makespan under performance prediction uncertainty are reviewed. The proposed Monte-Carlo based approach is presented in Section 2.4 and evaluated in Section 2.5. Finally, the chapter is concluded in Section 2.6.

2.2 Problem definition and assumptions

The specification of a scheduling problem may vary with the numerous characteristics of applications and underlying resources, and different performance metrics concerned. The contents of a scheduling problem which need to be specified usually include: (i) the modelling of the application (for example, independent tasks or dependent tasks represented by DAG); (ii) the concerned performance parameters and relevant definitions (for example, optimizing makespan may involve time parameters while improving network utilization may depend on bandwidth); (iii) the setting of the underlying system (for example, the number of available resources); (iv) a series of assumptions (for example, the scheduled tasks are non-preemptive) etc.

Although diverse circumstances in different underlying distributed systems may be considered in scheduling problems, the general pattern of resolving these problems seldom changes. Essentially, a scheduling solution consists of decisions about the allocation of tasks to resources (this may be trivial in homogeneous

systems) and the execution start time of the tasks assigned to each resource. These decisions are normally made based on an analysis of the problem input data, i.e., some available information about the tasks and resources, and produced by following the procedure defined by a certain scheduling algorithm.

In this section, all of the above-mentioned specifications related to the stochastic scheduling problem contents will be presented. It will begin by describing the model and definitions for the deterministic scheduling on heterogeneous resources, which is relatively simple and has been popularly studied. Then these concepts will be extended into the stochastic context. Finally, the associated assumptions will be listed at the end of the section. Most of the specifications presented in this section apply throughout this thesis. It should be noted that the focus is on time issues of application performance, and therefore, time parameters are of the main concern in this thesis.

2.2.1 Deterministic DAG Scheduling Model

Workflow applications can often be represented by a DAG, a generic model for applications consisting of a set of interdependent tasks. A DAG is normally denoted by $\mathcal{G} = (\mathcal{N}, \mathcal{E})$, where $\mathcal{N}$ is a set of nodes, and $\mathcal{E}$ is a set of directed edges between the nodes. A node $i \in \mathcal{N}$ is used to represent a task, which can be regarded as an indivisible computation unit like a program statement, an atomic sub-routine or even an entire program [GY92] (Therefore, the terms "node" and "task" are used interchangeably hereafter). An edge $e \in \mathcal{E}$ connecting node i and j is denoted by $i \rightarrow j$, which means that i is the parent node and j is the child, i.e., the input of j relies on the output of i. A path p is a sequence of edges among which each pair of neighbour edge share the adjacent endpoint. The path from i_0 to i_n is of the form $(i_0 \rightarrow i_1)$, $(i_1 \rightarrow i_2)$, $\cdots$ $(i_{n-2} \rightarrow i_{n-1})$, $(i_{n-1} \rightarrow i_n)$. It should be noted that there is no cycle path which satisfies $i_0 = i_n$ in a DAG. Edges indicate the precedence constraints of child nodes. A child node(task) can only begin execution when all of its parents are completed and all of the required input data of the task is available at the resource to which the child task is allocated. A node without any parent is called an entry node and a node without children is an exit node. Any node except the entry and exit should have one or more parents and children. Apparently, DAG $\mathcal{G}$ can equivalently be transformed with multiple entry nodes and/or exit nodes into $\mathcal{G}'$ with a single entry node and/or an exit node. For the sake of standardization, all of the DAGs in this thesis are

assumed to have only one entry node and one exit node. Each node and edge is assigned a weight in a DAG. In this thesis, the weight of a node i implies the computation volume of the task, and the weight of an edge e is the amount of data which needs to be transmitted from node i to j.

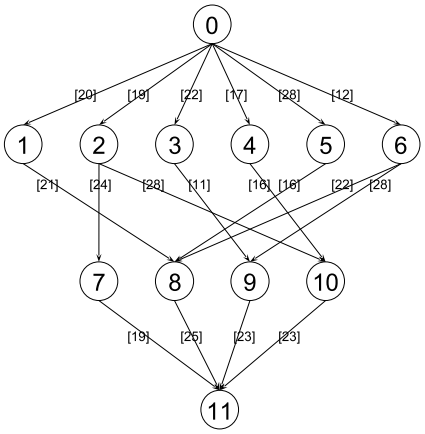
There is a set of heterogeneous resources $\mathcal{R}$. In order to determine the computation and communication time costs for tasks, besides the weights of nodes and edges, it is also necessary to consider the resources to which tasks are allocated and the transmission rate between these resources. In heterogeneous systems, the time required to execute a task to different resources may vary dramatically. Let $et_{i,p}$ denote the execution time of node i on resource p, and $tl_{(i,p)\rightarrow(j,q)}$ represent the latency of transmitting data from node i (allocated to resource p) to its child node j (allocated to resource q). This transmission latency is computed by

$$tl_{(i,p)\rightarrow(j,q)} = d_{i\rightarrow j} \times tr_{p,q} \tag{2.1}$$

where $d_{i\rightarrow j}$ is the weight of edge $i \rightarrow j$, and $tr_{p,q}$ means the transmission rate between resource p and q (i.e., the time taken to transmit a single unit of data between p and q). If both i and j are allocated onto the same resource, the transmission latency is assumed to be zero.

In the deterministic scheduling model, it is assumed that $et_{i,p}$ and $tl_{(i,p)\rightarrow(j,q)}$ are pre-known as constant values where $i, j \in \mathcal{N}$ and $p, q \in \mathcal{R}$. These values can normally be estimated by a performance model which predicts the performance of tasks, with or without historical data, on a set of specified machines. Although accurate performance prediction is crucial to the scheduling decision, the discussion about prediction techniques is beyond the scope and focus of this thesis.

To provide a concrete instance, a DAG application $\mathcal{G}$ is assumed, consisting of 12 nodes and running on 3 resources. As shown in Figure 2.1(a), node 0 is the entry node, node 11 is the exit node, and the number associated with each edge is the number of data units which need to be transferred between the two nodes, i.e., the edge weight. The estimation of the execution time of each node and the transmission rate between each pair of three resources are shown in Figure 2.1(b) and (c) respectively. In later Sections 2.3 and 2.4 this example will be used for illustration purposes.



(a) topology of DAG $\mathcal{G}$

Task	R_0	R_1	R_2	Task	R_0	R_1	R_2
0	9.0	48.0	19.0	6	9.0	19.0	23.0
1	45.0	60.0	14.0	7	8.0	12.0	5.0
2	11.0	16.0	9.0	8	25.0	84.0	67.0
3	14.0	37.0	29.0	9	3.0	8.0	1.0
4	8.0	14.0	17.0	10	14.0	19.0	18.0
5	40.0	75.0	66.0	11	53.0	86.0	20.0

(b) task execution time estimations of $\mathcal{G}$ on three heterogeneous resources

Connected Resources	Transmission time for per unit of data
R_0 and R_1	1.34
R_0 and R_2	1.72
R_1 and R_2	1.16

(c) transmission rates between resources

Figure 2.1: An Example of Deterministic Scheduling

2.2.2 Definitions of Deterministic DAG Scheduling

The result of a deterministic DAG scheduling heuristic is called a *static schedule*, which is generated by executing the scheduling heuristic on the model as described in Section 2.2.1 to achieve a pre-defined performance objective. One schedule may be represented in various forms, while the typical information contained in the schedule may essentially include: (i) the allocation of tasks to resources; and (ii) the start time of each task on its assigned resource. It should be noted that the second part usually indicates the execution order of the tasks in the same resource. This section provides several important definitions for deterministic DAG scheduling. Table 2.1 summarizes a list of terms and notations used in deterministic DAG scheduling.

Given that node i is allocated to resource p, besides the start time $st_{i,p}$, there are also several time parameters to be considered with in a schedule such as *data available time* ($dat_{i,p}$), *resource available time* ($rat_{i,p}$), and *finish time* ($ft_{i,p}$). Data available time $dat_{i,p}$ denotes the time at which all of the data required by node i arrives at resource p, which indicates the earliest possible time when task i can start according to the task dependency constraint. Thus we have:

$$dat_{i,p} = \max_{k \in Pred(i)} \{ft_{k,r(k)} + tl_{(k,r(k))\rightarrow(i,p)}\} \tag{2.2}$$

where $Pred(i)$ denotes the set of all immediate predecessors (i.e., parent tasks) of task i, and $r(k)$ means the resource to which k has been assigned. Moreover, task

Notation	Definition
$\mathcal{G}$	DAG $\mathcal{G}$.
$\mathcal{N}$	The set of nodes of $\mathcal{G}$.
$\mathcal{E}$	The set of edges of $\mathcal{G}$.
i, j, k	Task (node) i, j, k.
$i \rightarrow j$	The edge linking the node i and j, where i is the parent of j.
p, q	Resource p, q.
$et_{i,p}$	The Execution Time of task i on resource p, which is a constant in deterministic model, but a random variable in stochastic model.
$st_{i,p}$	The start time of task i on resource p.
$ft_{i,p}$	The finish time of task i on resource p.
$tl_{(i,p)\rightarrow(j,q)}$	The transmission latency between task i and j, which are separately allocated to resource p and q.
$dat_{i,p}$	Data Available Time, the arrival time of all required data needed by task i on resource p.
$rat_{i,p}$	Resource Available Time, the earliest time when resource p can execute task i.
CCR	Communication to Computation Ratio.
CP	Critical Path.
makespan	The overall execution time of application.

Table 2.1: Terms and notations used in deterministic DAG scheduling

i is queued in resource p and has to wait until the preceding tasks are completed and the resource is freed. Therefore, the start time $st_{i,p}$ also depends upon when the assigned resource can become available. Given that $ft_{l^*,p}$ is the completion time of the last preceding task of i, then $rat_{i,p} = ft_{l^*,p}$, and the start time of task i on resource p is defined by

$$st_{i,p} = max\{rat_{i,p}, dat_{i,p}\} \tag{2.3}$$

In addition, given that the estimated execution time of task i is $et_{i,p}$, the $ft_{i,p}$ can be computed by

$$ft_{i,p} = et_{i,p} + st_{i,p} \tag{2.4}$$

especially we have $ft_{entry\_node,r(entry\_node)} = et_{entry\_node,r(entry\_node)}$ given that the entry node starts at time 0, and the makespan of workflow equals to $ft_{exit\_node,r(exit\_node)}$.

For each DAG application, the ratio between the computation and communication costs is defined as being the *Communication-to-Computation Ratio*

(CCR). A small (large) CCR usually indicates that the corresponding application is computation-intensive (communication-intensive). This application feature is of importance, since different designs of scheduling heuristic may apply to applications with different features.

Another important concept in a deterministic DAG scheduling problem is the *Critical Path* (CP). The definition of DAG differs in various contexts. Given a parallel application, the CP indicates a sequence of tasks which must be executed sequentially and which consume the maximum time duration. For the DAG itself, the CP represents the longest path among all paths from the entry node to the exit node. Here, the length of a path is defined by the sum of the weight of nodes and edges constituting the path. For the DAG scheduling in homogeneous systems, the weight of a node and an edge is identical with the execution time of the task and the data transmission time between resources respectively. In such a case, the length of CP can be regarded as being the lower bound of the overall execution time of the application. However, this does not apply to heterogeneous systems, where the weight of the node is normally defined by the mean value of task execution times over different resources. Even so, the CP still play an important role of indicating the priority of each node in the graph, and this is often considered when making scheduling decisions. The CP can also be represented in a schedule, as shown in [SZ04b], where the CP is defined as the path with the least slack time from the first task which begins the execution to the last task which finishes. It should be noted that the CP in a schedule may not be the same as the one in a graph.

2.2.3 Stochastic DAG Scheduling Model

As previously mentioned, when a DAG application is executed in a dynamic distributed computing system such as Grids, the actual execution time of each task may exhibit variance over time. This section considers the full-ahead DAG scheduling model in a stochastic manner.

The majority of the description of the model and definition of deterministic scheduling can be reused in the stochastic context, for instance the representation of DAG, the definitions of CCR and CP etc. Moreover, to concentrate on the impact of performance prediction uncertainty on computational resources, the network performance, i.e., the transmission rates between resources is considered to be constant. Therefore, the transmission latency between interdependent tasks

is assumed to be deterministic.

The main difference between the models of deterministic and stochastic scheduling is the estimation of the time of the task execution. In the latter case, the performance estimation of a task on a resource is considered to be a random variable with a mean and a standard deviation instead of a constant value. Given a task i on resource p, the execution time prediction is denoted by $ET_{i,p}$ with the mean of $\mu_{i,p}$ and the standard deviation of $\sigma_{i,p}$, wherein the mean value $\mu_{i,p}$ can actually reflect the heterogeneity of resources. Similarly, as a counterpart of $st_{i,p}$, $ft_{i,p}$, $dat_{i,p}$ and $rat_{i,p}$ respectively, there are random variables $ST_{i,p}$, $FT_{i,p}$, $DAT_{i,p}$ and $RAT_{i,p}$, to which the relevant equations Eq.(2.2), Eq.(2.3) and Eq.(2.4) still apply. As a result, when computed based on these variables, the makespan of the application is also randomized. Therefore, the scheduling objective here turns to be the minimization of the expected value of the makespan.

Table 2.2 summarized the distinct notations used for stochastic DAG scheduling.

Notation	Definition
$ET_{i,p}$	The Execution Time Estimation of task i on resource p, random variable.
$\mu_{i,p}$	The mean value of $ET_{i,p}$.
$\sigma_{i,p}$	The standard deviation of $ET_{i,p}$.
$ST_{i,p}$	The start time of task i on resource p, random variable.
$FT(i,p)$	The finish time of task i on resource p, random variable.
$DAT(i,p)$	Data Available Time, the arrival time of all required data needed by task i on resource p, random variable.
$RAT(i,p)$	Resource Available Time, the earliest time when resource p can execute task i, random variable.
makespan	The completion time of application, random variable

Table 2.2: Terms and notations used in stochastic DAG scheduling

2.2.4 Assumptions

There are numerous issues which may affect the performance of a DAG scheduling heuristic, for example, the structure of the application, the characteristics of the underlying system, and the topology of the network. When all of these issues are taken into account, resolving a specific scheduling problem may become

highly complex. In order to avoid the unnecessary complexity due to redundant considerations and focus on the specific scheduling problem itself, appropriate assumptions need to be made according to the scheduling objective in the design of scheduling heuristic. Some major assumptions commonly made by many of the existing scheduling heuristics [Zha06] and the work described in this chapter are summarized as follows:

- *One task at a time.* It is assumed that at any given moment, a maximum of one task is allowed to be executed on one resource.
- *Full task-resource compatibility.* It is assumed that any resource is capable of executing any task.
- *Precedence constraint.* It is assumed that a task j cannot start running before all of its parent task i has been completed.
- *Non-preemptive scheduling.* It is assumed that the execution of a task cannot be interrupted until its completion.
- *Zero communication at the same resource.* It is assumed that the communication cost between task j and its parent task i is zero if these two tasks are allocated to the same resource.
- *Zero setting-up cost for communication.* It is assumed that the data transmission from a parent task i to a child begins immediately after task i has been completed. The delay for setting up communication is negligible.
- *Full network connection.* It is assumed that all resources are fully-connected, namely that there is always a network link between any two resources to allow data transmission.
- *Deterministic communication.* It is assumed that the network linking all of the resources has unlimited transmission capacity and that the transmission between two resources is only considered to be carried out via their directed link.
- *Pre-known information.* It is assumed that all information required (for example, the number of resources, the prediction of task execution time (either deterministic or stochastic), the prediction of data transmission time), is retrievable before the scheduling starts.

2.3 Related Work

In terms of related work, the existing static DAG scheduling heuristics in Section 2.3.1 are firstly investigated, after which a review is provided of the scheduling research efforts, focusing on minimizing the makespan under the performance prediction inaccuracy in Section 2.3.2.

2.3.1 Investigated Static Heuristics

There is abundant literature involving deterministic full-ahead DAG scheduling heuristics. Due to the NP-hardness of general DAG scheduling problems [GJ79], heuristic algorithms have been popularly proposed to avoid an exhaustive search for an optimal solution. Focusing on the performance of computational distributed systems, these heuristics are mainly designed for minimizing the makespan of DAG application. This is normally achieved via a sequence of steps using some obtained parameters as described in Section 2.2.1 with the assumptions presented in Section 2.2.4 to make a scheduling decision in order to find optimal or near optimal schedules. A thorough study of the current deterministic DAG scheduling is not regarded as being trivial in stochastic scheduling research. In the first place, there have been considerable research efforts put into the development of deterministic scheduling heuristics, which are worth investigating further in terms of their average performance in a stochastic model. Moreover, despite their different models of predicting task execution time, deterministic scheduling and stochastic scheduling still have quite a few scheduling issues in common, which is to say that the latter can be regarded as being an extension of the former, but with a more dynamic setting. This implies the possibility of developing new effective stochastic scheduling solutions based on the deterministic ones.

The remainder of this section will firstly present a taxonomy of the existing static full-ahead scheduling heuristics, and will then selectively introduce two representative heuristics which have proved to be efficient, and which will be employed in the implementation of this study's proposed scheduling approach. It should be noted that the focus is on those heuristics designed with the consideration of resources heterogeneity, which distinguishes this taxonomy from that one presented in [KA99] which focuses on scheduling heuristics for homogeneous systems.

Taxonomy

Various thoughts applied to the design of scheduling approaches endow different heuristics with their own design features, according to which, a taxonomy is provided as shown in Table 2.3. The provided taxonomy divides the existing static DAG scheduling heuristics into five categories: *list scheduling*, *workflow-based scheduling*, *clustering based scheduling*, *guided search scheduling*, and *task duplication based scheduling*, each of which is briefly described below.

- ***List Scheduling***. List scheduling [Cof76] is a widely studied scheduling method which makes an ordered list of tasks by assigning them some priorities and then schedules the tasks one by one in the descending order of priority. List scheduling heuristics may consist of three operations: (i) weighting: computing the weight of a task or an edge; (ii) listing: computing the priorities of tasks in a set according to the computed weights and sorting these tasks into an ordered list; and (iii) scheduling: picking up the task with the highest priority in the list, and giving it a scheduling decision including the resource to which it will be allocated and the execution order in the target resource. Operation (ii) and (iii) are repeated until all tasks are scheduled. According to the characteristics of task priority assignment patterns, they can be further grouped into *static priority*, *dynamic priority*, *critical path-based priority*, *level-based priority*, and *look-ahead priority* heuristics, as explained below:
  - *Static priority* means that the priorities of all tasks are computed before any task is scheduled, and these do not change during the scheduling procedure [KA99].
  - *Dynamic priority* refers to the task priority which may be computed in the scheduling procedure, and which consequently varies as the scheduling proceeds [KA96].
  - *Critical path-based priority* highlights the priorities of the CP nodes in the listing phase [THW02].
  - *Level-based priority* considers level, a kind of topological information of DAG, in making decision to prioritize [OH96].
- ***Clustering-based Scheduling***. Clustering-based scheduling heuristics [CJ01] merge the tasks of a DAG into clusters (i.e., clustering), and then map each

Existing Full-ahead Deterministic DAG Scheduling Heuristics	List Scheduling	Static Priority-based	HEFT	[THW02]
			HBMCT	[SZ04a]
			PCT	[MS98]
			k-DLA	[WY02]
			MSBC	[Che05]
			OLB	[AHK98]
			MET	[AHK98]
			MH(MCT)	[ERL90]
		Dynamic Priority-based	LDCP	[DK08]
			ETF	[BCT95]
			FLB	[RvG00]
			Min-min	[BJD+05]
			Max-min	[CLZB00]
			Duplex	[CLZB00]
			DPS	[ADUM97]
			BIL	[OH96]
			GDL(DLS)	[SL93]
			ILHA	[BBR02a]
		Critical Path-based	MCP	[WG90]
			FCP	[RvG00]
			CPOP	[THW02]
			HCPT	[HJ03]
			DCP	[KA96]
		Level-based priority	LMT	[IOF95]
	Workflow-based		ILS	[LPX05]
			WBA	[BJD+05]
	Guided Search-based		GA	[WSRM97]
			SA	[CP96]
			GSA	[CFW98]
			Tabu	[BSB+01]
	Clustering-based		Triplet	[CJ01]
	Task Duplication-based		TDS	[RA00b]
			STDS	[RA00a]
			HNPD	[BD05]
			LDBS	[DO02]

Table 2.3: A taxonomy of the existing deterministic DAG scheduling heuristics on heterogeneous resources.

cluster onto an available resource (i.e., mapping). The whole process normally consists of four steps: (i) *clustering*, (ii) *refining*, (iii) *mapping* and (iv) *local scheduling*. In the clustering step, each task is initially separated into one cluster, and then they are merged with each other into several clusters to eliminate the communication overhead between the dependent tasks by allocating them to the same cluster, which will be mapped to one resource. Then, a refining step of merging clusters may be repeated until the number of clusters no longer exceeds the number of resources. Next, the mapping step assigns each cluster to a resource by employing some heuristics, e.g., list scheduling. Finally, in the local scheduling step, the mapped tasks are sorted for execution in each resource. Clustering-based scheduling has been popularly studied and applied for homogeneous systems. However, its performance may be limited in heterogeneous systems, since the heterogeneity of resources may hardly be sufficiently considered in clustering.

- ***Workflow-based Scheduling***. Scheduling which considers the whole workflow rather than a set of ready tasks in scheduling is called workflow-based [BJD+05]. Workflow-based scheduling heuristics normally generate an initial schedule in the beginning, and then refines the schedule by a series of modifications (for example, swapping tasks across resources, reallocating assignments) in order to improve the application performance. The idea of scheduling the whole workflow is to avoid the possibility of short-sighted local decisions in the ready task-based scheduling by measuring the performance of the whole workflow. However, this global consideration may result in high complexity and an unacceptable algorithm overhead.

- ***Guided Search Scheduling***. Guided search techniques, which have been widely used in studies of optimization problems, can also be applied to resolve deterministic DAG scheduling problems. Such methods are called guided search scheduling heuristics, and typically include Genetic Algorithm (GA), Simulated Annealing (SA) etc [Win92]. These heuristics normally start by generating an initial schedule by some simple algorithm (for example, random allocation), then apply random search techniques to produce a new schedule based on the current one, and evaluate the newly produced schedule to determine whether or not this schedule will be kept. The search

and evaluation is repeated until a desired schedule is found or a time limit is reached. Generally, such an exhaustive search approach results in a better performance than heuristic-based ones (for example, list scheduling), but requires much more storage space and scheduling time, and consequently, it may not be suitable for large scale applications.

- ***Task Duplication-based Scheduling***. As its name suggests, a task duplication-based scheduling heuristic schedules a DAG application by allocating some of its tasks redundantly to more than one resource [LCWZ03]. The main idea of doing this is that the communication overhead between two directly dependent tasks (i.e., a parent and a child) will be reduced if they are mapped to the same resource. Normally, these heuristics fulfill duplication after an initial schedule has been constructed by some simple algorithm. A typical duplication consists of two steps: (i) a computation step which computes some specific parameters (for example, the latest start time of child tasks) to decide whether or not the duplication is necessary, and (ii) a mapping step, which decides the proper resource to accommodate the duplication. The final schedule will be obtained after the completion of the duplication phase. Although task duplication-based heuristics may have an advantage of reducing communication cost by utilizing resource idle time to allocate dependent tasks onto the same resource, their complexity is considerably high. This trade-off implies that such heuristics are particularly suitable for those applications which emphasise communication issues, for example, data intensive applications.

Description of Selective Heuristics

HEFT (Heterogeneous Earliest Finish Time) [THW02] is a static priority-based list scheduling heuristic which aims to minimize the makespan of DAG applications on a bounded number of heterogeneous resources. As described in Figure 2.2, the heuristic applies bottom-level to prioritize tasks in the listing phase, and then, in turn, selects the resource which manages to minimize the estimated finish time of the given task in the scheduling phase. It is worth mentioning that, when estimating the finish time of a task on a resource, HEFT enables the task to be inserted into the existing task queue of the resource as long as task

dependencies permit. The bottom-level of a task i is defined below:

$$bLevel(i) = w_i + \max_{j \in Succ(i)} \{w_{i \to j} + bLevel(j)\} \tag{2.5}$$

where $Succ(i)$ means the set of all of the immediate successors (i.e., child tasks) of task i, and w_i and $w_{i \to j}$ are respectively weights of nodes and edges, computed by

$$w_i = (\sum_{p \in \mathcal{R}} et_{i,p})/(|\mathcal{R}|) \tag{2.6}$$

$$w_{i \to j} = (\sum_{p,q \in \mathcal{R}} tl_{(i,p) \to (j,q)})/(|\mathcal{R}| \cdot |\mathcal{R}|) \tag{2.7}$$

where $|\mathcal{R}|$ is the number of resources.

Input: A DAG application G.
Output: A schedule for G.

Compute the weights of nodes and edges.
Compute the bottom-level for each node.
Sort all tasks in the descending order of priority and put them into list L.
while there are unscheduled tasks in L
    Select task i with the highest priority from list L.
    Compute $ft_{i,m}$ on each resource p.
    Allocate i to the resource p' that gives the minimum $ft_{i,p'}$ considering possible insertion.
    Remove i from list L.
endwhile

Figure 2.2: The HEFT Heuristic

HEFT is one of the most popular of the scheduling heuristics which are evaluated and extended for different computing environments (for example, ASKALON [WPF05]) with various assumptions. For instance, M-HEFT (Mixed-parallel HEFT) was presented in [BBR02b], which considered extending the HEFT heuristic in a more realistic parallel model with a more sophisticated resource structure and configurations. In addition, a HEFT-based adaptive rescheduling algorithm (AHEFT) was proposed in [YS07] to cope with the uncertainties in dynamic computing systems. However, this thesis only focuses on the original version of HEFT. A sample schedule of the above presented HEFT heuristic is given in Figure 2.3, where the heuristic is applied to the DAG example given in Figure 2.1.

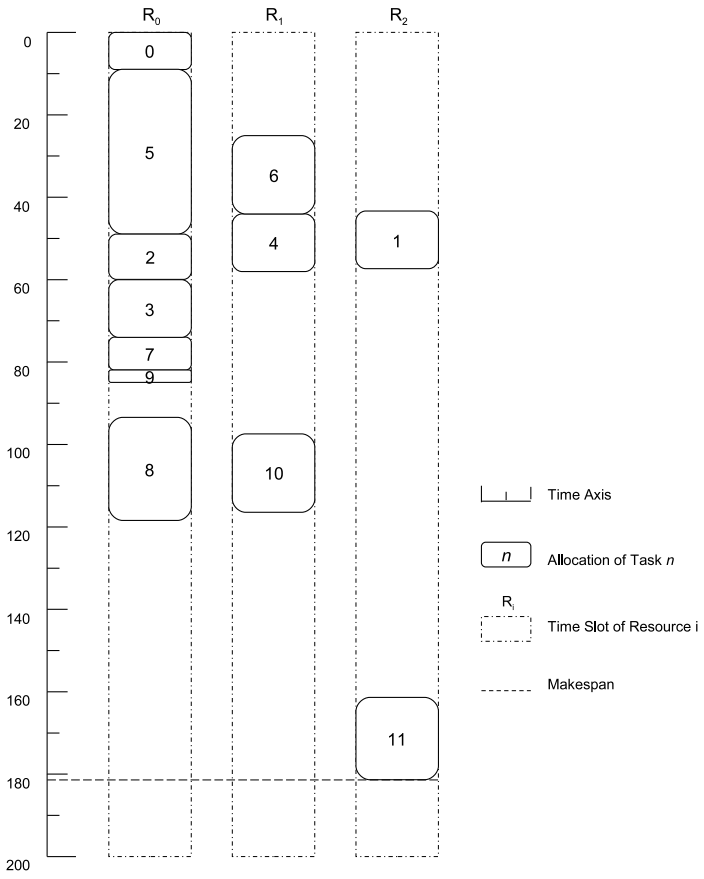


Figure 2.3: The example schedule generated by HEFT

HBMCT (Hybrid Balanced Minimum Completion Time) [SZ04a] is another effective static priority-based list scheduling heuristic, which simply uses bottom-level in the listing phase, but employs sophisticated optimizing techniques in the scheduling phase. The basic idea behind this heuristic is to divide all tasks into different groups comprising only independent tasks, and then apply BMCT, which is a scheduling heuristic for independent tasks, to minimize the maximum finish time of task in each group in order to optimize the makespan of the whole application. The heuristic detail is presented in Figure 2.4.

The HBMCT heuristic has not only demonstrated a comparable performance

Input: A DAG application G.
Output: A schedule for G.

Compute the weights of nodes and edges.
Compute the bottom-level for each node.
Sort all tasks in the descending order of priority.
Scan tasks in the sorted order and divide them into different groups according to their dependencies so that there are no dependent tasks in the same group.
for each group, in the descending order of the priorities of tasks in the group
    Schedule all tasks in this group by the BMCT heuristic presented in Figure 2.4(b).
endfor

(a) the outline of HBMCT

Input: A set of independent tasks S.
Output: A schedule for all tasks in S.

In the descending order of task priority, map each task to the resource that gives the minimum execution time.
for each resource
    Sort all assigned tasks in the ascending order of the earliest start time ST.
endfor
$Avg\_FT[] \leftarrow$ a vector of all tasks in ascending order of their average finish time across all resources.
repeat
    $m \leftarrow$ the resource giving the maximal finish time, MFT.
    Mark all tasks allocated to resource m in $Avg\_FT[]$ as unchecked and the rest checked.
    while there are unchecked tasks **and** *move_task* is false
        $t \leftarrow$ next unchecked task in $Avg\_FT[]$.
        for each resource n except m.
            Compute $ft_{t,n}$ with the assumption that t was inserted into the resource's task list.
        endfor
        $n \leftarrow$ the resource with the minimum $ft_{t,n}$.
        if $ft_{t,n} < MFT$ **then**
            Reallocate t from resource m to n.
            move_task $\leftarrow$ true.
        else
            Mark t as checked.
        endif
    endwhile
until *move_task* is false.

(b) The BMCT Heuristic

Figure 2.4: The description of HBMCT Heuristic

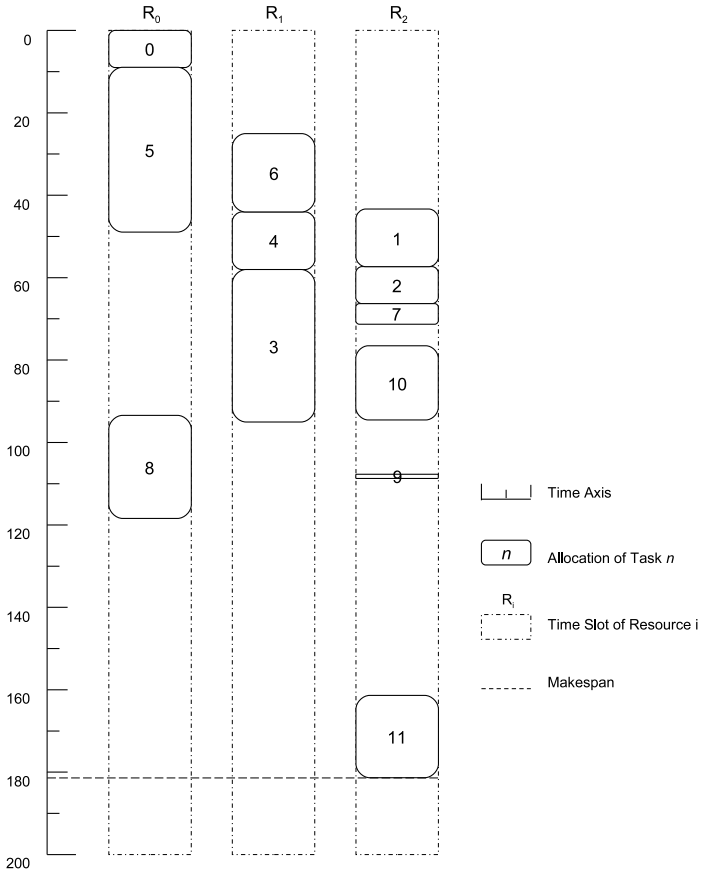


Figure 2.5: The example schedule generated by HBMCT

with HEFT in a variety of evaluations [SZ04a], but also exhibited active extensibility for more advanced scheduling problems, for example, resource reservation [ZS06a], multiple DAG scheduling [ZS06b] etc. HBMCT can also be considered to be adapted to stochastic scheduling problems, which will be presented in detail in Section 2.4. A sample schedule of HBMCT is given in Figure 2.5, as a result of applying the heuristic to the DAG example shown in Figure 2.1.

2.3.2 Other Works Related to Stochastic Scheduling

Given a DAG application in a dynamic distributed computing system (such as a grid), the deterministic schedule generated by a deterministic full-ahead scheduling heuristic is bound to exhibit a variable performance as a result of unpredictable situations occurring at runtime. In such a situation, it may be of interest to investigate the issues of (i) how the deterministic schedule behaves in a dynamic environment, which is reflected by the makespan and its variation of the schedule as the task execution time varies, and (ii) whether or not any robust full-ahead schedule exists which can guarantee the overall performance of the application at runtime, namely, which can achieve a reasonably good makespan in various cases of run-time changes. The concept of the robustness of a schedule, defined as being the average makespan of the schedule and the variation of the makespan, is proposed to measure the behaviour of the deterministic schedule in dynamic environments.

Motivated by the lack of studies attempting to investigate the deterministic scheduling heuristics in terms of their behaviour in dynamic environments, we contributed an implementation of heuristics to evaluate the robustness of the schedules they produce. Twenty of the heuristics listed in the previously provided taxonomy were selected for the evaluation, and some of the most widely used and cited were included in the selection while taking the algorithm overhead into account. These 20 heuristics, in alphabetical order, are: BIL, CPOP, DPS, Duplex, FCP, FLB, GDL, HBMCT, HCPT, HEFT, k-DLA, LMT, MaxMin, MCT, MET, MinMin, MSBC, OLB, PCT and WBA.

The evaluation and its results are published in [CJSZ08], where the stochastic model used to assess the robustness of deterministic schedules is depicted. In the stochastic model, task execution time is modelled as a normally distributed random variable. A deterministic schedule is generated by a deterministic heuristic using the mean as input. At each measurement, the execution time of a task on a resource is generated using the counterpart random variable and the makespan of the deterministic schedule is computed based on these generated values. It is worth mentioning that the initially computed task start and end times of the deterministic schedule may not be satisfied due to the variation in task execution time. In order to address this problem, two solutions are considered, one of which is called *sequence* strategy, which fully respects the order of the tasks specified by the deterministic schedule, i.e., task i is scheduled to be executed only after

the completion of all of the tasks which must be executed before task i, according to the deterministic schedule. The other solution is called *assignment* strategy, which respects the assignment of task to resources, but schedules a task for execution as soon as it becomes ready. These strategies are both evaluated in each measurement. Over a large number of measurements, the average makespan of each evaluated heuristic and the standard deviation of the makespan are obtained and analysed in the comparison between all heuristics.

The evaluation results in [CJSZ08] demonstrate that (i) it is better to respect the order of tasks in a deterministic schedule (i.e., use the sequence strategy) than to change this order in the run-time (i.e., use the assignment strategy); (ii) the robustness and deterministic makespan are somehow correlated: as it has been found elsewhere [SZ04b], the schedules which perform well in a deterministic case (i.e., when the mean of task execution time is used) tend to be the most robust; (iii) for the studied cases, heuristics including HEFT, HBMCT, GDL and PCT, are among the best for both average makespan and robustness.

There is a variety of workflow scheduling approaches to deal with the unpredictability of dynamic environments, and the first is to schedule every task only when it becomes ready (namely, when all of the tasks the current task depends on have been completed). Based on this idea, some heuristics [MFR07] have been developed to determine some particular order of task execution in order to improve application performance. Nevertheless, the evaluation in [SB08a] indicates that, without considering task execution costs in the scheduling of tasks, the practical effectiveness of these heuristics is limited in their current form. Another approach is rescheduling, which means producing an initial schedule for application based on a static prediction and then rescheduling the allocated tasks according to the variation during run-time. In many cases, rescheduling can indeed improve the application performance compared to a schedule which is generated based on a static prediction. However, this approach is costly, since it has to introduce extra scheduling efforts during run-time, which is a time-consuming solution even though some selective policy aimed at reducing the cost can be applied [SZ04b]. To tackle unpredictability, in addition to taking action during run-time as the aforementioned approaches do, it is also important to craft a static schedule with good properties (for example, minimizing the makespan) before the task execution starts (namely, full-ahead). A well-crafted static schedule without rescheduling can, not only save the extra scheduling efforts during run-time, but also obtain a

better application performance (to be demonstrated later) compared to a worse-crafted one with rescheduling. In addition, even if rescheduling is used, relevant studies [SZ04b] still indicate that a good initial schedule will help to reduce the extra cost of rescheduling [CJSZ08].

In the context of heterogeneous computing systems, there are a couple of scheduling studies, which consider the stochastic model of task execution times. In [DÖ01] and [DÖ04], stochastic scheduling approaches based on a Genetic Algorithm were proposed to minimize the scheduling length. With the same objective, in [KL07], in addition to the mean value of the random weights of DAG nodes, the standard deviations are considered to extend Min-Min and Max-Min algorithms in deriving a schedule. In [SJC+03], it is indicated that generic algorithms will cope well with deviations from the task execution time prediction. However, all of these approaches focus only on applications composed of independent tasks with no data dependencies. That is to say, DAG applications are not considered.

For DAG applications, Lòpez and Senar [LHS06] analyse the performance of several DAG scheduling heuristics and their counterpart dynamic version (with rescheduling) with the stochastic model. However, their study does not address the issue of how to produce an efficient schedule before run-time. In [KL05], static DAG scheduling heuristics including ETF and DLS were extended by estimating the earliest start times of tasks in a stochastic manner to minimize the scheduling length. This work introduced a computation and comparison of random variables to the process of computing the earliest start time for a task to make a mapping decision. However, a computation of random variables tends to be complicated and can hardly be possible for arbitrary random distributions. In contrast, the approach of this study avoids the complex manipulation of random variables by applying the Monte-Carlo approach to generate the resulting schedule.

2.4 A Monte Carlo-based Approach

The basic idea of the proposed approach is derived from the Monte-Carlo method [HH75], a popular solution for various problems which are infeasible or impossible to resolve by deterministic computation. The main characteristic of the Monte Carlo method is that it normally utilizes repeated random sampling to obtain numerous random samples of a stochastic problem, and computes its final results based on the obtained samples to resolve the problem. A common pattern of the Monte

Carlo method usually comprises the following steps:

1. defining a space comprising possible input values;
2. taking an independent sample randomly from the space;
3. performing a deterministic computation using the taken sample as input and keeping the result;
4. repeating Steps 2 and 3 until the pre-specified maximum repetition is reached;
5. aggregating the kept results of the individual computations into the final result.

The proposed Monte-Carlo based Scheduling approach (MCS hereafter) adapts the above-mentioned operations into a DAG scheduling scenario. Suppose that a DAG $\mathcal{G} = (\mathcal{N}, \mathcal{E})$ is being scheduled on a set of resources $\mathcal{R}$, the adapted operations can be interpreted as follows:

- *Defining Input Space*. In contrast to the generic Monte-Carlo method, the input space of MCS (denoted by $\mathcal{I}$) is a set consisting of the random task execution time predictions of the given application, namely

$$\mathcal{I}_{\mathcal{G}} = \{ET_{i,p} : i \in \mathcal{N}, p \in \mathcal{R}\} \tag{2.8}$$

- *Random Sampling*. In MCS, taking a sample from the space is defined as sampling each random prediction in the space and resulting in a set of static predictions. This can be expressed by function $f_{smp} : V_{|\mathcal{N}|} \mapsto C_{|\mathcal{N}|}$ as below.

$$\mathcal{P}_{\mathcal{G}} = f_{smp}(\mathcal{I}_{\mathcal{G}}) = \{t_{i,p} : i \in \mathcal{N}, p \in \mathcal{R}\} \tag{2.9}$$

  where $\mathcal{P}_{\mathcal{G}}$ can be viewed as being a set of static task execution time predictions of $\mathcal{G}$, $t_{i,p}$ is a random sampling of $ET_{i,p}$, $|\mathcal{N}|$ means the number of tasks in DAG $\mathcal{G}$, $V_{|\mathcal{N}|}$ represents the space of all random vectors of size $|\mathcal{N}|$, $C_{|\mathcal{N}|}$ denotes the space of all such vectors each of which has $|\mathcal{N}|$ constant components. Particularly, $\overline{\mathcal{P}_{\mathcal{G}}} = \{\mu_{i,p} : i \in \mathcal{N}, p \in \mathcal{R}\}$ is used to denote the estimation set consisting of the means of task execution time predictions, where $\mu_{i,p}$ is the expected value of $ET_{i,p}$.

- *Deterministic Computation.* There are two types of deterministic computations defined in MCS, one of which is to compute a static schedule. Given a DAG $\mathcal{G}$ and the associated set of static predictions $\mathcal{P}_\mathcal{G}$, any static scheduling heuristic $\mathcal{H}$ (as listed in Section 2.3.1) can be applied to generate a static schedule $\Omega_\mathcal{G}$, namely,

$$\Omega_\mathcal{G} = \mathit{Static\_Scheduling}_\mathcal{H}(\mathcal{G}, \mathcal{P}_\mathcal{G}) \tag{2.10}$$

  Particularly, we have the *mean static schedule* which is obtained by applying $\mathcal{H}$ to $\overline{\mathcal{P}_\mathcal{G}}$, namely,

$$\overline{\Omega}_\mathcal{G} = \mathit{Static\_Scheduling}_\mathcal{H}(\mathcal{G}, \overline{\mathcal{P}_\mathcal{G}}) \tag{2.11}$$

  The other type of computation is to calculate the static makespan of a generated schedule on the assumption that the exact execution time of each task on each resource is known as $\mathcal{P}^*_\mathcal{G}$. This computation can be denoted by

$$m^* = \mathit{Calculate\_Makespan}(\mathcal{G}, \mathcal{P}^*_\mathcal{G}, \Omega_\mathcal{G}) \tag{2.12}$$

Generally, MCS consists of two main phases, namely, '*producing*' and '*selecting*'. In the producing phase, a considerable number of samples are taken from the Input Space and accordingly a long list of different static schedules are generated by employing a specific static scheduling heuristic. Again, in the selecting phase, a certain number of samples are taken to evaluate the generated schedules, which are then compared to obtain the selected schedule for output.

The outline of MCS is depicted in Figure 2.6, where a static scheduling heuristic $\mathcal{H}$ is employed by MCS. Here, the heuristic $\mathcal{H}$ applied (Line 5) can be any static DAG scheduling heuristic. In the outline, the first while loop (Line 3 through 7) describes the producing phase and the second while loop (Line 8 through 14) presents the selecting phase. Before the loops start, the mean static schedule $\overline{\Omega}_\mathcal{G}$ is computed and kept, and $\overline{M}_{std} = \mathit{Calculate\_Makespan}(\mathcal{G}, \overline{\mathcal{P}_\mathcal{G}}, \overline{\Omega}_\mathcal{G})$ is also obtained as a base of the threshold to be used later (Line 2). Next, the producing phase repeatedly takes a sample of the random prediction of the task execution time, i.e., $\mathcal{P}_\mathcal{G}$ (Line 4), and produce a static schedule $\Omega_\mathcal{G}$ by using $\mathcal{H}$ (Line 5). If $\Omega_\mathcal{G}$ has ever been produced before, $\Omega_\mathcal{G}$ is discarded immediately. Otherwise, $\overline{M}_\Omega = \mathit{Calculate\_Makespan}(\mathcal{G}, \overline{\mathcal{P}_\mathcal{G}}, \Omega_\mathcal{G})$ is computed and compared with the

Input: A DAG application $\mathcal{G}$ with stochastic performance prediction $\mathcal{I}_{\mathcal{G}}$.
Output: A full-ahead schedule $\Omega_{\mathcal{G}}$ for $\mathcal{G}$.

1: Create an empty schedule list $\mathcal{L}$.
2: Compute the mean static schedule as defined in Eq.(2.11) and put it into $\mathcal{L}$.
3: **while** the termination conditions of producing phase is not met **repeat**
4: Take a sample of the stochastic performance prediction as defined in Eq.(2.9), which results in a set of static task execution time predictions $\mathcal{P}_{\mathcal{G}}$.
5: Generate a static schedule $\Omega_{\mathcal{G}}$ by applying a static heuristic $\mathcal{H}$ to $\mathcal{P}_{\mathcal{G}}$, as defined in Eq.(6.1).
6: Add $\Omega_{\mathcal{G}}$ into $\mathcal{L}$ if $\Omega_{\mathcal{G}}$ is never produced before and QUALIFIED.
7: **endwhile**
8: **while** the limitation of loops in selecting phase is not reached **repeat**
9: Take a sample of the stochastic performance prediction as defined in Eq.(2.9), which results in a set of static task execution time predictions $\mathcal{P}^*_{\mathcal{G}}$.
10: **for** each schedule $\Omega_{\mathcal{G}}$ in L **do**
11: Evaluate the makespan of $\Omega_{\mathcal{G}}$ by assuming that $\mathcal{P}^*_{\mathcal{G}}$ depicts the actual task execution times of $\mathcal{G}$ as defined in Eq.(2.12).
12: **endfor**
13: **endwhile**
14: Compute the average makespan for each $\Omega_{\mathcal{G}}$ over the loops of evaluation.
15: **Return** the schedule with the minimum average makespan as the result schedule.

Figure 2.6: The outline of MCS

above-mentioned threshold. If $\overline{M}_{\Omega} < \overline{M}_{std} \cdot (1 + \Delta)$, the schedule is QUALIFIED and recorded, and the counterpart $\mathcal{P}_{\mathcal{G}}$ is also kept. Otherwise, the schedule is discarded, where $\Delta \in \mathbb{R}$ is a variable weight to control the strictness of the threshold. The idea behinds this threshold is to reduce the evaluation overhead in the selecting phase. Apparently, if a produced schedule $\Omega_{\mathcal{G}}$ does not have a reasonably short $\overline{M}_{\Omega}$, it is unlikely that it will achieve a good average makespan, and so it is unnecessary to record it. Once the producing phase has been completed, the selecting phase starts loops to evaluate the quality of the produced and recorded schedules. In each loop, a random sample $\mathcal{P}^*_{\mathcal{G}}$ is taken (Line 9), and each recorded schedule is evaluated by computing $m^* = \mathit{Calculate\_Makespan}(\mathcal{G}, \mathcal{P}^*_{\mathcal{G}})$. (Line 10). This loop is repeated until a specified limit is reached, after which the schedule with the minimum average value of m^* over these loops is selected as the final result. (Line 15).

It should be noted that the heuristic H used in the MCS approach can be any deterministic DAG heuristic, while those of low complexity, with little overhead and a good performance on static scheduling problems are preferred. Based on this consideration, HEFT and HBMCT are adopted in this implementation. By

applying MCS to the DAG example shown in Figure 2.1, two sample schedules are obtained, as shown in shown in Figures 2.7 and 2.8. The former is generated by MCS using HEFT, and the latter by HBMCT. As can be seen, these schedules are quite different from the sample schedules generated by deterministic scheduling (as shown in Figures 2.3 and 2.5) and obtain a better makespan with the same setting of task execution times (i.e., the mean).

However, MCS is an iterative approach. Therefore, the number of repetitions executed in both the selecting and producing phases is closely related to the quality of the makespan obtained by the MCS, as well as the extra cost introduced. To achieve an efficient and effective output, it is crucial to determine the appropriate limitations of the necessary loops, i.e., how many iterations should be taken in the loops of the producing and selecting phases (Lines 3 and 8). In order to illustrate these issues, an investigation using a concrete DAG example is presented in the next section.

2.4.1 An Example

In this section, the DAG example provided in Figure 2.1(a) is used, with its stochastic prediction model, to investigate how MCS performs with a varying setting of loop terminations. The test DAG has 12 nodes, and the size of the data to transmitted between each two interdependent tasks is denoted by the value associated with the edge between two connected nodes. It is assumed that the DAG is run on 3 resources. For each random task execution time $ET_{i,m}$ of task i on resource m, it is assumed that the mean $\mu_{i,m}$ is the same as the estimation present in Figure 2.1(b), and that the standard deviation is $\sigma_{i,m} = 0.167\mu_{i,m}$. In addition, the data transmission rates between the 3 resources are provided in Figure 2.1(c). In order to implement MCS, two static DAG scheduling heuristics HEFT and HBMCT are employed. As depicted in Figures 2.9, 2.10 and 2.11, the details of the investigation which applied MCS to the test random DAG are presented as below.

In the first stage of the investigation, the taking of random sampling in the producing phase of MCS was repeated for a considerable number of times to observe how the number of differently produced schedules (N_{ps}) and the elapsed time evolves. The evolution result is shown in Figures 2.9(a) and 2.9(b), where HEFT and HBMCT are separately used. In both figures, N_{ps} is scaled by the left Y axis, and the elapsed time by the right one. It should be noted that in both X

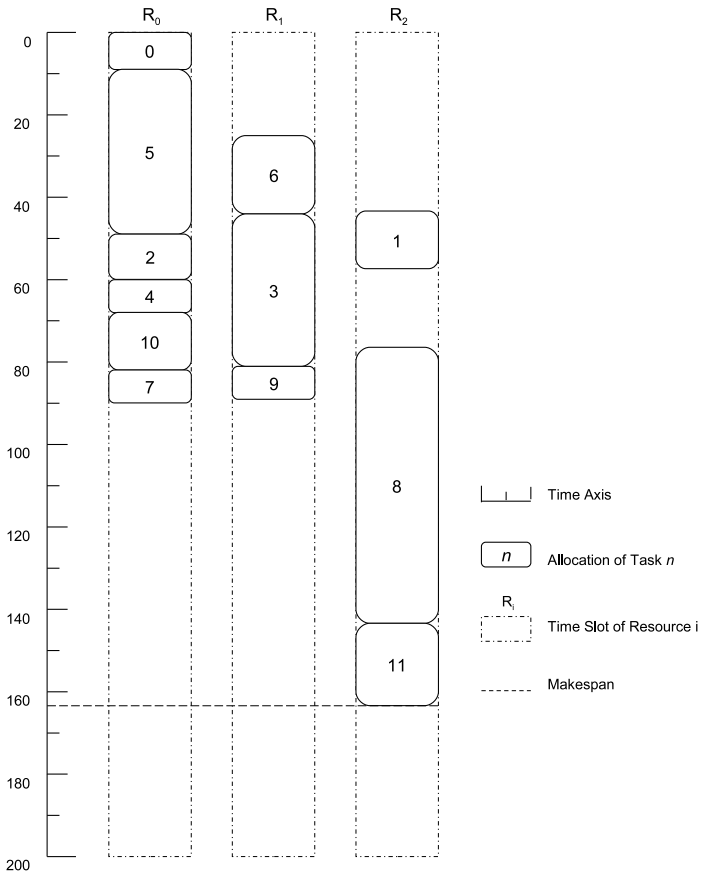


Figure 2.7: The schedule result produced by MCS using HEFT

axis and right Y axis, a Log10 scale is adopted. In the case of using HEFT, the producing lasted for about 8 hours and a total of 9020 different schedules were obtained. On average, one iteration lasted about 0.2 milliseconds. In the first 5000 loops which lasted about 1 second in total, 291 new schedules were found, and then the emergence of new schedules became rarer and rarer as the iteration was repeated. The situation was similar when HBMCT was used. After about 8 hours, the repeated loops produced a total of 9394 schedules and each repetition lasted about 0.19 microseconds. 246 new schedules were found in the first 5000 repetitions, while the number of schedules produced every 5000000 repetitions became fewer than 10 by the end of the producing phase. At this stage, it was presumed that the sample space of the schedules which each heuristic can produce had been extensively explored.

Next, up to 200 samples were taken in the selecting phase to evaluate the

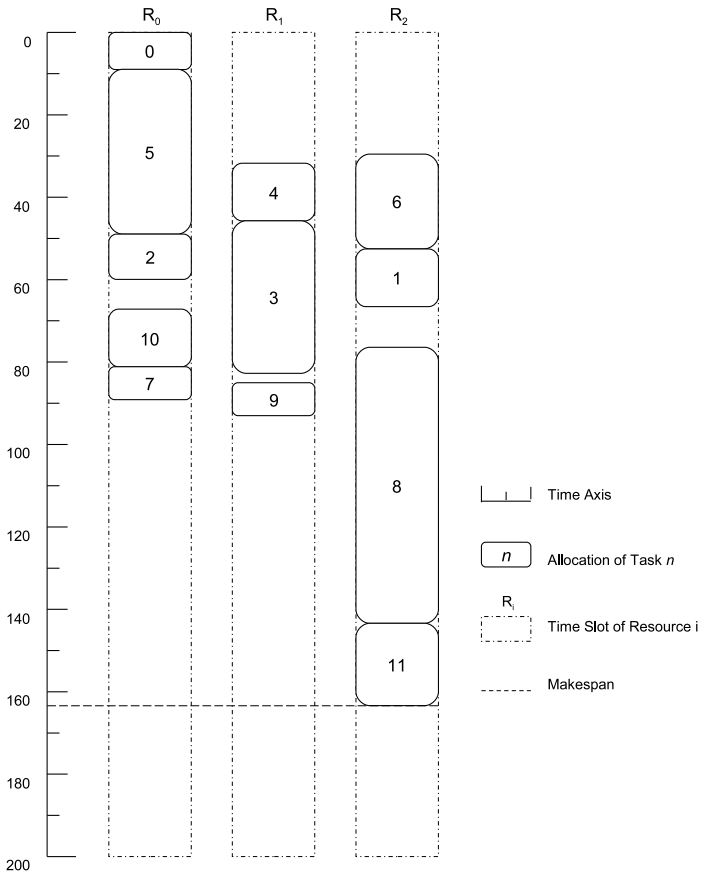
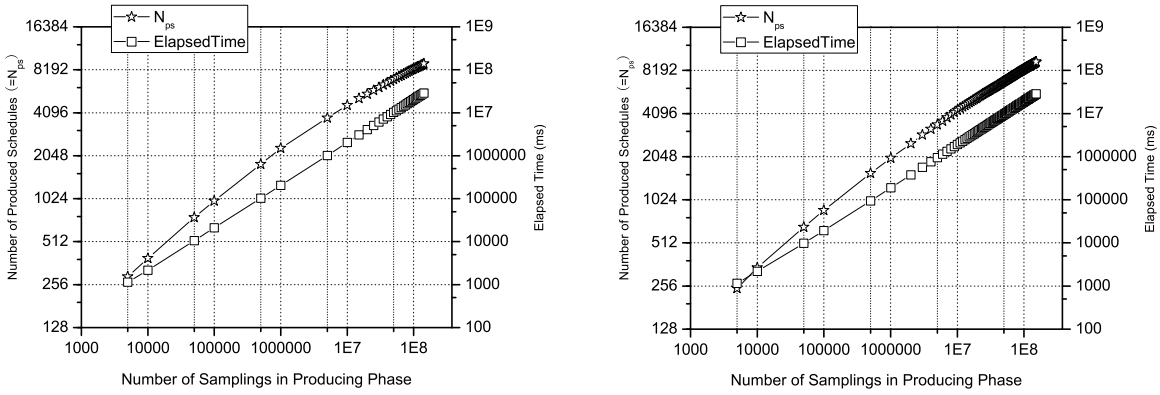


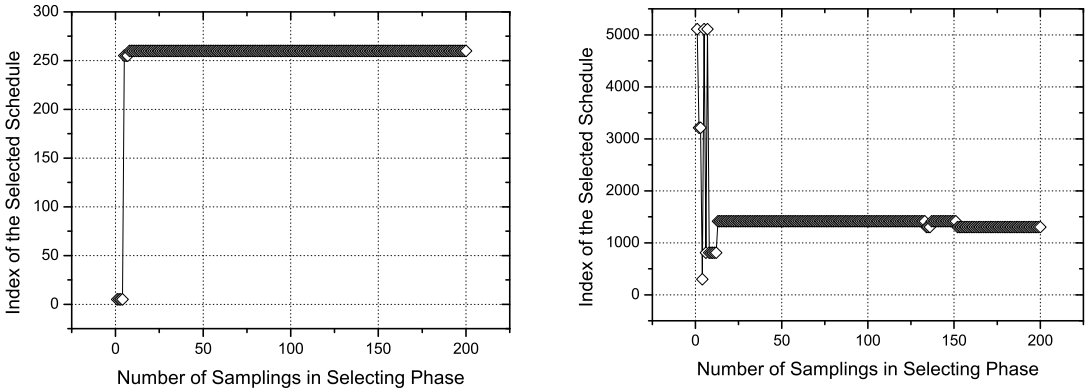
Figure 2.8: The schedule result produced by MCS using HBMCT

produced schedules and to observe how the index of the selected output schedule (denoted by I_{sel}) evolved as the number of samplings in selecting phase (evaluation sampling, hereafter) increased. Figure 2.10 shows the evolution of I_{sel} when the number of evaluation samplings changed. It can be seen that, when the number of evaluation samplings used is small (for example, fewer than 10), the result of I_{sel} may not be stable. However, the result of I_{sel} tends to converge into a constant value as the number of evaluation samplings becomes large. Especially in the case of HEFT, the result of I_{sel} remains the same when more than 10 evaluation samplings are used. In the case of HBMCT, three schedules may possibly be selected when more than 10 evaluation samplings are used. These schedules actually obtain average makespans which are close to each other. These are 164.33, 163.91 and 163.85 respectively in detail. This implies that as long as sufficient evaluation sampling is taken, the makespan of MCS will fall into a small



(a) using HEFT (b) using HBMCT

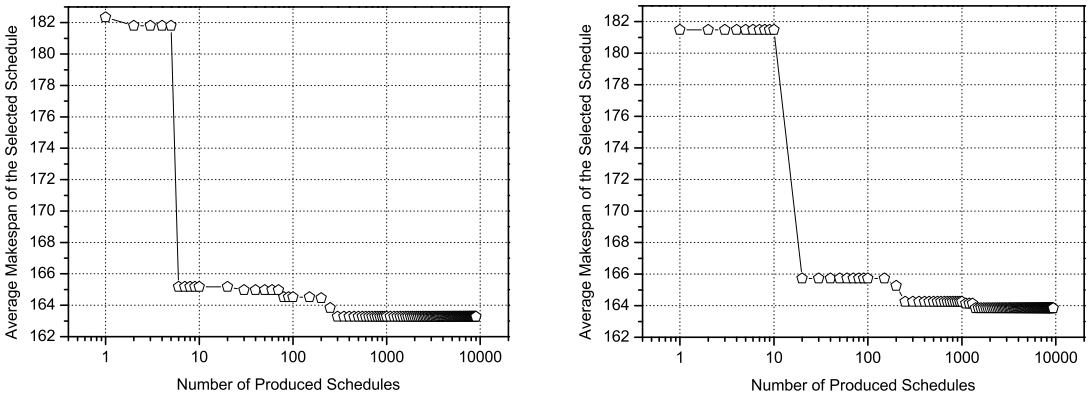
Figure 2.9: Evolution of N_{ps} and Elapsed Time during the producing phase of MCS for the test random DAG



(a) using HEFT (b) using HBMCT

Figure 2.10: Evolution of I_{sel} during the selecting phase of MCS for the test random DAG

range, even though the selection of the result schedule may be variable.



(a) using HEFT (b) using HBMCT

Figure 2.11: Evolution of M_{sel} as the number of produced schedules varies for the test random DAG

Finally, how the performance of MCS, which can be reflected by the average makespan evaluated in the selecting phase (M_{sel}), evolved as the number of produced schedules grew, was investigated. As shown in Figures 2.11(a) and 2.11(b), the result of M_{sel} achieved significant improvement as the first hundreds of schedules were produced, but there was little improvement afterwards. In the case of using HBMCT, the M_{sel} dropped 9.5%, from 181.48 to 164.23 after the first 500 different schedules were produced, which means that less than 10 seconds were spent on the producing phase according to Figure 2.9(a). This M_{sel} continued to be improved, but slowly, as more new schedules were produced. Subsequent improvement was only 0.23% until the end of the producing phase, which lasted for 8 hours. Similarly, in the case of HEFT, the M_{sel} was improved by 10.5% after the first 500 schedules were produced and no further improvement was made thereafter. This implies that exhaustive exploration in the producing phase, which may find numerous possible schedules but results in heavy overheads for MCS, may be unnecessary, because the majority of the improvement is usually achieved by the first few iterations.

In summary, it can be seen from the above example that (i) in the producing phase of MCS, a reasonable number of loops are sufficient to produce a 'good' schedule, which can significantly improve the average makespan, in the schedule list; (ii) in the selecting phase of MCS, only around 20 random samples are necessary in the evaluation of the produced schedules to pick up a 'good' schedule

from the list as the result schedule. This indicates that it may be feasible for MCS to substantially reduce its overheads without significantly losing its performance by setting a small limit of the repeated samplings in both the producing and selecting phases, and consequently, applying it to the large applications consisting of many tasks.

2.5 Evaluation

In order to evaluate the performance of MCS, several experiments were conducted to compare the makespan of the schedule obtained by MCS employing static scheduling heuristic $\mathcal{H}$ with schedules generated by other approaches which are respectively denoted as *Static*, *Autopsy* and *Static Schedule with Rescheduling*.

- *Static* refers to the schedule which is computed by applying $\mathcal{H}$ to the mean of the random task execution time predictions (i.e., $\mu_{i,p}$).
- *Autopsy* refers to the schedule which is constructed by $\mathcal{H}$ after the actual task execution times are known.
- *Static Schedule with Rescheduling* (*ReStatic*, for short) means the procedure which firstly generates an initial schedule by the Static approach and then recalls the heuristic used in the Static approach to reschedule all of the remaining non-executed tasks each time a task is about to begin execution.

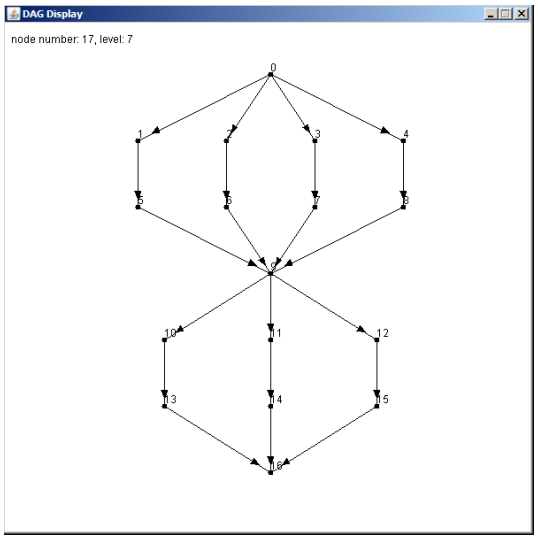
In addition, applying rescheduling to MCS, denoted by *ReMCS* was also considered. ReMCS uses the result of MCS as the initial schedule, and then calls the heuristic used in MCS to reschedule all of the remaining non-executed tasks each time a task is about to begin execution. It should be noted that when making a rescheduling decision, ReMCS adopts the task execution time sample $\mathcal{P}_{\mathcal{G}}$, which is recorded with $\Omega_{\mathcal{G}}$, as input.

The loop termination condition was set for the MCS compared in the evaluation. In detail, the producing phase terminates when it reaches the time limit of 60 seconds or the number of produced schedules reaches 10000, and in the selecting phase, each produced schedule is evaluated and compared by an average makespan over 20 random samplings. $\Delta = 0.02$ was set as the threshold used in the producing phase.

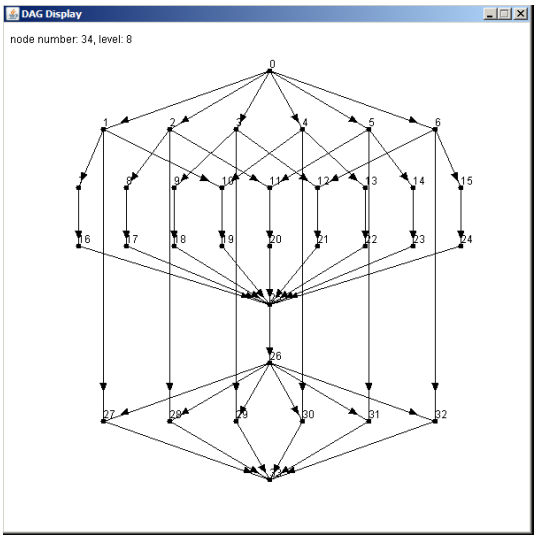
The above-stated scheduling approaches were compared using four types of DAGs, with different structures and sizes, running on three heterogeneous resources. The selected DAGs included fMRI [ZWF+04] with 17 nodes, Montage [BGL+04] with 34 nodes, AIRSN [HDW+05] with 53 nodes and LIGO [DKM+02] with 77 nodes as shown in Figure 2.12. These DAGs were derived from various scientific applications and could cover a wide spectrum of DAG topologies and DAG sizes. It should be noted that Figure 2.12 was derived from a collection of screenshots of the evaluation program where all of the DAGs were standardized to have only one entry node and one exit node. The method in [ASMH00] was adopted to model the heterogeneity of the mean of task execution time prediction which was randomly generated in the range of [1, 100]. In the method, in brief, two values are respectively selected from a uniform distribution at the range of [1, 10], and then the product of the two selected values is computed and adopted as a generation of a task execution time. The communication-to-computation ratio (CCR) was randomly chosen from the interval [0.1, 1]. As assumed in MCS, the actual execution time of each task was randomly distributed within a certain boundary around the mean of execution time estimation. Similar to the notion QoI defined in [SZ04b], the notion of *Quality of Estimation* (QoE, denoted by $0 < \delta < 1$) was to describe the upper boundary of percentage deviation which the actual task execution time may have in respect of the mean value. For example, given that $\delta = 0.5$ and the mean estimated task execution time is $\mu_{i,p} = 50$, then the task execution time estimation is an uniform random variable $ET_{i,p}$ distributed at intervals of $[(1-\delta)\mu_{i,p}, (1+\delta)\mu_{i,p}] = [25, 75]$, and the actual time is a random sample of $ET_{i,p}$.

In one experiment, a DAG was generated to which the relevant parameters stated above were applied. By varying the setting of QoE from 0.1 to 0.5 (0.1 per step), a different scheduling approach was applied to the DAG. The makespan of each approach for this DAG averaged over 100 random generations of actual execution time. Such an experiment was repeated 50 times and the average result of each approach is considered. Also, the average time cost of applying MCS to a DAG was considered, and all of the experiments were run on a PC with Pentium 4 CPU of 3.2 Ghz and 1 GB Memory.

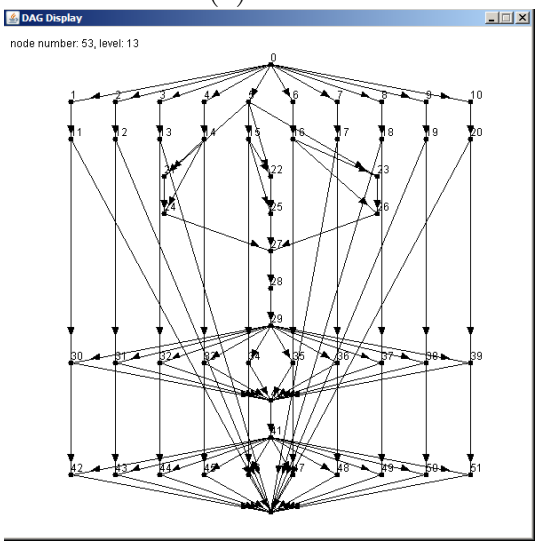
The performance results of each of the five different approaches on four different types of DAGs are shown respectively in Figures 2.13, 2.14, 2.15, and 2.16.



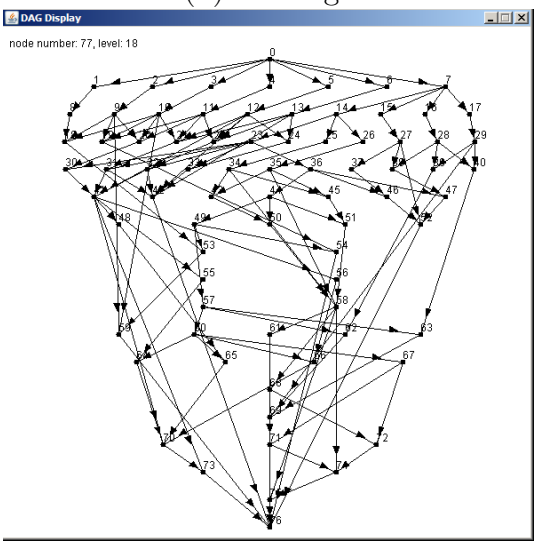
(a) fMRI



(b) Montage

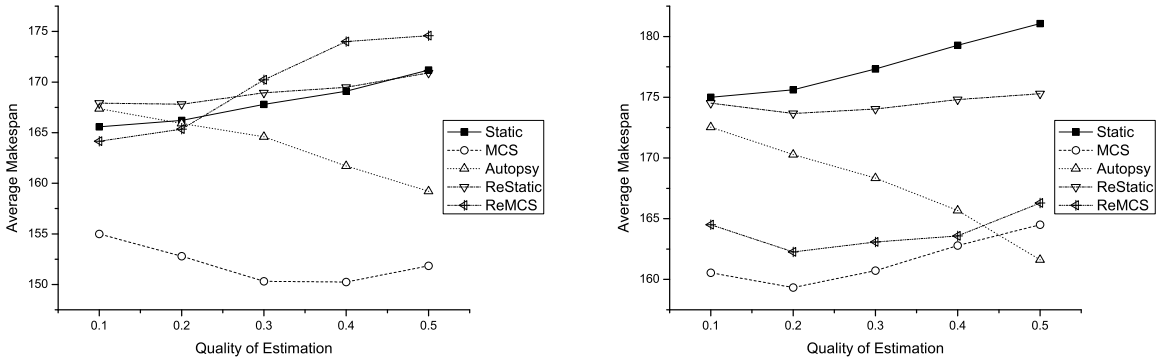


(c) AIRSN



(d) LIGO

Figure 2.12: DAG examples

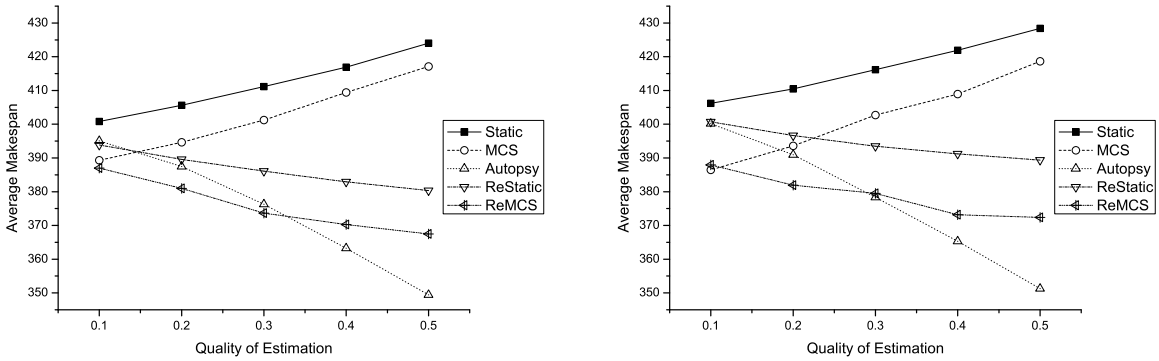


(a) using HEFT (b) using HBMCT

Figure 2.13: Average makespan (over fifty fMRI DAGs with 17 nodes running on 3 resources) of five compared scheduling approaches

In the case of fMRI DAG with 17 nodes (Figure 2.13), it is interesting to see that, when using either HEFT or HBMCT, MCS outperforms other scheduling approaches in almost all QoE settings. In all of the settings of QoE, MCS can achieve an average makespan which is 10% better than Static. However, when HEFT is used, the results are surprising: (i) ReStatic with HEFT in this case can worsen the makespan of Static, especially when QoE is small; (ii) ReMCS does not perform as well as expected, which degrades the performance of MCS even more; (iii) Autopsy does not exhibit a good performance when QoE is less than, or equal to, 0.2, but performs second best when QoE is equal to 0.3 or larger. When HBMCT is used, the result is not that surprising. Although ReMCS still degrades the performance of MCS, it performs the second best except when QoE is equal to 0.5. As expected, by using HBMCT, Autopsy obtains a better average makespan than ReStatic which, in turn, performs better than Static. By comparing HEFT with HBMCT, it can be seen that: for MCS, using HEFT obtains a shorter average makespan than using HBMCT, while for ReMCS, using HBMCT turns out to be better than using HEFT.

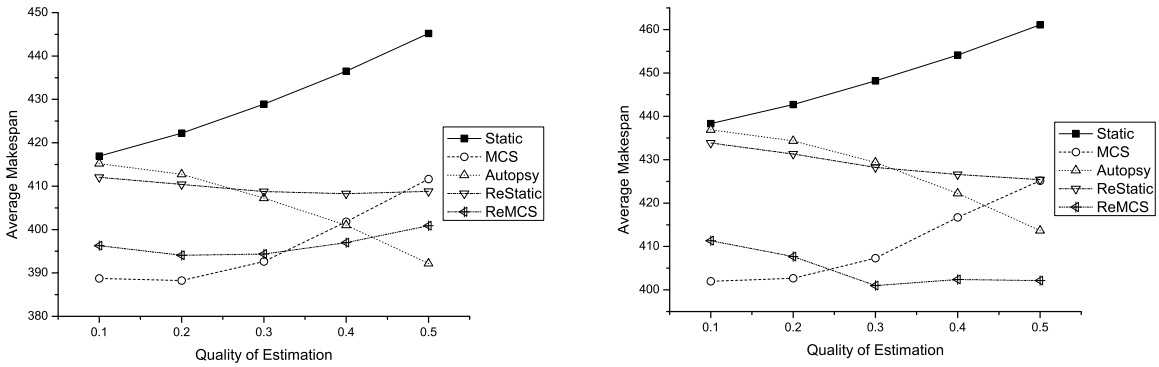
In the case of Montage DAG with 34 nodes (Figure 2.14), there seems to be little difference between using HEFT and HBMCT. Therefore the following analysis applies to both scenarios where HEFT and HBMCT are used separately: Static always obtains the worst average makespan in all settings of QoE. MCS and ReMCS performs best when QoE is equal to 0.1. When QoE is equal to 0.2, ReStatic and Autopsy performs as well as, or better than, MCS but worse



(a) using HEFT (b) using HBMCT

Figure 2.14: Average makespan (over fifty Montage DAGs with 34 nodes running on 3 resources) of five compared scheduling approaches

than ReMCS. When QoE is equal to 0.3, ReMCS and Autopsy perform the best, while ReStatic performs better than MCS. When QoE becomes larger, Autopsy outperforms other approaches, while the results of comparing other approaches remains the same.

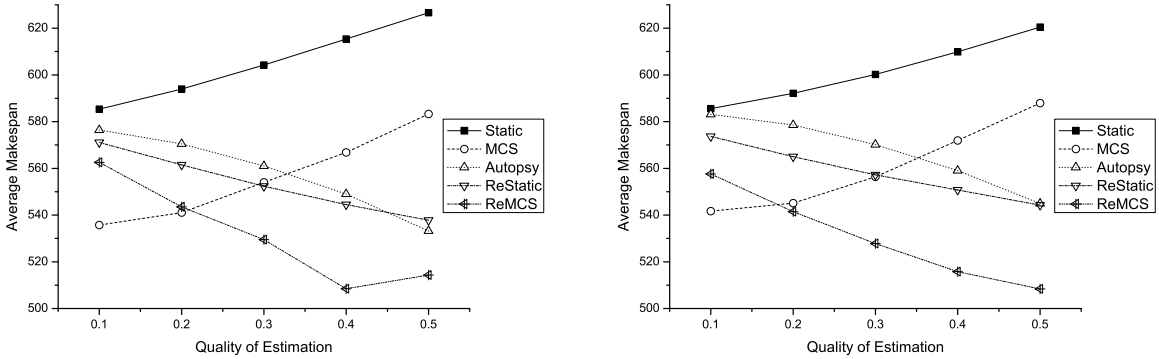


(a) using HEFT (b) using HBMCT

Figure 2.15: Average makespan (over fifty AIRSN DAGs with 53 nodes running on 3 resources) of five compared scheduling approaches

In the case of AIRSN DAG with 53 nodes (Figure 2.15), Static produces the worst performance in all settings of QoE and when using for both HEFT and HBMCT. ReStatic can improve the result of Static to obtain an average makespan close to the result of Autopsy when QoE is less than 0.3. When QoE is small (for example, less than 0.2), MCS obtains the best average makespan,

while ReMCS takes the top place when QoE is equal to 0.3 and 0.4. When QoE is equal to 0.5, by using HEFT, Autopsy performs the best while, when using HBMCT, ReMCS outperforms other approaches. In terms of using HEFT or HBMCT, MCS can usually obtain better results by using HEFT, while ReMCS obtains similar results when using them, especially when QoE is larger than 0.4.



(a) using HEFT (b) using HBMCT

Figure 2.16: Average makespan (over fifty LIGO DAGs with 77 nodes running on 3 resources) of five compared scheduling approaches

In the case of LIGO DAG with 77 nodes (Figure 2.16), again, the displayed results appear to be similar when using either HEFT or HBMCT. Static still displays the poorest performance in all of the settings of QoE. The performance of ReStatic is always located in the position between Static and Autopsy. MCS performs better than Autopsy when QoE is less than 0.3, but worse when QoE is greater than 0.3. ReMCS always achieves a better average makespan than Autopsy, but does not perform as well as MCS when QoE is equal to 0.1.

In summary, several deductions can be made from the above observations, as follows:

- MCS can usually obtain a better average makespan than Static with run-time changes of task execution time;
- Applying rescheduling to the schedule produced by MCS usually performs better than applying rescheduling to the schedule generated by the Static approach;
- The rescheduling technique is more effective in a highly dynamic context than a low one;

- When the distribution of the random task execution time estimation is known, applying MCS and/or ReMCS can achieve even better results than the Static with perfect task execution time estimation, i.e., Autopsy.

These deductions demonstrate the advantages of the proposed Monte-Carlo based scheduling approach. When the performance prediction is uncertain, the proposed approach can outperform the existing solutions on minimizing a workflow makespan in most of cases.

Apart from comparing the performance of MCS with other scheduling approaches, the time cost of MCS was also measured with the aforementioned parameter setting in the experiments. The average time cost results of two separate MCS approaches using HEFT and HBMCT are shown in Tables 2.4 and 2.5. In each table, the average time costs for the producing phase and the selecting phase are also measured. As can be seen from the results, in most cases, the average time cost of running MCS is around 60 seconds. When finding 10000 newly qualified schedules, the producing phase is completed sooner than the time out limit of 60 seconds, while the time cost of the selecting phase implies the number of schedules produced in the producing phase.

Combined with the performance evaluation results, the time cost results suggest that, apart from the substantial improvement made on application performance during run-time with task execution time changes, the MCS may also introduce a significant overhead during scheduling, compared to the list scheduling heuristics [SZ04a]. However, this may not be a fundamental limitation of a static full-ahead scheme [KL05], especially in cases where the scheduler has sufficient time to make scheduling decisions before the application is ready to start execution. In addition, as demonstrated in Section 2.4.1, by carefully setting the termination conditions of the producing and selecting phases, the scheduler can easily strike a good trade-off between the scheduling overhead and the performance improvement.

2.6 Closing Remarks

This chapter addressed a stochastic workflow scheduling problem which models the uncertainty of task time estimation, which affects the performance of workflow execution in a stochastic manner. The chapter began by investigating the extensively studied deterministic scheduling problem, after which an overview

	Total Time Cost for MCS (= Producing Phase + Selecting Phase)			
	fMRI (17 nodes)	Montage (34 nodes)	AIRSN (53 nodes)	LIGO (77 nodes)
$\delta = 0.1$	60036.26	39871.54	57114.24	61745.68
$\delta = 0.2$	60467.22	57990.40	62156.14	68109.30
$\delta = 0.3$	61006.52	64058.22	65455.24	71790.94
$\delta = 0.4$	59597.84	63437.18	64288.72	68218.80
$\delta = 0.5$	56771.92	62227.22	63044.04	63587.80

(a) Average Time Cost for the whole MCS

	Time Cost for Producing Phase			
	fMRI (17 nodes)	Montage (34 nodes)	AIRSN (53 nodes)	LIGO (77 nodes)
$\delta = 0.1$	60000.00	26423.82	37793.28	35382.52
$\delta = 0.2$	60000.00	47711.66	45953.98	45180.66
$\delta = 0.3$	59399.02	57642.94	53856.22	55331.62
$\delta = 0.4$	57097.52	59703.14	57142.74	59435.92
$\delta = 0.5$	53920.44	60000.00	58992.52	60000.00

(b) Average Time Cost for the Producing Phase

	Time Cost for Selecting Phase			
	fMRI (17 nodes)	Montage (34 nodes)	AIRSN (53 nodes)	LIGO (77 nodes)
$\delta = 0.1$	36.26	13446.12	19318.8	26361.28
$\delta = 0.2$	467.22	10278.12	16200.3	22927.74
$\delta = 0.3$	1607.2	6414.04	11599.02	16458.68
$\delta = 0.4$	2500.0	3734.04	7145.98	8782.88
$\delta = 0.5$	2851.48	2226.92	4051.2	3587.8

(c) Average Time Cost for the Selecting Phase

Table 2.4: Average time cost (in msec) for applying MCS with HEFT to different types of DAG with different settings of QoE (denoted by δ)

was given of the common concepts and assumptions adopted in the deterministic scheduling problem. This was then extended into the stochastic context.

The existing techniques to resolve the deterministic scheduling problems were summarized, and an extensive taxonomy of well-known deterministic scheduling heuristics was presented. An example was provided to enhance the depiction of the problem and support the understanding of the described heuristics. The majority of these heuristics were implemented, and contributed to the evaluation of the robustness of a deterministic heuristic in the stochastic context, which enabled the conclusion of the limitation of these heuristics in the stochastic context.

This chapter has proposed a full-ahead scheduling scheme to tackle the scheduling problem with a stochastic performance prediction. Based on the Monte-Carlo method, the proposed approach, MCS, seeks such a full-ahead schedule which can obtain a competitive average makespan no matter how the stochastic task execution times vary. An extensive simulated evaluation was undertaken and the experimental results demonstrated that an MCS employing a deterministic scheduling heuristic could always outperform the application of the heuristic using the mean

	Total Time Cost for MCS (= Producing Phase + Selecting Phase)			
	fMRI (17 nodes)	Montage (34 nodes)	AIRSN (53 nodes)	LIGO (77 nodes)
$\delta = 0.1$	60041.00	44829.74	63494.46	69759.04
$\delta = 0.2$	60384.96	62396.58	64106.80	73858.46
$\delta = 0.3$	61078.64	64714.08	67489.44	69855.30
$\delta = 0.4$	60553.42	63827.76	66163.76	65694.08
$\delta = 0.5$	60195.62	62185.28	64478.68	62158.14

(a) Average Time Cost for the whole MCS

	Time Cost for Producing Phase			
	fMRI (17 nodes)	Montage (34 nodes)	AIRSN (53 nodes)	LIGO (77 nodes)
$\delta = 0.1$	60000.00	32081.66	45287.90	42455.26
$\delta = 0.2$	60000.00	52415.94	46609.00	52959.36
$\delta = 0.3$	60000.00	58324.40	54146.52	57701.64
$\delta = 0.4$	58894.68	60000.00	57821.92	59869.90
$\delta = 0.5$	58265.24	60000.00	60000.00	60000.00

(b) Average Time Cost for the Producing Phase

	Time Cost for Selecting Phase			
	fMRI (17 nodes)	Montage (34 nodes)	AIRSN (53 nodes)	LIGO (77 nodes)
$\delta = 0.1$	41.0	12747.14	18204.96	27303.78
$\delta = 0.2$	384.02	9979.7	17496.22	20899.1
$\delta = 0.3$	1079.64	6389.06	13342.92	12153.34
$\delta = 0.4$	1658.1	3827.46	8341.54	5823.22
$\delta = 0.5$	1930.06	2184.96	4478.36	2158.14

(c) Average Time Cost for the Selecting Phase

Table 2.5: Average time cost (in msec) for applying MCS with HBMCT to different types of DAG with different settings of QoE (denoted by δ)

of random performance prediction by an average makespan. In some cases, the schedules obtained by MCS, with or without rescheduling, could even perform better, on average, than those schedules built after the accurate execution time prediction for each task on each resource is known. It was also suggested that MCS can be applied to larger applications with controllable overheads.

Although it can improve workflow performance under prediction uncertainty, the proposed Monte-Carlo scheduling approach, as a full-ahead scheme, has to rely on performance prediction to make a scheduling decision. The next chapter will consider the problem of how to schedule a workflow to cope with grid uncertainties in a situation where there is no means to obtain a performance prediction.

Chapter 3

Just-in-time Scheduling to Maximize Ready Tasks

The previous chapter focuses on a full-ahead scheduling scheme which requires the stable availability of a set of resources and information about performance estimation. However, in some practical Grid computing situations, workflow scheduling may be faced with severe uncertainty, where, hardly any information can be obtained about the underlying platform. In such a scenario, a just-in-time scheduling scheme is often a better choice than a full-ahead scheme. This chapter concentrates on the development of a just-in-time scheduling scheme when limited information about the environment is available.

The main contribution of this chapter is a novel low-cost just-in-time scheduling heuristic for DAG applications, which is not only useable for any possible DAG, but also exhibits a comparable or better performance than the existing solutions.

3.1 Background

Although many full-ahead static heuristics have been proposed for DAG scheduling to minimize the makespan (the whole application execution time) for heterogeneous systems [THW02, SZ04a], these approaches may not be suitable for highly dynamic Grid environments due to the limitation of their assumptions. Full-ahead scheduling schemes normally rely on future information and assume that (i) the resources will always be available when needed, and (ii) the inter-dependent task communication time and task execution time can be accurately

predicted. However, in some cases, there may be no knowledge about environmental information, or the information may be too hard to be obtained in grids with a temporal unpredictability [MFR07]: (i) the interdependent task communication over the Internet may not be stable; (ii) the non-dedicated Grid resources in charge of executing the DAG tasks may exhibit uncertain availability and/or task processing rate. Some approaches have been suggested to address a particular part of this uncertainty, for example, the unpredictability of the task execution time has been taken into account in rescheduling [SZ04b] and the stochastic scheduling work presented in the previous chapter, whereas they still rely on the guaranteed availability of resources.

In contrast, a just-in-time scheduling scheme attempts to minimize the impact caused by grid uncertainties by making scheduling decisions on-the-fly [AG91, PLR+95], which means that the allocation of a task will not be decided until the task becomes ready for execution, i.e., its parents have finished their execution. Without considering any task-resource mapping before the execution of an application, a just-in-time scheduling heuristic only determines an allocation order of the application's tasks. During runtime, ready tasks are allocated in the allocation order to resources for execution. With increasing attention in recent years, the just-in-time scheduling scheme has been adopted in quite a few practical grid projects such as Condor [TTL05], Pegasus [DBG+04] and Triana [TPS+02, TSW03], where some simple scheduling strategies, for instance FIFO, have been used. A scheduling pattern which allocates ready tasks to resources immediately when they become available, is encouraged in large computation environments where, from a user's point of view, free resources may quickly become busy if not allocated immediately [SB08a]. In this case, it has been observed that, due to the aforementioned temporal unpredictability, using FIFO to sequence the allocation of tasks may lead to an ineffective execution of an application with complex task dependencies [MRY06]. This is because the FIFO, which does not have any deliberate prioritization of task allocation, may encounter a so-called gridlock [MRY06], namely, there is no ready tasks for allocation when a set of resources requests arrive.

In order to minimize the risk of encountering this situation, a new scheduling goal has been suggested by a series of papers [MRY06, MFR07] to schedule the tasks of a DAG application in such an order that the number of ready tasks generated during execution is as large as possible for assignment to the resources

becoming available. Intuitively, once the tasks of an application are executed in such a order, the application should obtain a considerable parallel speedup and the resource requests should be better utilized no matter how the dynamic resources behave. Following the suggested scheduling goal, a decomposition-based heuristic named IC-Optimal (ICO, hereafter) and its extension have been proposed in [MFR07]. It is shown in [HRV07] that ICO outperforms some simple scheduling strategies on the maximization of the number of ready tasks, and the makespan.

Compared to full-ahead scheduling heuristics, ICO has an important advantage, in that it takes the DAG structure only as input and does not require any information about performance estimation. It may be argued that overlooking task execution and communication cost may result in a degraded scheduling performance. In a case where the task execution cost has a significant impact on the scheduling process and a performance prediction can somehow be obtained, the classic critical path based scheduling heuristic may outperform ICO [SB08a]. Nevertheless, when the actual task execution and communication time is trivial to the scheduling process [MFR07] or the performance estimate is highly inaccurate [SB08b], the scheduling method which focuses on maximizing ready tasks can still be a better choice.

The strategy of maximizing the number of ready tasks may be promising for DAG scheduling on the Grid. However, restricted by the decomposition-based design feature, ICO may have the following three disadvantages: (i) high complexity, (ii) incomplete applicability — the decomposition procedure may fail at some DAG topologies in terms of the algorithm description, and (iii) oversight of global optimization — the optimum schedule may possibly be ignored due to the decomposition. This indicates the necessity and the possibility of developing a new approach with lower execution cost, complete applicability for arbitrary DAG topology and better performance to achieve the same scheduling goal.

Given this motivation, the focus is put upon developing a low-cost just-in-time approach with the aim of maximizing the number of ready tasks in order to resolve the workflow scheduling problem with temporal unpredictability. This chapter proposes a novel priority-based heuristic (PB), which uses a numerical priority, instead of the decomposition-based techniques used by ICO, and has advantages of: (i) low running costs, (ii) complete applicability to arbitrary DAG topologies, and (iii) improved performance with respect to the scheduling aim.

The proposed heuristic is therefore expected to provide an efficient solution of a just-in-time scheme for Grid computing systems.

The remainder of the chapter is organized as follows: the preliminaries and the problem statement are presented in Section 3.2; the related work is reviewed in Section 3.3; the proposed heuristic is introduced in Section 3.4; the complexity of the heuristic is analysed in Section 3.5; the performance of the heuristic is evaluated in Section 3.6; finally, a summary is provided in Section 3.7.

3.2 Preliminaries

We focus on workflow applications which can be represented by a DAG modelled as follows: a DAG $G = \{N, E\}$ is a directed graph consisting of a set of nodes N and a set of edges E, each of which is of the form $(i \rightarrow j)$, where $i, j \in N$. A node i represents the counterpart task, and an edge $i \rightarrow j$ denotes the inter-task dependency between i and j. The execution of j cannot begin until the execution of i has been completed, and j becomes a *ready node* when all of its parents are completed. Given an edge from i to j, i is called a parent node of j, and j a child of i. Parentless nodes are called *source nodes*, and childless nodes *sink nodes*. For standardisation, it is specified that the DAG has a single entry node and a single exit node, since all DAGs with multiple entry or exit nodes can be equivalently transformed to this specification. Apparently, an entry node of G must be a source node, and an exit node must be a sink node.

As mentioned in Section 3.1, the goal is to determine a schedule S for the given DAG G (a permutation of tasks indicating the order of assigning tasks to resources in this chapter) which maximizes the number of ready tasks for mapping to new resources when they become available. The study proceeds under an idealized assumption that tasks will be executed in the order of their allocation [MRY06]. Thus, the goal becomes to produce as many ready tasks as possible after each task execution. $NR_S(i)$ is used to represent the number of ready tasks produced at the completion of the ith task in the order of execution. Obviously, $NR_S(n-1) \equiv 1$ due to the DAG standardisation adopted. A schedule S^* is called the *optimal schedule* if it maximizes the number of ready tasks at the completion of each task, i.e.,

$$(\forall i) NR_{S^*}(i) = \max_{S' \in S_G} \{NR_{S'}(i)\} \tag{3.1}$$

where S_G is the set of all possible schedules of G and $0 \leq i < n$. It is worth

mentioning that some DAGs do not admit any optimal schedule [MRY06], and for these DAGs, a metric called normalized *AREA* [CR09] (denoted by $V(S)$) as defined below, is used to describe the implicit quality of a schedule.

$$V(S) = \frac{\sum_{i=0}^{n-1} NR_S(i)}{n} \tag{3.2}$$

Given these definitions, the scheduling problem is to sort all of the tasks of a given DAG into a suitable order to obtain the optimal schedule if the DAG admits one, or to maximize $V(S)$ if it does not.

3.3 Related Work

The DAG execution model presented in Section 3.2 can be formalized by the Internet-Computing (IC) Pebble Games which uses pebbles to model the execution of a DAG. The placement and/or removal of various types of pebbles are used to represent the transition of task status (for example, ready and completed). Such games have been studied in [Ros04, RY05, MR05, MR06] for executing DAGs on the Internet. These studies were extended to the ICO algorithm proposed in [MRY06] to obtain optimal schedules for DAGs with some a specific structure. Simulation experiments carried out in [HRV07] indicate that ICO significantly improves the execution time of a large class of DAGs over three simple, intuitively compelling scheduling heuristics. Malewicz et al. [MFR07] extended the ICO algorithm to a practical heuristic applied in the Condor Project [TTL05], and the usefulness of its implementation was assessed in [SB08a, SB08b].

The ICO heuristic is discussed based on the paper of [MFR07]. Although some other improvements of the ICO are published in [CMR06, CMR07b, CMR07a, SCR08], they do not conflict with the discussion. As mentioned in Section 3.1, the ICO is a decomposition-based heuristic derived from the observation that some simple DAGs, (so-called Connected Bipartite Building Blocks (CBBB) [MRY06]) consisting entirely of source nodes and sink nodes, can easily obtain an optimal schedule by first running all of their source nodes in a certain order followed by the sink nodes in an arbitrary order. A couple of examples of CBBB (as shown in Figure 3.1) are provided in [MRY06, CMR06], where most of the CBBBs are

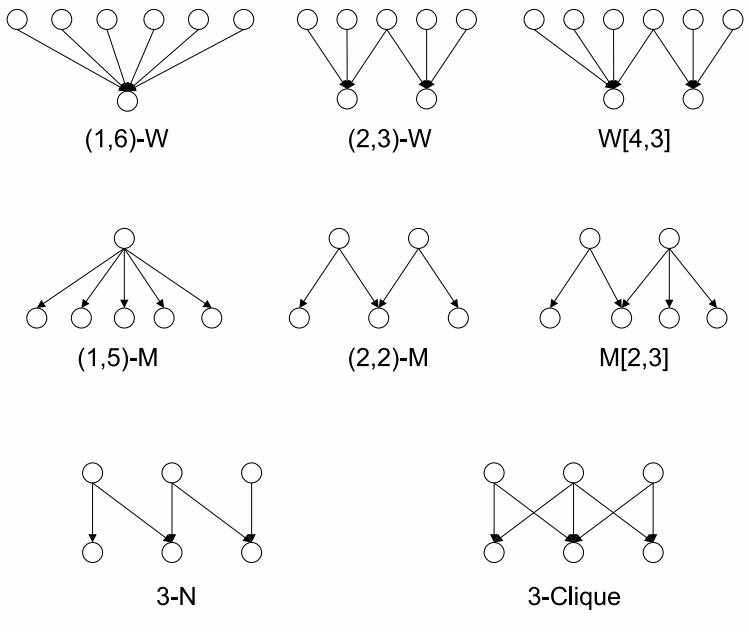


Figure 3.1: Examples of CBBBs

named after letters suggested by their topologies, and several integers which specify their source and/or sink numbers (as the name shown beneath the CBBBs in Figure 3.1). Some scientific applications can be considered to be a composition of CBBBs, such as Laplace, FFT, and Fork-join [SZ04a] as illustrated in Figure 3.2. In addition, more patterns of CBBBs have been expanded in [CMR07a] to allow for the composition of larger families of DAGs. This grounds the idea of decomposing the DAG application to blocks, like some kinds of CBBBs, obtaining the ICO schedule for all of the blocks and then combining them to form the final schedule for the whole application.

As shown in Figure 3.3, the procedure of ICO consists of six steps, which are briefly described as follows (the detailed description and illustration is referred to [MFR07]): (i) remove every short cut (i.e., every edge from node u to v satisfying that v can be reached from u other than going through this edge [MFR07]) of G; (ii) decompose G into building blocks (can be CBBB or not); (iii) generate a schedule for every building block; (iv) compute inter-block priority (priority within every pair of building blocks); (v) determine the execution order of the building blocks; (vi) combine the schedule of every building block to build the final schedule.

Although ICO has been evaluated to transcend some simple scheduling schemes,

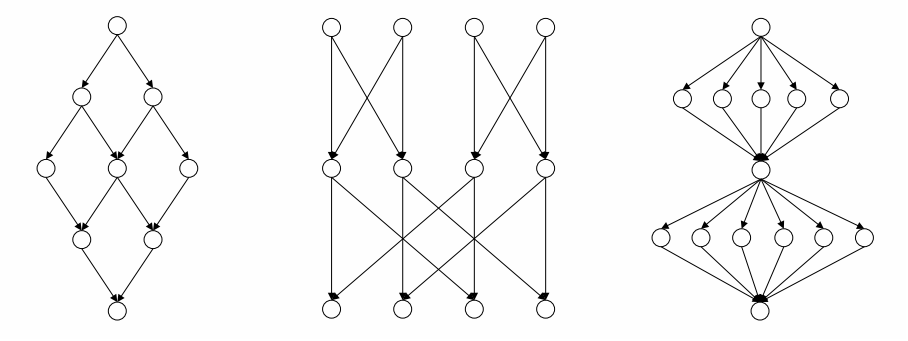


Figure 3.2: Examples of Scientific Application: Laplace(Left), FFT(Center), and Fork-join(Right)

for example, FIFO for just-in-time schedulers [MFR07, HRV07], the decomposition-based design feature of ICO causes it to have inevitable drawbacks, which include:

- Difficulty of implementation. According to the algorithm described in [MFR07], ICO greatly relies on the manipulation of some complex graph data structures, e.g., building block, CBBB, super dag [MFR07] etc, which are usually difficult to implement.

- High Overhead. As presented in Section 3.5, the complexity of ICO is high. Moreover, the manipulation of graph data structures may be complex, especially for DAGs which have an irregular topology, e.g., LIGO (as shown in Figure 2.12(d)). These facts, as will be demonstrated in Section 3.6.3, may result in a high computational overhead for ICO.

- Failure to capture the optimal schedule in some cases. This may be caused by two limitations, one of which is the decomposition-combination mechanism. ICO combines the building block schedules to construct the entire schedule in the following way: given building blocks B_1 with schedule S_1 and B_2 with S_2, where B_1 has higher priority than B_2. A combined schedule is then formed by executing all non-sinks of B_1 then all non-sinks of B_2 and finally all sinks in an arbitrary order. Supposing that the optimal schedule of $B_1 + B_2$ requires the execution of the first part of sources in B_1 then some sources in B_2, this cannot be captured by the presented combination. The other limitation is that the optimal schedule has to be captured by matching a decomposed block with the predefined CBBBs. However, this should be finite. Once a block actually has an optimal schedule but is not

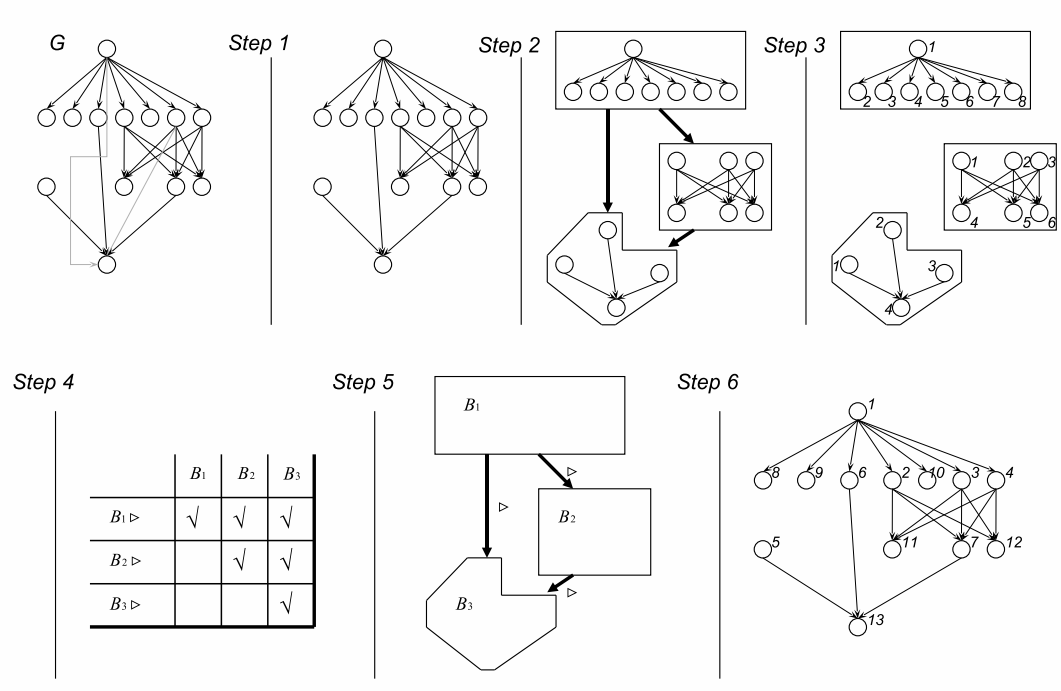


Figure 3.3: Six steps of ICO algorithm [MFR07]

(isomorphic to) a CBBB, the optimal schedule may be overlooked. A DAG example, for which ICO fails to obtain the optimal schedule, is provided in Section 3.6.2.

- Failure to decompose some specific DAGs. The ICO heuristic proposed in [MFR07] is claimed to be able to process all DAGs. However, there is a limitation in its decomposition which may lead to failure when processing some specific DAGs. This is demonstrated by an analysis of a concrete DAG example in Appendix A.

In contrast, PB, a new scheduling heuristic with a different task prioritizing technique is proposed, as presented in Section 3.4, to provide a new just-in-time scheduling solution with higher efficiency, better performance and a wider applicability for DAG applications.

3.4 The Priority-based (PB) Heuristic

The key idea of the PB heuristic is to award each DAG node a numerical priority to describe its ability to produce ready tasks. A node, which can enable more

ready tasks instantly or potentially (expressed by the priority) by its completion, is considered stronger and should be executed preferentially. This principle determines the prioritized sequence of task execution, i.e., the result schedule. In contrast to the ICO heuristic, which prioritizes DAG nodes in a decomposition manner, the PB heuristic calculates the priority of nodes according to their inter-dependencies, for example, in-degrees (the number of parent tasks). This approach has the following advantages:

- Scheduling costs (including both time cost and memory cost) can be saved by manipulating simple numerical values instead of complex topological structures;
- The risk of overlooking the optimal schedule can be reduced by considering the schedule globally instead of locally as ICO does (within building blocks);
- The heuristic can be applied to DAGs with any topological structure.

Four concepts are defined to make up a combined priority to compare the capability of a node to produce the ready tasks. For each node:

Direct Quotient (DQ) depicts the direct contribution a node can make to producing ready tasks. This is defined as the number of tasks which become ready immediately after the completion of the current node. Apparently, to achieve the optimal schedule, the scheduled node must have the highest DQ.

Level Quotient (LQ) depicts a node's topological position in the DAG. This is defined as the maximum length from a node to the exit node. It is assumed that the exit node is at level 0. Then, a node with a maximum length l to the exit node is placed onto level l. Apparently, the entry node is located at the highest level. Given a collection of ready tasks, it is preferable that the node on the highest level is run first unless there is any other node with higher DQ.

Export Quotient (EQ) and Import Quotient (IQ) are recursively defined. EQ is a value only used for two tasks which have the same DQ and LQ to distinguish the priority of tasks. IQ is not used to compare task priority, but to help to calculate the value of EQ for each task. This can be illustrated by the

Input: A DAG application G.
Output: A schedule for G.

1: Compute the initial DQ, LQ, EQ and IQ for each node in G.
2: Add the entry node into the Ready List L.
3: **while** L is not empty **repeat**
4: Schedule the node v in L with the highest priority P, the comparison of task priority is jointly decided by DQ, LQ and EQ following the decision tree shown in Figure 3.5.
5: Remove v from L.
6: Remove v from G.
7: **for** each child x of v in G **do**
8: Decrease the in-degree of x.
9: *UpdatePriority*(x).
10: **endfor**
11: Add new ready tasks into L.
12: **endwhile**

where *UpdatePriority* is a recursive procedure presented as below:
UpdatePriority(*currentNode*)
if the in-degree of *currentNode* is NOT equal to 0
Update the IQ of *currentNode*.
for each parent p of *currentNode* **do**
Update the EQ and DQ of p.
UpdatePriority(p).
endfor
endif

Figure 3.4: The PB Heuristic

following definition. Given a node v

$$IQ_v = \begin{cases} 0 & : \quad v \in S_{sole} \cup S_{ready} \\ (EQ_v + 1)/ID_v & : \quad otherwise \end{cases} \tag{3.3}$$

$$EQ_v = \begin{cases} 0 & : \quad v \in S_{sole} \\ \sum\limits_{u \in Succ(v)} IQ_u & : \quad otherwise \end{cases} \tag{3.4}$$

where ID_v means the in-degree of node v, $Succ(v)$ denotes the set of child tasks of v, S_{ready} means the set of ready nodes and S_{sole} is the set of *sole nodes*. A sole node is a node which can only be executed exclusively, for instance, the entry node and the exit node of a DAG. Therefore, $EQ_{exit\_node} = 0$.

Based on the definitions of DQ, LQ, EQ and IQ, the detail of the PB heuristic is presented in Figure 3.4. It should be noted that LQ does not change as the scheduling heuristic is executed, but DQ, EQ and IQ do. As an example,

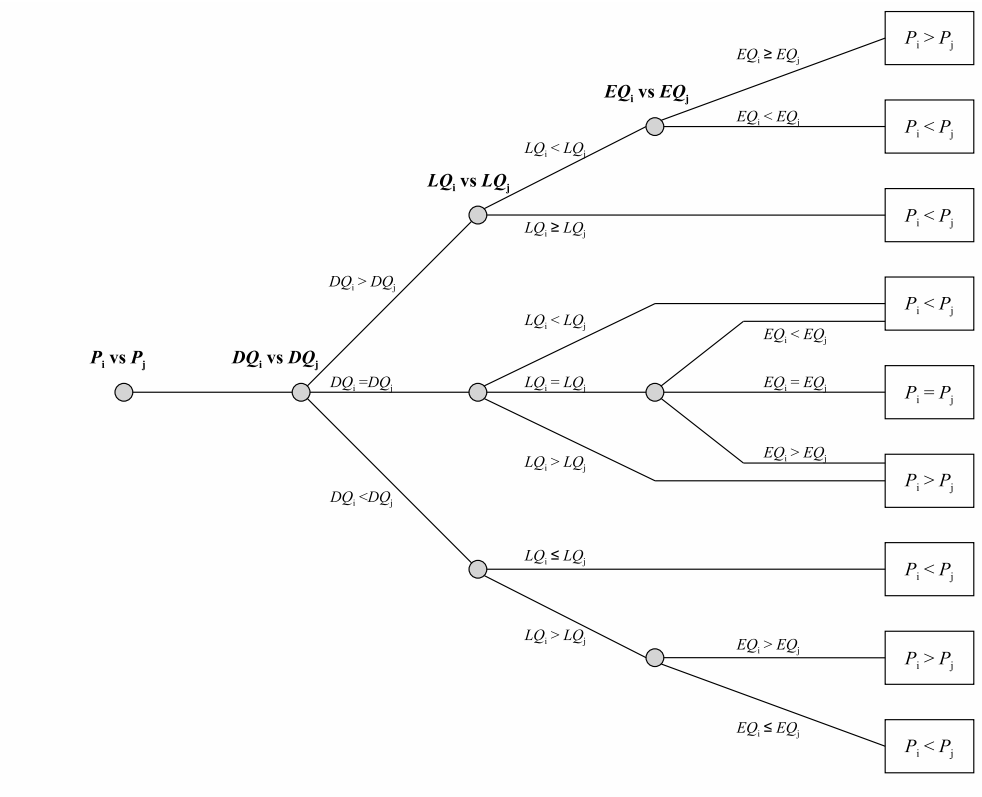


Figure 3.5: The decision tree of comparing task priority

Figures 3.6 and 3.7 illustrate how DQ, EQ and IQ vary as the scheduling goes on.

Given the DAG G on the left-hand side of Figure 3.6, the initial values of DQ, EQ and IQ for each node can easily be calculated from their definition (results in the right half of Figure 3.6). Every pair of numbers in the position of the counterpart node is in the form of $< DQ, EQ, IQ >$. When node 0 has been completed, nodes 1 and 2 become ready. Therefore, the IQ value of nodes 1 and 2 turns to zero, while their DQ and EQ do not change. Subsequently, node 2 is selected because it has a higher DQ than node 1. When node 2 has been completed, DQ and EQ of node 1 are accordingly updated as shown in Figure 3.7.

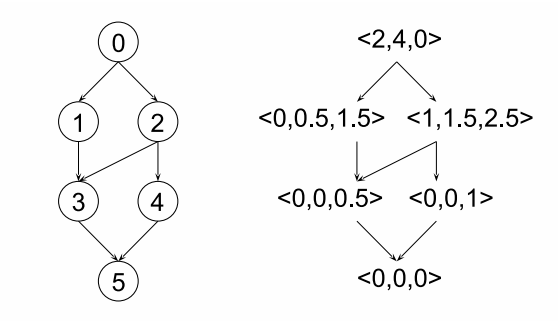


Figure 3.6: An example of computing the initial $< DQ, EQ, IQ >$

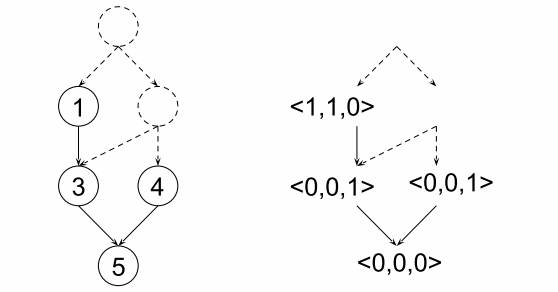


Figure 3.7: Computing the updated $< DQ, EQ, IQ >$ after node 2 is scheduled

3.5 Complexity Analysis of PB and IC-Optimal

Preliminaries Given a DAG G, the number of nodes, denoted by $|V|$, is considered as the input size to analyse the complexity of PB and ICO. Let $|E|$ denote the number of edges. In the case where ICO decompose G into several building blocks, let n denote the number of building blocks comprising G, $|B_i|$ denote the number of nodes in building block i, $|E_i|$ denote the number of edges in building block i, $|S_i|$ denote the number of source nodes in each building blocks, src denote the number of source nodes of DAG G, snk denote the number of sink nodes of DAG G. Then, the direct or indirect connection between these variants and $|V|$ is provided by the following equations:

1. $|E| = c_0|V|^2$, therefore $O(|E|) = O(|V|^2)$
2. $|E| = \sum_{i=0}^{n-1} |E_i|$, especially when $|E_0| = |E_1| = \cdots = |E_{n-1}| = e$, $e = \frac{|E|}{n} = c_1|V|^2$, therefore $O(|E_i|) = O(|V|^2)$
3. $|V| = \sum_{i=0}^{n-1} |S_i| + snk$, especially when $|S_0| = |S_1| = \cdots = |S_{n-1}| = s$,

$s = \frac{|E|-snk}{n}$, therefore $O(|S_i|) = O(|V|^2)$

4. $|V| = \frac{\sum_{i=0}^{n-1} |B_i|+snk+src}{2}$, especially when $|B_0| = |B_1| = \cdots = |B_{n-1}| = b$, $b = \frac{2|V|-snk-src}{n}$, therefore $O(|B_i|) = O(|V|)$

Complexity of PB Based on the algorithm described in Section 3.4 and the above definitions, a complexity analysis of PB is provided in Table 3.1. The sum of the values of the complexity column presents the complexity of PB, which is $O(|V|^3)$.

PB (G)	Complexity	Explanation
Line 1	$O(\|V\|^2)$	traverse every edge of G
Line 2	$O(1)$	only one time
Line 3	$O(\|V\|)$	WHILE loop, one node is scheduled per loop until all complete
Line 4	$O(\|V\| \times \log \|L\|)$	$\|L\|$ is the length of L at each WHILE loop and $0 < \|L\| \leq \|V\|$
Line 5	$O(\|V\|)$	one time per WHILE loop
Line 6	$O(\|V\|)$	one time per WHILE loop
Line 7	$O(\|V\|^2)$	FOR loop, no more than number of edges of G
Line 8	$O(\|V\|^2)$	one time per FOR loop
Line 9	$O(\|V\|^3)$	no more than number of edges of G per WHILE loop
Line 10	$O(\|V\|^2)$	one time per FOR loop
Line 11	$O(\|V\|^2)$	one time per WHILE loop
Line 12	$O(\|V\|^2)$	one time per WHILE loop

Table 3.1: The complexity analysis of PB

Complexity of IC-Optimal In a similar way, the complexity of the ICO heuristic described in [MFR07] is analysed, although this is a more complicated process. The analysis result for each step of ICO is presented in Table 3.2. The sum of the complexity column suggests that the complexity of ICO is $O(|V|^3 \times \log |V|)$.

Summary The worst-case complexity of PB and ICO is analysed, which is determined by a dynamic priority calculation for the former, and DAG decomposing for the latter. Apparently, the complexity analysis shows that ICO has a higher complexity than PB, since DAG decomposing is a more complicated process than

IC-Optimal (G)	Complexity
Step 1	$O(\|V\|^{\alpha})$, where $\alpha = \log_2 7$
Step 2	$O(\|V\|^3 \times \log \|V\|)$
Step 3	$O(\|V\| \times \log \|V\|)$
Step 4 and 5	$O(\|V\|^2)$
Step 6	$O(n \times \log n)$

Table 3.2: The complexity analysis result of ICO

dynamic priority calculation, which is also the crucial difference between PB and ICO.

3.6 Experimental Evaluation

This section compares PB against ICO and two simple, intuitively compelling just-in-time scheduling heuristics, namely FIFO and GREEDY, as defined in Section 3.6.1. The experiment is designed to investigate the following three aspects of a heuristic:

- How quickly can the heuristic schedule a DAG, which is evaluated by the metric of time cost, i.e., the execution time of the heuristic;
- How well can the heuristic maximally generate ready tasks, which is evaluated by the metric of turnout of ready tasks, i.e., a sequential set of the numbers of ready tasks produced at the completion of each task;
- Based on the scheduling result of the heuristic, how quickly can the application be completed in the batch mode as described in Section 3.6.2, which is evaluated by the metric of batched-makespan, i.e., the number of resource batches needed to complete a DAG application.

3.6.1 Competing Heuristics

Descriptions of FIFO and GREEDY are provided as follows, given a DAG G,

FIFO Heuristic

1. FIFO initially schedules G's source nodes and put them into a ready task pool P;

2. When a remote resource R becomes available, FIFO allocates a task t from P to R, while the allocation is enforced in the order of the DAG definition, which can be regarded as 'randomly';

3. When t is completed, it is removed from P, and those of t's children which have become ready are placed into P;

4. Steps 2 and 3 are repeated until all of the nodes of G have been executed.

The GREEDY Heuristic

1. GREEDY initially puts G's source nodes into a prioritized queue Q in non-decreasing order of out-degree, and those nodes of equal out-degree are enqueued in random order;

2. When a remote resource R becomes available, GREEDY allocates the task t at the head of Q to R ;

3. When t is completed, it is removed from P, and GREEDY enqueues those of t's children which have become ready in non-decreasing order of out-degree, and those nodes of equal out-degree are enqueued in random order.

4. Steps 2 and 3 are repeated until all nodes of G are executed;

It should be noted that both FIFO and GREEDY have random steps in producing the order of task execution. This means that their scheduling results are not stable. Therefore, in subsequent experiments, the above-mentioned metrics of these two heuristics were evaluated by the average value over multiple measurements.

3.6.2 The Experimental Setup

Similar to the description in [HRV07], it is assumed that a DAG application is executed in a *batch mode* (a variant of which is studied in [MR05]): A READY pool is maintained to hold the ready tasks for the assignment to task execution requests from resources. These available resources appear batch by batch. As each batch arrives, the ready tasks are allocated by means of one task per resource. If more resources arrive than ready tasks, the unused resources will simply disappear, and can be regarded as utilized by other applications. If fewer resources

arrive than ready tasks, the unallocated tasks will be returned to the pool. It is assumed that all of the tasks allocated at the ith batch will be completed before the $(i+1)$th batch arrives. More concretely, given that a DAG application with n nodes is run by schedule S. When the ith coming batch appears with r_i resources and there are t_i ready tasks in the ready-task pool, then $\min(r_i, t_i)$ tasks will be allocated and executed. Suppose that resource batches arrive constantly until all of the tasks are completed, the number of resource batches required to complete a DAG are counted, and defined as the metric of a *batched-makespan*.

Classification of DAGs There are a various types of DAG applications which can be used to compare the four competing heuristics. As described below, these DAGs are classified into three categories according to the features of their composition methods.

1. *Category A*: Based on the compositions of CBBBs, a method of generating complex DAGs is provided in [HRV07]. The provided method attempts to ensure that all generated DAGs admit so-called IC-optimal schedules, i.e., for all of these DAGs, ICO can obtain optimal schedules. These DAGs are classified into *Category A*. The majority of known applications like LAPLACE [SZ04a], FFT [SZ04a], MONTAGE [BJD+05], LU [BBR02a], STENCIL [BBR02a], FORK-JOIN [BBR02a], DOOLITTLE [BBR02a], and LDM^t [BBR02a] belong to this category. Since the comparisons based on these DAGs obtain similar results, LAPLACE (shown in Figure 3.2) is selected to represent them.

2. *Category B*: For some composites of CBBBs, PB can produce optimal schedules, whereas ICO, FIFO and GREEDY cannot. These DAGs are classified as *Category B*. As shown in Figure 3.8(a), a typical example of this category is a building block composed of the CBBBs W[5,2] and W[2,6] as depicted in Section 3.3. In this example, it can easily be imagined that node 7 must be scheduled before node 0 through 3 to obtain an optimal schedule. This schedule can be captured by PB but not ICO. Such building blocks, varying in source numbers, sink numbers or even topologies can be used to construct more complex DAGs which still fall into Category B.

3. *Category C*: There are still lots of randomly generated DAGs, which may not be composite of CBBBs. For these DAGs, it is likely that none of the

four heuristics can produce the optimal schedule. These DAGs are classified as *Category C*. Random DAGs of Category C are generated by the method described in Appendix B, and an example of this DAG category is provided in Figure 3.8(b).

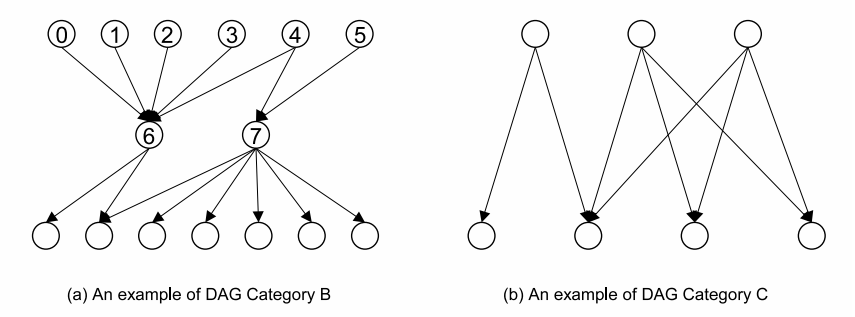


Figure 3.8: Examples for DAG *Category B* (left) and *Category C* (right)

The Time Cost Experiment This experiment was undertaken to investigate how the time cost for each heuristic grows as the number of nodes increases. The investigation covered all DAG categories (or their representative). By specifying the number of nodes, a DAG was generated for each of them, and each of the competing heuristics was executed 50 times for the generated DAG. The average time cost was measured over these 50 executions to make the comparison. The selected numbers of nodes were 25, 100, 400, 900, 2500, 3600 and 6400.

The Turnout of Ready Tasks Experiment This experiment was undertaken to investigate the number of ready tasks produced after each task execution of a DAG. Recalling the definition in Section 3.2, once a heuristic is applied to a DAG G with n nodes and then schedule S is produced, its turnout of ready tasks can be depicted by a n-entry vector $V_S =< NR_S(0), NR_S(1), \cdots, NR_S(n-1) >$. Given two vectors V_S^1 and V_S^2, their comparison results can be classified into the following six cases:

1. $V_S^1 == V_S^2$: for each step i, $NR_S^1(i) = NR_S^2(i)$;
2. $V_S^1 \gg V_S^2$: for each step i, $NR_S^1(i) \geq NR_S^2(i)$ and $\exists$ step j, $NR_S^1(j) \neq NR_S^2(j)$;

3. $V_S^1 \ll V_S^2$: for each step i, $NR_S^1(i) \leq NR_S^2(i)$ and $\exists$ step j, $NR_S^1(j) \neq NR_S^2(j)$;

4. $V_S^1 > V_S^2$: none of case 1, 2 and 3 is satisfied, and $\sum NR_S^1(i) > \sum NR_S^2(i)$;

5. $V_S^1 < V_S^2$: none of case 1, 2 and 3 is satisfied, and $\sum NR_S^1(i) < \sum NR_S^2(i)$;

6. $V_S^1 \ ? \ V_S^2$: none of case 1, 2 and 3 is satisfied, and $\sum NR_S^1(i) = \sum NR_S^2(i)$;

1000 DAGs were generated for each DAG category by means of a random selection of the aforementioned selected numbers of nodes. Each of the competing heuristics was applied to each generated DAG, and its turnout of ready tasks was obtained in the form of vector V_S separately. Four vectors were compared in pair, and the number of times by which a particular case was hit according to the comparison result was accumulated. These numbers can suggest which heuristic delivers the better turnout of ready tasks.

The Batched-makespan Experiment Recalling that it is assumed that DAG applications are executed in the batch mode described in Section 3.6.2, in more detail, the number of resources arriving at each batch is assumed to be a random variable following exponential distribution with a mean value in the set of {2, 4, $\cdots 2^{10}$}. Two DAGs with two different sizes were used for each DAG category. The DAG sizes were around 100 and 900 respectively. In detail, two Laplace DAGs with 100 nodes and 900 nodes were used in Category A; two special cases of CBBB composites, one containing W[2,48] and M[49,2] with a total of 105 nodes, the other containing W[2,448] and M[449,2] with a total of 905 nodes, were adopted for Category B. In terms of the DAGs of Category C, which may be diverse in topology even of the same size, 50 DAGs were generated for a specific size (100 or 900) and average results were obtained. For each generated DAG, every competing heuristic was executed in the batch mode 500 times, and the means of the observed batched-makespan, M_{PB}, M_{ICO}, M_{FIFO} and M_{GREEDY}, were obtained separately.

3.6.3 Experimental Results and Discussion

Time Cost Results

The results of the time cost experiment are shown in Figure 3.6.3. The curves depict the growth of the mean cost of each competing heuristic on each DAG

category. In all cases of different DAG types, the cost of ICO is much more than other heuristics as the DAG size increases due to its higher complexity. In contrast, PB maintains comparable cost with simple heuristics such as FIFO and GREEDY. PB is distinguished from ICO in terms of time costs, not only by a lower complexity but also the manipulation of simple numerical values instead of a complex data structures. Moreover, ICO may exhibit various efficiencies from one DAG category to another due to the structural variety of DAG. In the case of *Category A*, i.e., LAPLACE, the cost of ICO is not too expensive, since the decomposition is easy and the decomposed building blocks can simply be matched to one of the pre-defined CBBBs. However, in the case of *Category C*, the decomposition may be difficult because of the complex task dependencies, and the decomposed building blocks are usually not matchable to any CBBB. Therefore, the cost of ICO increases dramatically in this DAG category, whereas it appears that PB does not suffer from this problem.

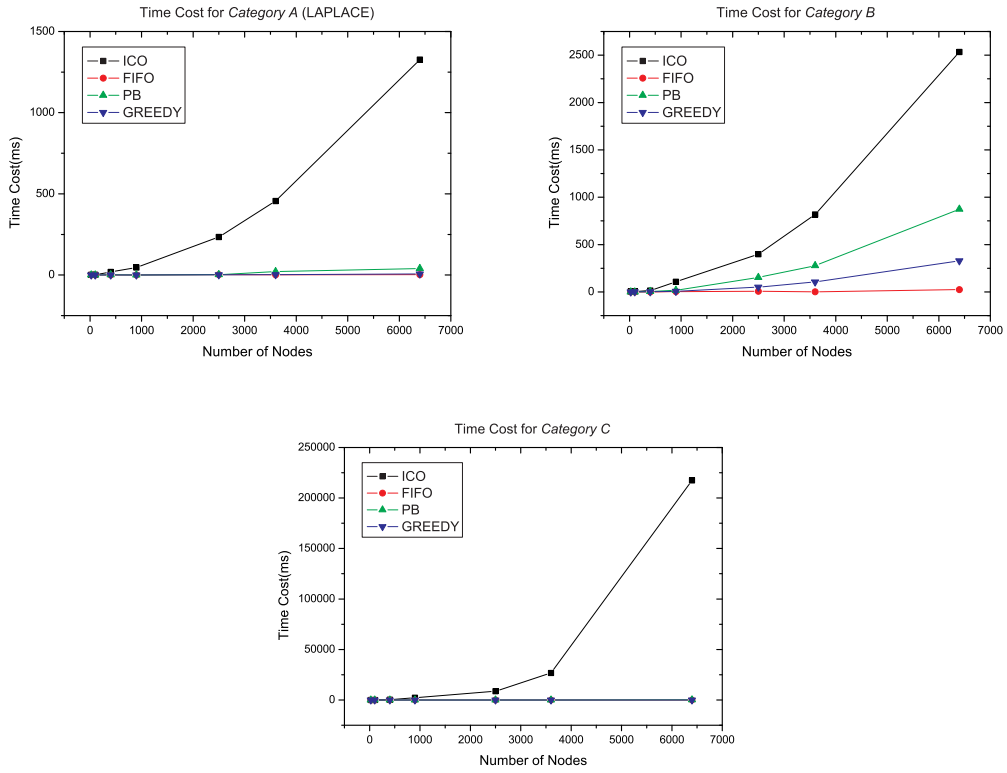


Figure 3.9: Time cost of heuristic for each DAG category

Turnout of Ready Tasks Results

The results of the turnout of ready tasks provided in Figure 3.10 indicate the main difference between the three DAG Categories. For *Category A*, the pair-wise comparison shows that PB and ICO always capture the same optimal schedule, as opposed to FIFO and GREEDY. Therefore, at the completion of each task, PB and ICO obtain the same number of ready tasks, and the number is greater than, or at least equal to, the FIFO and GREEDY obtained. For *Category B*, PB achieves the optimal schedule, while other heuristics fail to do so. Therefore, PB always obtains at least the same, if not more, ready tasks compared to the others. For the last Category, where the optimal schedule can hardly be acquired, PB maintains advantages over other heuristics. It is suggested in this table that, every time PB is compared with other heuristics, the comparison results always fall into the cases which favour PB.

Batched-Makespan Results

In the batched-makespan experiment, various DAG sizes and resource batch sizes were specified to simulate a wide spectrum of dynamic computing circumstances. To compare PB against other competing heuristics, it is assumed that the batched-makespan of PB is 1, and the batched-makespan of other competitors is normalised by computing the ratio of $R_{ICO} = M_{ICO} \div M_{PB}$, $R_{FIFO} = M_{FIFO} \div M_{PB}$, and $R_{GREEDY} = M_{GREEDY} \div M_{PB}$. Therefore, the reported values of other competing heuristics larger than 1.0 represent the advantage of PB. The results in Figure 3.11 show that, in all cases, PB obtains at least equal, if not less, batched-makespan compared with other heuristics. It should be noted that, when the resource supply is sufficient enough for the request of ready tasks, the batched-makespan is simply determined by the length of the critical path of a DAG. In this extreme condition, all heuristics will perform as well as each other. Otherwise, PB can normally improve the batched-makespan by 10-20% compared to FIFO and GREEDY for the DAG *Category A* and *B*. Even in *Category C*, which has the slightest advantage, about 5-10% improvement can still be expected. Moreover, it is indicated that the results of the batched-makespan are in accordance with the turnouts of the ready tasks, i.e., the heuristic which produces a better turnout of ready tasks can usually obtain a better batched-makespan. This verifies the effectiveness of the scheduling goal to maximize the number of ready tasks.

Pair-Wise Comparison on Turnout of Ready Tasks for *Category A*

	Cases of Relationship					
	$V_S^1 == V_S^2$	$V_S^1 >> V_S^2$	$V_S^1 << V_S^2$	$V_S^1 > V_S^2$	$V_S^1 < V_S^2$	$V_S^1 ? V_S^2$
S_1=PB, S_2=ICO	1000	0	0	0	0	0
S_1=PB, S_2=FIFO	0	1000	0	0	0	0
S_1=PB, S_2=GREEDY	0	1000	0	0	0	0
S_1=ICO, S_2=FIFO	0	1000	0	0	0	0
S_1=ICO, S_2=GREEDY	0	1000	0	0	0	0
S_1=FIFO, S_2=GREEDY	0	150	698	49	0	103

Pair-Wise Comparison on Turnout of Ready Tasks for *Category B*

	Cases of Relationship					
	$V_S^1 == V_S^2$	$V_S^1 >> V_S^2$	$V_S^1 << V_S^2$	$V_S^1 > V_S^2$	$V_S^1 < V_S^2$	$V_S^1 ? V_S^2$
S_1=PB, S_2=ICO	0	1000	0	0	0	0
S_1=PB, S_2=FIFO	0	1000	0	0	0	0
S_1=PB, S_2=GREEDY	0	1000	0	0	0	0
S_1=ICO, S_2=FIFO	0	330	0	24	646	0
S_1=ICO, S_2=GREEDY	0	22	0	41	937	0
S_1=FIFO, S_2=GREEDY	0	6	155	273	566	0

Pair-Wise Comparison on Turnout of Ready Tasks for *Category C*

	Cases of Relationship					
	$V_S^1 == V_S^2$	$V_S^1 >> V_S^2$	$V_S^1 << V_S^2$	$V_S^1 > V_S^2$	$V_S^1 < V_S^2$	$V_S^1 ? V_S^2$
S_1=PB, S_2=ICO	0	0	0	1000	0	0
S_1=PB, S_2=FIFO	0	342	0	658	0	0
S_1=PB, S_2=GREEDY	0	93	0	907	0	0
S_1=ICO, S_2=FIFO	0	0	0	1000	0	0
S_1=ICO, S_2=GREEDY	0	0	0	1000	0	0
S_1=FIFO, S_2=GREEDY	0	0	0	393	607	0

Figure 3.10: Pairwise comparing results on turnout of ready tasks

3.7 Closing Remarks

This chapter presents a novel just-in-time scheduling heuristic, the Priority-Based heuristic, aimed at maximizing the number of ready tasks at each step of executing a DAG application. When such a scheduling goal is achieved, the application is expected to perform efficiently on the remote resources, even though their behaviour changes unpredictably over time. With low-cost, PB is applicable to any DAG with an arbitrary structure, therefore it is compatible with any potential DAG application. The experimental evaluation reveals that, compared with ICO, PB achieves significant improvement in the aspects of time cost and applicability, and has a certain advantage in terms of the turnout of ready tasks and

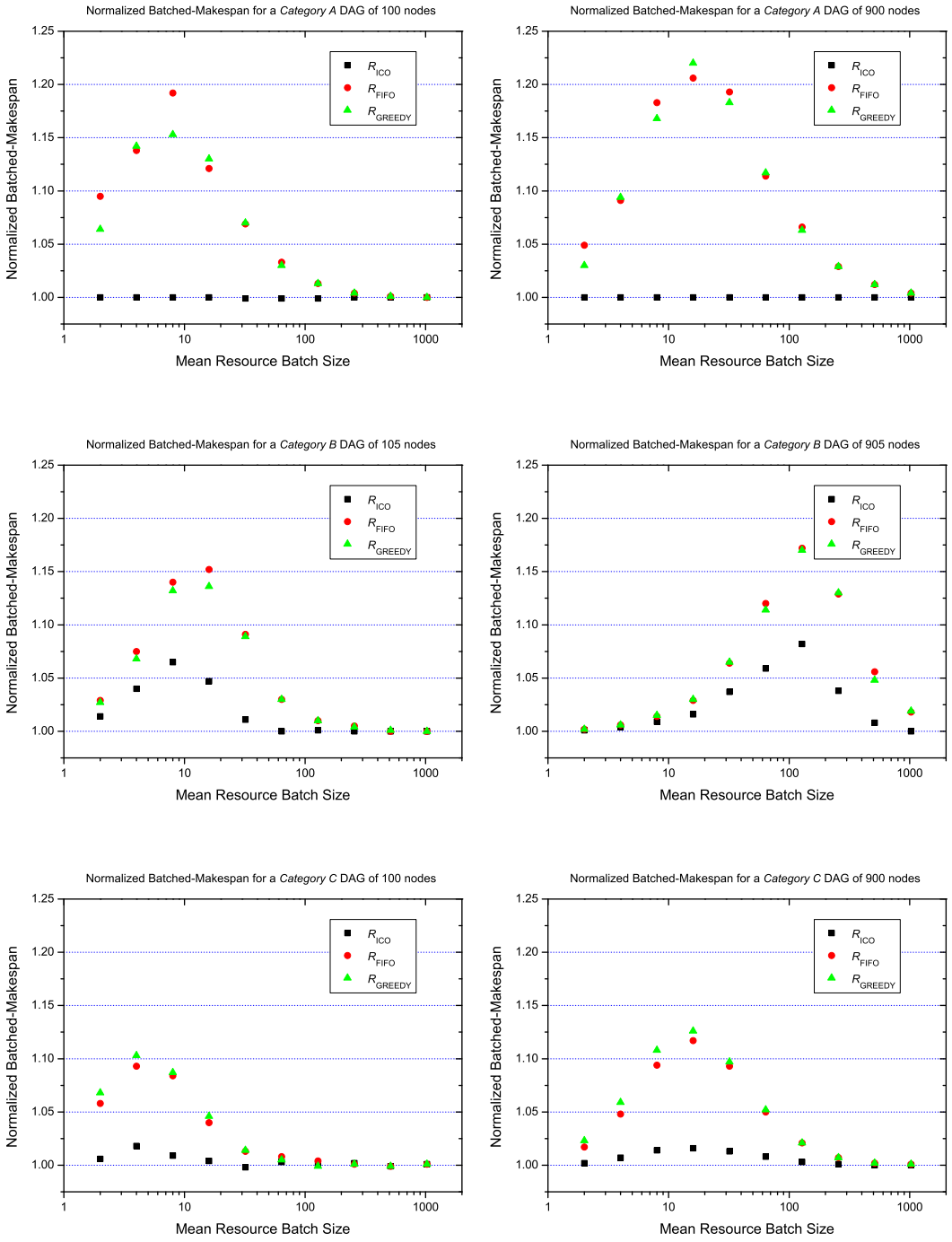


Figure 3.11: Normalized Batched-makespan results ($R_{ICO} = M_{ICO} \div M_{PB}$, $R_{FIFO} = M_{FIFO} \div M_{PB}$, and $R_{GREEDY} = M_{GREEDY} \div M_{PB}$) for specified DAGs

batched-makespans. Compared with FIFO and GREEDY, PB makes notable progress on the latter two metrics with comparable costs. In general, it is suggested that, when considering DAG structure only as the input, PB outperforms other existing just-in-time heuristics when scheduling applications on dynamic remote resources, therefore can become a competitive scheduling solution for the increasingly popular Internet computing, e.g., the Grid.

Either via full-ahead or just-in-time scheduling scheme, the thesis so far have attempted to optimize the makespan of workflow under various uncertainties in the grids of a performance-driven model, where, nevertheless, Quality-of-Service is not considered. The next part of the thesis moves to a different scheduling problem with different scheduling objectives, namely, to address the uncertainties in workflow scheduling in the grids of a QoS-driven model.

Chapter 4

SLA-based Workflow Scheduling

As mentioned in Chapter 1, this thesis consists of two main parts, the first of which, i.e., Chapters 2 and 3, addressed workflow makespan optimization problems under various grid uncertainties in the performance-driven model where no Quality-of-Service (QoS) is considered. The next part, beginning with this chapter, moves on to bypass grid uncertainties with workflow scheduling by using a QoS-driven model, where users need their QoS constraints to be satisfied by grid systems which provide grid services, and the service providers want to maximize their benefits. This is the approach normally followed by market-based grids.

This chapter provides background material on market-based scheduling systems, surveys the current state of the art, and outlines the study's SLA-based workflow scheduling model designed for both guaranteeing users' QoS requirements and maximizing service providers' benefits. The chapter is organized as follows: Section 4.1 compares scheduling in the QoS-driven model with scheduling in the performance-driven model. Section 4.2 provides a survey of a number of existing market-based scheduling studies and identifies the outstanding issues to guarantee QoS for workflow applications under grid uncertainties. Section 4.3 presents the study's model of an SLA-based scheduling system and analyzes the complex SLA-based workflow scheduling problem to solve. Section 4.4 concludes the chapter.

4.1 Background

In the context of Grid computing, many users may need their workflow applications to be completed within a specific period, for example, weather forecasters

may run climate modelling workflow simulation for forecasting [QAY08]. Essentially, what these users require from the Grid system is Quality-of-Service (QoS). Here, the term of QoS refers to a guaranteed level of performance (for example, completing a job in less than 10 minutes) [qos], rather than the achieved service quality. Apparently, from these users' points of view, only when a certain level of QoS is delivered, can a Grid be regarded as an efficient and effective system.

Many of the traditional workflow scheduling approaches are based on the performance-driven model, which is naturally unable to guarantee QoS. In the dynamic case where resources exhibit varying availability, there is obviously no chance to ensure the completion of workflow. Even in a case where a stable set of resources is assumed, the enactment of workflow often relies on queue-based scheduling systems, i.e., a workflow task cannot be run until it reaches the head of the queue. Since the actual task execution time usually differs from the estimation, the completion time of tasks in a queue may approximate to 'whenever' [MSK+04]. As a result, the execution time of the whole workflow may be even more uncertain, due to the task dependencies. Although some efforts have been made to cope with the grid uncertainties (e.g. [JHSN05]), as in the previous chapters of this study, many traditional grid systems still try to complete applications as they come, without any guarantee on their completion time [ES01, ABJ+04, LAH+04, DBG+04, CJSN03, ST04]. Therefore, without a good prediction for the whole system [JFI+07], it is difficult for a user to know, even approximately, when the application can be completed.

To handle the issues stated above, advance reservation of resources has been suggested to allow a user to reserve resources in order to ensure that they will be available for task running within a requested period. However, this approach in its suggested form, (i.e., one task exclusively occupies a precise period on a resource), may have drawbacks due to the unpredictability of task execution time. If the task finishes earlier than the reserved time, the resource will be left idle, which is undesirable for the service provider; if the task finishes later than the reserved time, the QoS guarantee may fail and the user's requirement will not be met. Therefore, more sophisticated approaches must be developed in order to not only satisfy the user's QoS requirements but also hold the resource owner's benefits, even though these two objectives are often contrary.

Currently, there has been an increasingly popular trend for Grid technologies to progress towards a market-based paradigm to strike the trade-off between the

conflicting requirements of users and service providers. In market-based grids, users are allowed to consume services based on their QoS (Quality of Service) requirements and in turn make payment for a successful service provision. In such a paradigm, it is envisaged that there may be conflicting requirements of the user and/or the system, such as application deadline, user's budget and resource owner's profit. Therefore, a major shift in the underlying scheduling technology is needed to deal efficiently with these requirements while still taking the grid uncertainties into account, and SLA-based workflow scheduling may provide a promising solution to achieving this.

Analogous to a contract in the real business world, Service Level Agreements [Hil93] (SLA) play a crucial role in a market-based Grid. An SLA is a bilateral contract between the user and the service provider. Applying SLAs is a mandatory prerequisite for a market-based Grid, since this provides an explicit statement of the expectation and obligation of both sides—the user and the service provider, in their business relationship. SLAs are an important link connecting the flow of processing a user application in a market-based Grid. Firstly, in order to run an application, the user needs to send a request to the Grid system and specify his/her QoS requirements, which are usually expressed as a set of constraints. Next, the QoS requirements need to be agreed upon by the user and the service provider before the application can be executed. Then, if an agreement is reached, the user needs to make advance reservations for his/her application. SLAs act as a material which records all of the agreed QoS constraints and advance reservations, and then guides the execution of the workflow in the right direction to fulfill the QoS requirements of the user. At the end of running the application, if the QoS requirements are met, the user pays the service fees specified in the SLA, and the service providers receive income. Otherwise the service providers pay the penalty specified in the SLA and the user receives compensation. Therefore, such a scheduling scheme in the context of market-based Grid is known as 'SLA-based scheduling'.

Scheduling in the QoS-driven model, namely SLA-based scheduling, is quite different from those in the performance-driven model. Firstly, there are far more issues, such as multiple QoS constraints, advance reservation, and service price, to be considered in the former model as opposed to the latter. Secondly, with an attempt to guarantee that the workflow will be executed within a specific period, there is a big shift in scheduling objectives. In SLA-based scheduling, instead of

pursuing the optimization of the makespan, the scheduler may focus on meeting the user's hard constraints, while maximizing the profit for the service providers. Moreover, there are always multiple workflows requiring QoS guarantee in market-based Grids, and this probably leads to resource competition, and consequently makes the scheduling problem even more complex. At present, although there has been an increasing academic interest on market-based Grids, the study of a SLA-based workflow scheduling is still in its infancy. This provides the main motivation of the work on SLA-based scheduling in this thesis.

4.2 Review of State of The Art

Essentially, the SLA-based scheduling work in this thesis is to build a prototype of a scheduling system for a market-based Grid model. In the recent decade, numerous market-based scheduling and resource management systems have been proposed with various design features, focuses and objectives for a Grid computing platform. To depict the state of the art and distinguish this work from other related work, in this section, a taxonomy and a survey of the published market-based scheduling and resource management works is provided.

4.2.1 Taxonomy

Based on related work [YB06a], which presented a taxonomy of market-based resource management systems for so-called utility-driven cluster computing, several taxonomies were considered for the market-based Grid scheduling systems with a focus on scheduling issues. The taxonomies are considered from five perspectives: *Market Model*, *Resource Model*, *Application Model*, *QoS*, and *Scheduling*.

Market Model Perspective

The taxonomy from the *Market Model* perspective examines how the concepts present in economics are derived and incorporated into the design of the market-based Grid model. This also depicts the features of the underlying infrastructure on which the application scheduling is carried out. In the market model perspective taxonomy, two sub-taxonomies are considered: *economic model* and *benefit focus*, which are illustrated in Figure 4.1.

The *economic model* derived from [YB06a] indicates how resources or services

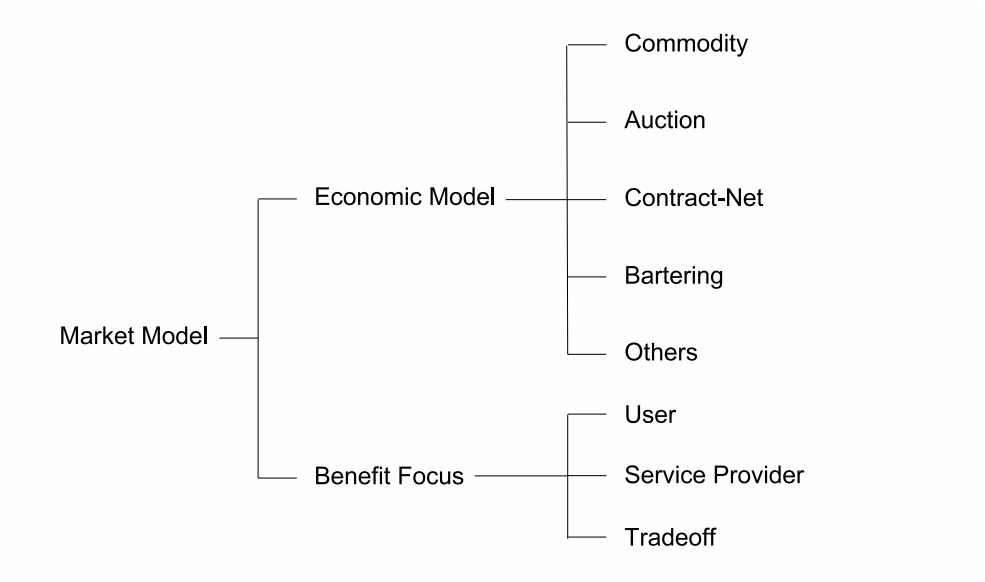


Figure 4.1: The taxonomy from Market Model Perspective

are traded in a market-driven computing environment. A particular economic model is normally determined by the market interaction between users and service providers.

- In the *commodity* market, service providers specify service prices and users pay for the service they consume in terms of the price. There are various pricing policies, which can be flat or variant and depends on different factors e.g., the current supply and demand. The charge to the user can be calculated using various parameters such as usage time and usage quantity.

- In an *auction* market, multiple users submit bids through an auctioneer, who acts as a coordinator and sets the rules of the auction, to negotiate who wins the access to a single service. The negotiation continues until the highest bid is accepted.

- In the *contract-net* market, the user first announces his/her requirements to invite bids from potential service providers. These service providers then evaluate the requirements, and decide to respond with bids or ignore the announcement according to their interests and capabilities. The user who receives the bids thus selects the most suitable service provider and sends a contract which specifies conditions needing acceptance and confirmation for the selected provider to sign.

- In the *bartering* market, there are a group of community members, who can be either users or service providers, sharing resources and services with each other. Thus, a corporative sharing environment is formed.

In addition to the set of common economic models as listed above, a more complete list can be found in [BAGS02].

The *benefit focus* identifies the party for whom the market-based scheduling system aims to achieve benefits. As the name implies, having a *user* benefit focus means that the scheduling system aims to meet the requirements specified by the users, and to optimize their perceived benefits. Similarly, having *service provider* benefit focus means that the profits of the resource owners is the dominant objective of the scheduling system. In contrast, the *trade-off* benefit focus has no particular bias to users or service providers, but instead, aims to strike a trade-off which can be accepted by both parties.

Resource Model Perspective

The taxonomy from the *Resource Model* perspective distinguishes the characteristics of the grid resources which are modelled in scheduling studies. This features the particular computing platform which the market-based scheduling system focuses on and significantly influences the design of the scheduling approaches. The resource model perspective taxonomy consists of four sub-taxonomies as outlined below and shown in Figure 4.2.

Management Control depicts how resources are organized and controlled in Grid systems. The control can be *centralized*, i.e., all resources can be fully controlled and managed by a central grid scheduler; or *decentralized*, namely different groups of resources are controlled by different local managers respectively; or *hierarchical*, which is a mixture of centralized and decentralized where local managers are coordinated by a global manager.

Resource Diversity identifies whether the resources which comprise the computing system are *homogeneous* or *heterogeneous*, namely whether these resources are identical or diverse in one or more aspect (e.g., configuration, performance, load etc.).

Execution Support classifies resource models into two categories, according to how many tasks they support to execute at the same time: *single-programming* means it is assumed that, at most one single task can be executed on a resource

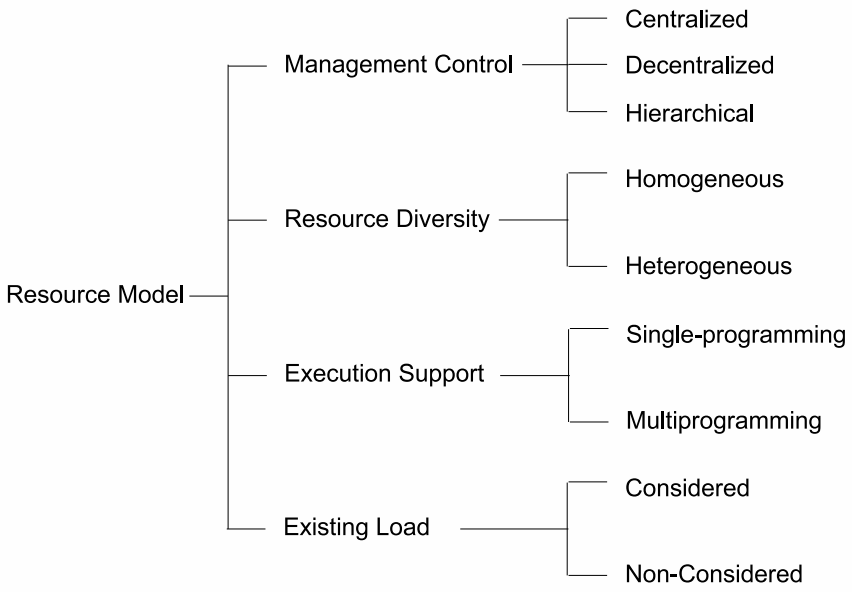


Figure 4.2: The taxonomy from Resource Model Perspective

at the same time, while *multiprogramming* means that, multiple tasks can be concurrently executed on a resource [WHP08].

The *Existing Load* may be *considered* in some market-based scheduling systems or *non-considered* in others. In the former case, scheduling approaches must prevent the new scheduled application from disrupting the previously made scheduling decisions, while, in the latter case, all resources are assumed to be entirely free of load within the period during which the current application is scheduled to run.

Application Model Perspective

The taxonomy from the *Application Model* perspective categorizes attributes of applications which are considered to be scheduled and executed in market-based Grid systems. Market-based scheduling systems must take these attributes into account to ensure that the specific requirement rising from various application types can be successfully fulfilled. As illustrated in Figure 4.3, the application model perspective taxonomy is considered in three facets as follows:

The *Application Composition* depicts how multiple tasks are collected within an application defined by the user. It should be noted that, with a focus on a

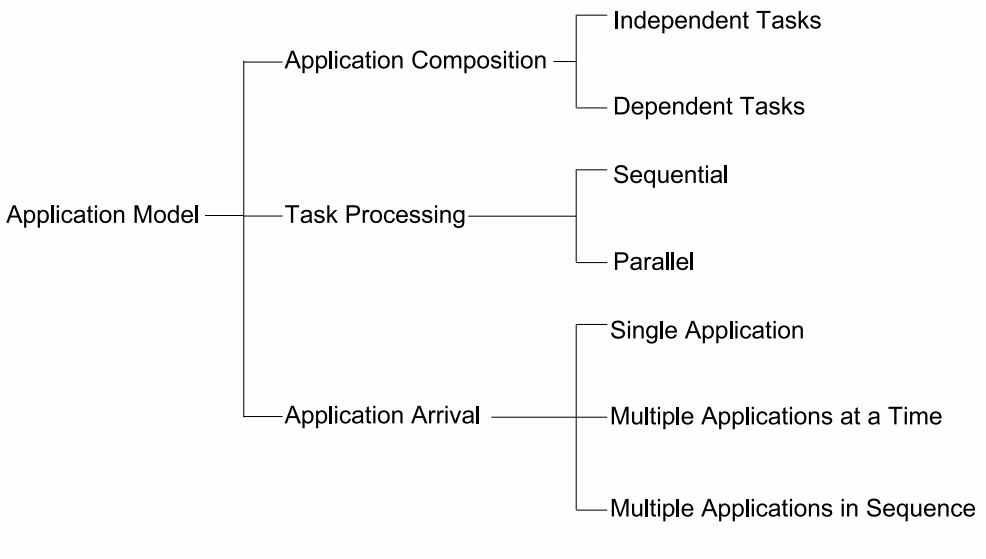


Figure 4.3: The taxonomy from Application Model Perspective

distributed computing platform, only multiple-task applications are considered here. In terms of the connection between the tasks within one application, the application composition can be classified into two categories: *independent tasks* and *dependent tasks*. The former means that all tasks can be independently executed in different resources at the same time, while the latter means that some tasks may depend on input data which can only be computed upon the completion of other tasks.

The *Task Processing* describes the type of processing a task requires. For a *sequential* type of task processing, the task must be sequentially executed in a single processor, while, for a *parallel* type of task processing, the task may require the use of multiple processors.

The *Application Arrival* portrays the scenario in which how many and how the application requests arrive and how they are considered in terms of scheduling. Generally, there are three different scenarios which have appeared in the existing scheduling studies:

- *single application*, i.e., the scheduling system only concentrates on the scheduling process on a single application;
- *multiple applications at a time*, i.e., the scheduling system schedules multiple applications in a batch, which can be regarded as arriving at the same

time;

- *multiple applications in sequence*, namely, the scheduling system investigates scheduling issues over a bunch of applications arriving sequentially within a certain period.

QoS Perspective

The taxonomy from a *QoS* perspective specifies how the QoS related parameters and policies are set in a market-based scheduling system, which also influences the design of a specific scheduling approach. As shown in Figure 4.4, the QoS perspective taxonomy proposed here consists of three aspects: *QoS Attribute*, *QoS Specification* and *QoS Guarantee*.

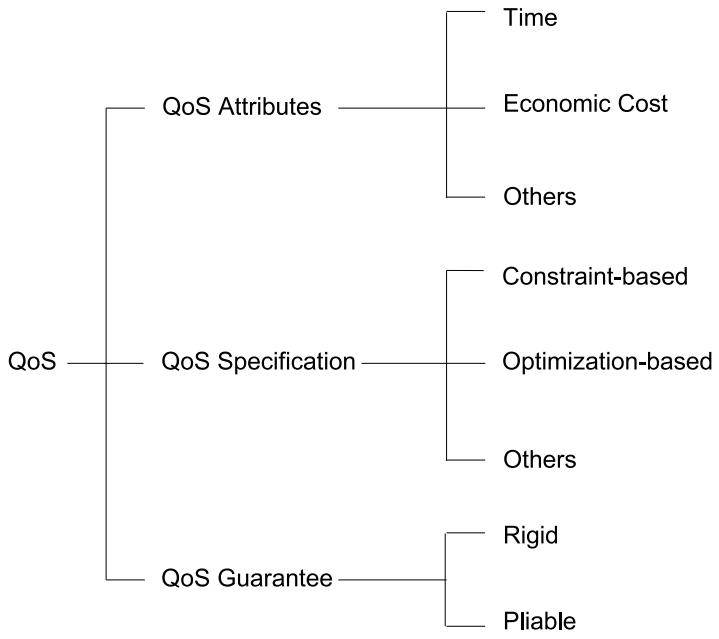


Figure 4.4: The taxonomy from QoS Perspective

The *QoS Attribute* describes the kind of requirements that users require the system to deliver. The attribute can be *time*, *economic cost* or some other metrics (e.g., reliability, security etc.) which are not as common as the first two mentioned. A typical example of a *time* QoS attribute is the deadline required by the user for the application to be completed. The *economic cost* attribute

is normally monetary and reflects the budget the user is willing to pay for the successful running of the application.

The *QoS Specification* depicts how users perceive satisfaction with the QoS attributes concerned. A QoS specification can be *Constraint-based*, *Optimization-based* or some other form. The constraint-based QoS specification normally defines a range of value for a particular QoS metric, and the success or failure of the service provision depends on whether the delivered QoS result falls within the defined range or not. The optimization-based QoS specification expresses the user's desire to optimize a particular QoS metric.

The *QoS Guarantee* distinguishes how the scheduling system reacts to a recognized breach of the user's QoS requirements. The *rigid* QoS guarantee is usually supported in the case of a constraint-based QoS specification, in which the scheduling system stop scheduling the rest of the application once it is detected that the QoS constraint has been violated and the service providers receive no reward from a failed service provision. In contrast, a *pliable* QoS guarantee ensures that the application will be accomplished, although a certain number of penalties, as pre-specified in the contract, will be applied to the service providers.

Scheduling Perspective

The *Scheduling* perspective taxonomy analyzes the assumptions and designs adopted in various scheduling approaches for different market-based scheduling systems. Four sub-taxonomies are considered in the scheduling perspective taxonomy, as shown in Figure 4.5.

The *Prediction Accuracy* determines whether or not the inherent uncertainty of performance prediction is considered in the scheduling study. The *accurate* represents the fact that the actual task performance is assumed to be always the same as its estimate, while the *inaccurate* means that the prediction error is modelled and taken into account in the design of the scheduling approach.

The *Planning Scheme* classifies scheduling approaches into two main groups in terms of the moment when the scheduling decision is made. The *full-ahead* planning denotes the fact that the whole application is scheduled before its execution starts, while *just-in-time* scheduling makes scheduling decisions for each individual task only when the task is ready to start.

The *Advance Reservation* distinguishes whether or not the advance reservation mechanism is *supported* or *non-supported* in the design of the market-based

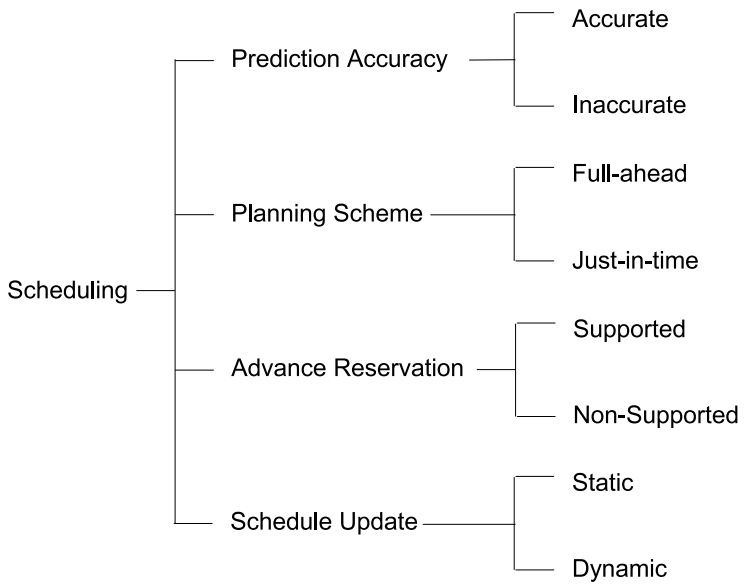


Figure 4.5: The taxonomy from Scheduling Perspective

scheduling system.

The *Schedule Update* determines whether or not the scheduling result of a task, which includes the assignment to a resource and the scheduled start time, is *static* (i.e., unable to be modified) or *dynamic* (i.e., adaptive to run-time changes).

4.2.2 Survey

This section provides a survey covering a selective collection of the market-based scheduling and resource management systems developed for Grid computing platforms. The selected scheduling systems include: Faucets [KKP+04], Nimrod [BA09], VGE [BBES05], Gridbus [YB09], SLA-based Scheduling Heuristics (SLA-BSH, hereafter) [SY08], GridFlow [CJSN03] and SLA-aware Execution Framework (SLA-AEF, hereafter) [Qua06a]. Some published works which focus on market mechanisms for resource management rather than scheduling issues, such as RESERVOIR [RBL+09], SweGrid [GEJ+08], MOSIX [ASL+08], Bellagio and Mirage [ABC+09], Tycoon [SLAO06], Catallaxy-based [AAE+06, ERS+05], CATNET [ERA+03], G-commerce [WPBB01], Gridmarket [CYL04], and OCEAN [PHP+03], are omitted. Using the taxonomies presented in the previous section, the survey is summarized in five tables respectively: the Market

System/Paper	Economic Model	Benefit Focus
Faucets	Contract-Net, Bartering	Service Provider
Nimrod/G	Commodity	User
VGE	Commodity	User
Gridbus	Commodity	User
SLA-BSH	Commodity	Service Provider
GridFlow	Bartering	User
SLA-AEF	Commodity	User

Table 4.1: Survey using the Market Model perspective taxonomy

System/Paper	Management Control	Resource Diversity	Execution Support	Existing Load
Faucets	centralized	NA	Multiprogramming	Considered
Nimrod/G	Decentralized	Heterogeneous	Multiprogramming	Considered
VGE	Decentralized	Heterogeneous	Single-programming	Non-Considered
Gridbus	Decentralized	Heterogeneous	Single-programming	Considered
SLA-BSH	Decentralized	Homogeneous	Multi-programming	Considered
GridFlow	Hierarchical	Heterogeneous	Multi-programming	Considered
SLA-AEF	Decentralized	Heterogeneous	Multi-programming	Considered

Table 4.2: Survey using the Resource Model perspective taxonomy

Model Perspective (Table 4.1); the Resource Model Perspective (Table 4.2); the Application Mode Perspective (Table 4.3); the QoS Perspective (Table 4.4); and the Scheduling Perspective (Table 4.5). Similar to [YB06a], the 'NA' keyword is used in the tables to represent the fact that either the specified sub-taxonomy is not addressed by the particular scheduling system or there is insufficient information from the references to determine. Following these summarized tables, the details of each selected market-based scheduling system are briefly presented.

Faucets

Faucets is a framework proposed to target a computational grid. The framework supports a market-driven Compute Server [KKP+04] selection. Two of the main aims of Faucets is to achieve (i) user friendliness, which frees users from manually

System/Paper	Application Composition	Task Processing	Application Scenario
Faucets	NA	Parallel	Multiple Applications in Sequence
Nimrod/G	Independent Tasks	Sequential	Multiple Applications at a Time
VGE	Dependent Tasks	Sequential	Single Application
Gridbus	Dependent Tasks	Sequential	Single Application
SLA-BSH	Independent Tasks	Parallel	Multiple Applications at a Time
GridFlow	Dependent Tasks	Parallel	Multiple Applications in Sequence
SLA-AEF	Dependent Tasks	Parallel	Single Application

Table 4.3: Survey using the Application Model perspective taxonomy

System/Paper	QoS Attributes	QoS Specification	QoS Guarantee
Faucets	Time	Constraint-based	Pliable
Nimrod/G	Time,Cost	Optimization-based	Rigid
VGE	Time,Cost	Optimization-based	NA
Gridbus	Time,Cost	Optimization-based	Pliable
SLA-BSH	Time,Cost	Constraint-based	Pliable
GridFlow	Time	Optimization-based	Pliable
SLA-AEF	Time,Cost	Optimization-based	NA

Table 4.4: Survey using the QoS perspective taxonomy

System/Paper	Prediction Accuracy	Planning Scheme	Advance Reservation	Schedule Update
Faucets	Accurate	Just-in-time	NA	Dynamic
Nimrod/G	Accurate	Just-in-time	Non-Supported	Dynamic
VGE	Inaccurate	Full-ahead, Just-in-time	Supported	NA
Gridbus	Inaccurate	Full-ahead	Supported	Dynamic
SLA-BSH	Accurate	Full-ahead	Supported	Dynamic
GridFlow	Inaccurate	Full-ahead, Just-in-time	NA	Dynamic
SLA-AEF	Accurate	Full-ahead	NA	NA

Table 4.5: Survey using the Scheduling perspective taxonomy

discovering the best resources and monitoring the job execution, and (ii) improved utilization of the Compute Server.

Aiming to maximize resource utilization, Faucets adopts a contract-net economic model to facilitate server selection. In the framework, every user announces a QoS contract for his/her job, and then Compute Servers compete for the job by submitting bids via a bidding and evaluation system. Finally, the user chooses the most suitable Compute Server when all of the bids are collected. It is claimed in [KKP+04] that the Faucets architecture is also suitable for establishing a bartering economic model.

Faucets uses a Central Faucets Server (FS) to mange and control grid resources (i.e., Compute Server). Each resource has multiple processors and supports several applications to be run simultaneously as long as the number of required processors does not exceed the number of processors this resource has. The existing load of the resource must be analyzed in Faucets in order to avoid the situation of resource processors being left idle. Faucets does not mention whether its resources should be homogeneous or heterogeneous.

The application modelled in Faucets is a kind of adaptive job, which is a parallel program which can change the number of required processors in runtime. The fewer processors used, the longer it takes to complete the job. It is not clear how an adaptive job is composed, but it is certain that parallel processing is

needed to execute the job. The characteristics of adaptive jobs enable the design of a smart job scheduler, which is triggered when a new job arrives in the system to analyze the job's deadline and processor requirements and then to decide if the job can be accepted or not. This implies that the scheduler focuses on the scenario of 'multiple applications arriving in sequence'.

In terms of QoS attributes, users in Faucets require their application to be completed before a specified deadline. In the state described in [KKP+04], Faucets adopts a pliable QoS guarantee. In order to maximize resource utilization, Faucets may shrink the allocated processor number of the existing jobs to cater to the requirement of a new incoming job when there are insufficient available processors to satisfy the new job. This may result in a delay of the existing jobs and a loss of profit.

In Faucets, it is the users who specify the amount of time needed to complete the job. Thus, the uncertainty of task execution time prediction is not taken into account. The scheduling decision is made in a just-in-time manner, while advance reservation is not mentioned. The schedule update is indicated to be dynamic, since the allocated processor numbers of the existing jobs are adaptive in runtime.

Nimrod/G

Nimrod/G is an extended work of Nimrod [ASGH95] in order to support the execution of parameter sweep applications in Grid environments. The design of Nimrod/G is based on a commodity market model. In Nimrod, there is a broker acting on behalf of each user. The broker obtains service prices, which may vary from one application to another, depending on the user's QoS constraints and the resources meeting these constraints. Nimrod/G has its benefit focused on users and aims to optimize users' QoS requirements.

Each broker in Nimrod/G is respectively associated with a scheduler, which works independently for resource discovery, resource trading, resource selection and job assignment. Therefore, the management control of resources in Nimrod/G can be viewed as being decentralized. Nimrod/G targets heterogeneous resources each of which allows multiple tasks to be run on it simultaneously. Existing loads of resource are profiled and analyzed by a dispatcher and a set of actuators to perceive the ability of the resource and assign jobs accordingly.

Nimrod/G supports parameter sweep applications each of which consists of

multiple independent tasks with different parameter values which can be executed sequentially on a processor. Multiple applications are considered in a batch in the scheduling process of Nimrod/G [BGA00].

Users in Nimrod/G specify deadline and budget QoS constraints for the application to run, and specify whether they would like one or both of the metrics to be optimized. Therefore, both constraint-based and optimization-based QoS specifications are considered in Nimrod/G. The QoS guarantee is rigid, since Nimrod/G stops allocating resources to the remaining tasks of a job once the job's deadline or budget constraints are violated.

Uncertainty of task execution time prediction is not taken into account in the scheduling of Nimrod/G, nor is Advance Reservation. Although the scheduling adviser in Nimrod/G creates a schedule for an application based on users' requirements before the application begins to be executed, resources may be discovered and allocated progressively for the application depending on the perspective to achieve the user's QoS constraints.

SLA-based Scheduling Heuristics

SLA-based Scheduling Heuristics is an EPSRC-funded research project which aims to provide different level of service in Grid environments by forging a Service Level Agreement between different parties in Grids, such as user, broker and service provider.

Based on a commodity market model, the research presented in [YS06] evaluates the effectiveness of several simple scheduling heuristics with various pricing settings with the aim of maximizing resource utilization and income.

The research targets on autonomous and homogeneous resources, which have multiple processors to support multi-programming execution. Therefore, a decentralized resource management control is modelled. Moreover, the existing load of resource must be considered in the scheduling to maximize resource utilization.

The research investigates the process of scheduling multiple applications arriving at a time. Each application consists of multiple independent tasks which require parallel processing.

For each application, the user specifies deadline and budget constraints and signs an SLA with the broker who, in turn, negotiates with service providers to find suitable services to satisfy the user's QoS requirements. A breach of deadline constraint will result in a loss of profit, while the application will be completed.

That is to say, a pliable QoS guarantee is provided.

A full-ahead planning scheme is adopted in the research with the assumption that the task execution time estimated by the user is the same as the actual duration. A form of dynamic advance reservation is supported in the research, which allows rescheduling at run-time.

VGE

Vienna Grid Environment (VGE), currently utilized in the context of EU Project GEMSS, is a service-oriented Grid infrastructure based on standard Web Service technologies [BBES04].

A pricing model is considered in VGE, therefore a commodity market is established. Users of VGE may consider multiple QoS constraints each of which is associated with a weight ranging between 0 and 1. The weights of all constraints add up 1 and for each constraint, the result parameter is normalized by multiplying the weight. The aim of the scheduling in VGE is to maximize the user's objective function, which is defined as being the sum of the normalized parameters of all constraints.

In VGE, resources which provide various services may be decentralizedly managed and controlled by different service providers. The services of VGE are installed on heterogeneous resources, each of which is viewed as a single-programming. The existing load of resource is not considered in VGE.

VGE supports workflows which are comprised of multiple dependent tasks. The tasks require sequential processing in a processor. In the scheduling scenario of VGE, only one single application is considered.

Users of VGE usually have a QoS requirement on the criterion of begin time, end time, and budget. Optimized results on these criterion are preferred. Therefore, pliable QoS guarantee is provided by VGE.

Two alternative planning approaches, namely static planning and dynamic planning [BBES05, ZBN+04], are considered in VGE. The selection of planning approach depends on whether or not the meta data required for performance prediction is statically known. Running evaluation on real machines, VGE admits inaccuracy in performance prediction. Nevertheless, there is insufficient information to know the exact detail about the schedule update in VGE.

Gridbus

In the Gridbus Project developed by GRIDS Laboratory in the University of Melbourne, a series of research into economy-based application scheduling on global Grids has been carried out [YB04, YB06b, YB07, YB09, TKB07]. These scheduling studies are based on an infrastructure with commodity market model and aims to maximize the user's utility.

Decentralized resource management and control is implemented in the Gridbus workflow enactment engine based on a tuple-space model [YB04]. In Gridbus scheduling studies, grid resources are modelled to be heterogeneous, single-programming and the existing load of resource is considered.

The scheduling studies in Gridbus focus on workflow applications consisting of multiple dependent tasks. These tasks require sequential processing. In the evaluation of the proposed scheduling algorithms, normally one single application is considered in Gridbus.

The workflows submitted to Gridbus are usually associated with budget and deadline constraints. The workflow scheduling for Gridbus focus on the optimization of one or two of the QoS constraints. Thus, a pliable QoS guarantee is provided.

Many full-ahead evolutionary-based planning heuristics have been proposed to deliver users' QoS requirements in Gridbus [YB06b, YB07, TKB07] to resolve the multi-objective optimization problem in the context of utility computing [Be09]. In these heuristics, inaccuracy of performance prediction, advance reservation and rescheduling are taken into account.

GridFlow

GridFlow [CJSN03] is a Grid workflow management system developed by the High Performance Systems Group at the University of Warwick. This system is built on the top of the PACE prediction toolkit [NKP$^+$00], TITAN local resource manager [SCT$^+$02] and ARMS (an agent-based resource management system [CJS$^+$02]) which uses the A4 (Agile Architecture and Autonomous Agents) methodology [CST$^+$02].

The economic model of GridFlow can be regarded as bartering, as monetary price for resource usage is not considered while multiple agents, each of which plays as a representative of local grid, are coordinated to optimize workflow execution time.

A hierarchical structure is adopted in GridFlow where simulation, execution and monitoring are provided at the global grid level which work on top of ARMS system. At each local grid, sub-workflow scheduling [CJSN03] and conflict management are processed on top of TITAN and PACE. Here, sub-workflow is a flow of closely related tasks that can be executed in a predefined sequence on resources of a local grid. The resources considered in GridFlow are heterogeneous and multi-programming. The existing load is taken into account. Only MPI and PVM applications are considered in GridFlow, where multiple applications can be submitted via the GridFlow User Portal.

GridFlow focuses on makespan optimization for workflows. Thus the QoS specification is optimization based and the QoS guarantee is pliable. A fuzzy timing technique is applied in GridFlow to address the inaccuracy of task execution time prediction. Full-ahead planning is adopted in the global grid level and just-in-time scheme is used in the local resource scheduling. It is not clear whether advance reservation is supported in GridFlow, while rescheduling is applied when large delays of some sub-workflows occur.

SLA-aware Execution Frame

The SLA-aware Execution Frame is a PhD project [Qua06a] which aims to support SLA for a workflow in the Grid environment. The work focuses on supporting the execution of a workflow on reserved Grid resources within the scope of a business contract, and provides a mapping mechanism, an SLA negotiation protocol, and an error recovery mechanism for workflows within an SLA context.

With determined prices for various services, the SLA-based Execution Frame adopts a commodity market model and focuses benefit on minimizing the total economic cost for users.

In the SLA-aware Execution Frame, decentralized resource management and control are considered. Heterogeneous resources supporting multi-programming and belonging to different resource management systems are modelled. Existing loads of resource are taken into account to cope with the competition of a limited number of processors in a resource.

The SLA-aware Execution Frame considers workflow applications consisting of sub-jobs, which can be sequential or parallel programs. In the scheduling evaluations, only a single workflow scenario is considered.

Users in an SLA-aware Execution Frame is assumed to require constraint-based time requirement and optimization-based cost requirement, while it is not certain whether the QoS guarantee is delivered in a rigid or a pliable fashion.

In the context of the SLA-aware Execution Frame, a full-ahead mapping mechanism is implemented to satisfy the specific user's runtime requirement and to optimize the cost. The mapping mechanism includes three sub-algorithms. L-Tabu finds cost optimal mapping solution for light workflow in which the amount of data to be transferred among sub-jobs is small. H-Map finds a cost optimal mapping solution for heavy workflows in which the amount of data to be transferred among sub-jobs is large. w-Tabu finds the runtime optimal solution for both cases of workflows. The uncertainty of performance prediction is not considered in these scheduling studies. The SLA-aware Execution Frame does not talk about issues of advance reservation and schedule update.

4.2.3 Summary

It can be seen from the survey that the majority of the existing market-based scheduling systems consider applications consisting of independent tasks and/or mainly aim at optimizing user-specific objectives, e.g., to complete the application as quickly as possible, to minimize the user's economic costs etc. Among the surveyed systems, Nimrod/G, VGE, Gridbus, GridFlow and the SLA-aware Execution Frame mainly concentrate on maximizing the user's benefits; although the research of SLA-based Scheduling Heuristics aims to maximize resource utilization and income, only applications consisting of independent tasks are considered. Therefore, there is lack of research efforts focusing on both guaranteeing hard QoS constraints for workflow applications and maximizing service providers' profits. Moreover, there is also a lack of investigation into how to tackle the grid uncertainties, which affect the performance of a market-based scheduling system. This thesis proposes an SLA-based scheduling system to fill these gaps.

4.3 A Model of An SLA-based Workflow Scheduling System

During the recent years, research into market-based Grid models, especially in terms of workflow scheduling and resource management, has led to various designs

and implementations based on inspiration from different business models. With specific purposes and focuses, these designs may vary significantly in complexity and functionality. In this section, the topology, negotiation, pricing and event flow was modelled for a market-based Grid with a focus on an SLA-based workflow scheduling which aims to both meet the user's QoS requirements and maximize the resource owner's profits.

4.3.1 Involved Entities and Topology

In a market-based Grid, there are service providers who own the resources and provide them to satisfy the user's particular demand, and there are users who need to use the services available from these resources. In the case of running workflow applications, allowing users to communicate directly with service providers may lead to great difficulties for users to handle sophisticated works, e.g., discovering desired services, monitoring the running process and reacting to running errors etc. To hide these complexities from users, it is necessary to introduce a broker to act as an intermediary between the users and the service providers. Therefore, a total of three entities is considered in this model: *user*, *broker*, and *service provider*.

A **user** is the consumer of the provided services. To send an application running request to the system, a user needs to submit the workflow application he/she wants to run, specify his/her QoS requirements (here, the focus is on execution time metric, i.e., the application must be completed before a certain deadline), and a budget is set as the maximum payment he/she wants to spend. If the request is declined, the user receives a rejection; otherwise, the user is given an offer of a Service Level Agreement (SLA) and waits for the running results of his/her application.

A **broker** acts as a virtual service provider to the user and a virtual user to service providers. The broker does not actually possess any service. Therefore, it cannot directly schedule the tasks to the services. Instead, it plans the submitted application based on the retrieved information about services. If the planning result shows that the consignment from the user is infeasible (which means that the estimated completion time exceeds the deadline and/or the total running cost turns out to be more than the budget in this thesis), the user's request is rejected. Otherwise, the broker signs an SLA with the user, reserves tasks with the planned service providers and agrees independent SLAs with the providers.

A **service provider** is also called a **Local Resource Manager (LRM)** who owns resources and provides particular services available from its resources. The information of these services is registered in a service repository to be retrieved by the broker or published to users. A service provider respond to the enquiries from the broker about the availability of a requested time slot of its resources, which helps to make job planning decision. A service provider also processes advance reservation requests from the broker and locally schedules the tasks successfully reserved into resources for execution.

This thesis assumes a centralized topology in which all workflow applications are submitted through a broker. For each application, a user agrees an SLA with the broker, and in turn, the broker agrees an SLA for each task with an LRM who owns the resource on which the task is reserved. Figure 4.6 illustrates the topology.

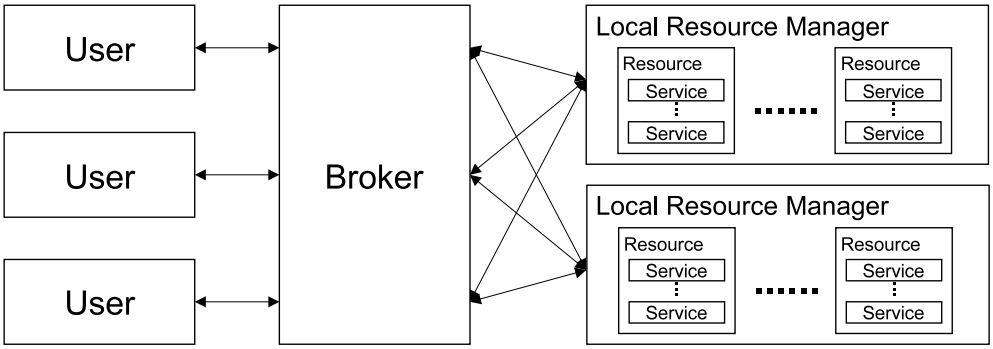


Figure 4.6: The centralized topology of market-based model

4.3.2 Service Provision

In the modelled market-based Grid, there are many heterogeneous resources providing various services. It is assumed that all of these resources are fully connected, and the transmission rate, i.e., the time needed to communicate per unit of data between resources, is predictable. The resources are respectively under the control of different administrative domains. The resources of each domain are managed by a administrative software, i.e., Local Resource Manager (LRM).

The provided services belong to diverse service types, and for each service type, one service instance (service, for short) is offered in each resource. Each

service has its own configuration, including the unique identifier, the service type, the service capacity and the service price. Different services on different resources possess diverse processing capabilities and are assigned different prices. Normally, it is assumed that the service hosted at the more powerful resource has the higher capability and charges the higher price.

Advance reservation [Mac03] is supported in each resource to ensure that a task can be executed within a specific period of time, even though the Grid resources are non-dedicated. When a task is agreed to be executed on a resource at some future point, a corresponding advance reservation is generated and kept with the resource. That is to say, a specific time span is reserved for the resource. It is commonly assumed that a resource can execute only one task at a time without considering pre-emption [ZS06a], therefore the reserved time spans for the same resource are not allowed to overlap.

4.3.3 Application and Constraints

Rather than applications consisting of independent tasks, workflow is the focus of the model of application. As introduced in Chapter 1, workflow is popularly used to represent the applications derived from problems in scientific or industrial fields, such as bioinformatics and astronomy. These applications normally consist of multiple interdependent tasks, and these tasks are executed based on their control or data dependencies, i.e., a task cannot start execution until it gets all of the input data from the tasks it depends on. Like many scheduling studies [SZ04a, SZ04b, MKK+05, BJD+05, WPF05, WPF05, WPF06, YB06b, CJSZ08, BCD+08, YBR08, WPPF08], our work particularly considers the workflows that can be represented by Directed Acyclic Graphs (DAGs). In each DAG, each node denotes a task, and each directed edge represents the dependency between two interdependent tasks. If there is a directed edge from node i to node j, node i is the parent node and j is the child node. The nodes without parent nodes are called entry nodes, and the nodes without child nodes are called exit nodes. Without losing generality, it is considered that a DAG has a single entry node and a single exit node.

To use the services provided by LRMs, a user needs to specify a set of parameters for each DAG task, including the *service type* the task wants to use and the *service size* which quantifies the computational demand of the task to run. Combined with the information of running that type of service on a particular

resource, these two parameters can be used to estimate the execution time of the task on the resource. In addition, the sizes of data transmitted between interdependent tasks also need to be specified to estimate the transmission latency.

When submitting an application to a market-based Grid, the user also needs to specify QoS constraints. Here, QoS metrics are considered on two aspects: *time* and *cost*. For the time constraints, the user specifies the earliest time when the application can start running and the latest time when the application has to be completed, i.e., the deadline. Regarding cost constraints, the user sets a budget as the maximum amount he/she wants to pay to run the application. To meet the user's constraints, the system must offer an SLA with a cost below the budget and complete the application before the deadline.

In more details, a suitable SLA for users should explicitly describe the client's (i.e., user's) QoS constraints as well as the goal of the server. Some works on the standardization of the list of basic components which should be contained in an SLA have been carried out, e.g., WS-Agreement [WSA]. Focusing on the aforementioned specific QoS constraints for workflows, the following guaranteed terms of a SLA are considered in this study:

- Earliest start time of workflow, Stt
- Latest finish time of workflow, Ddl
- Total Charge, c^{tot}

Figure 4.7 provides a graphical representation of the time constraint of a workflow w. Supposing there is a simple workflow comprising of 4 tasks running on 2 resources. In terms of task dependencies, task 1 and 2 depends on task 0, task 3 depends on task 1 and 2, and task 1 and 2 are independent of each other. The time line in the figure denotes wall clock time, referenced by the planner when mapping workflow tasks to resources. Stt and Ddl are values in wall clock time units and indicate the specific period within which the workflow must be executed. t_{ex} represents the execution duration of the whole workflow. To meet the time constraint, the time starting workflow (task 0, equivalently) must be later than Stt_w and the time finishing workflow (task 3, equivalently) must be earlier than Ddl_w. Moreover, the task dependencies must be respected during execution.

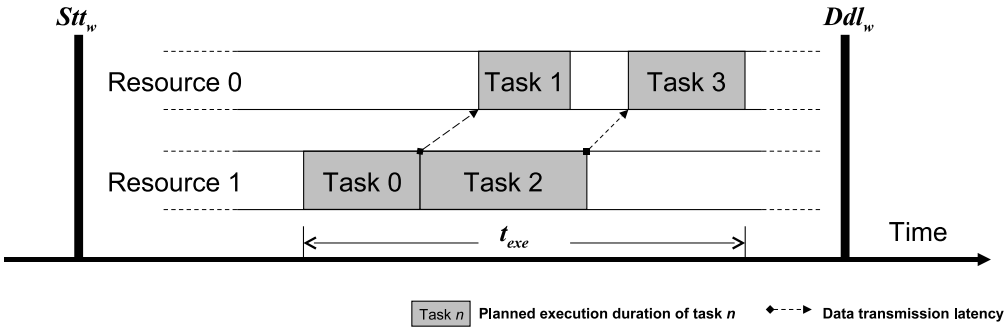


Figure 4.7: Time Constraint for a Workflow

4.3.4 Negotiation

A negotiation procedure is required in market-based Grid since there are typically conflicting objectives, from users and service providers, which need to be addressed in order to reach an agreement which satisfies both negotiating parties. It is apparent that an efficient negotiation mechanism for the market-based Grid should have minimal time cost and interference by the user and/or service provider. To determine a suitable negotiation mechanism for workflow applications on Grid, besides the topology of Grid, it is also necessary to take into account the resource heterogeneity and the complex task dependencies.

It is easy to imagine that all negotiations between clients (which can be the user or broker) and servers (which can be the broker or LRM) in the context of market-based Grid can be logically categorized into three types [YSOG05]: (i) Bid by Client, namely multiple clients bid for a service; (ii) Bid by Server, namely multiple servers bid for a client request; and (iii) Match-making, which means a third party is involved in producing an agreement to satisfy both client and server. The first negotiation type is apparently unsuitable for the model in this study. On one hand, there are not multiple brokers, but only one broker to bid for the service provided by LRM; on the other hand, if multiple clients bid for a service published by the broker, lots of user interference will have to be introduced. Negotiation type (ii) also has drawbacks. Because there is only one broker, similar to type (i), the case that multiple LRMs bid for a client request published by the broker can only be considered. In this case, if the workflow is considered for the bid as a whole, the winning LRM has to run the application independently and thus lose scalability. Otherwise, the workflow has to be considered for multiple

bids for several components (or tasks in the extreme case), and due to the task dependencies of the workflow, some bids have to be negotiated sequentially. This results in a time-consuming solution.

User		Broker		LRM
Submit a workflow with QoS constraints	→	Receive input from User		
		Enquire LRM about dynamic information	→	Process enquiry
		Receive input from LRM	←	Respond to enquiry
		Examine if it is possible to meet user constraints		
Receive rejection	←	If no, decline user's request; otherwise, ask for LRMs' confirmation	→	Process confirmation
		Receive acknowledgement (or rejection)	←	Acknowledge confirmation (or reject)
Receive SLA offer (or rejection)	←	Send offer (or rejection), SLA is recorded (or not)		

Figure 4.8: A single sequence of the modelled match-making negotiation

This thesis assumes a match-making negotiation mechanism (type (iii)) which avoids the drawbacks of other negotiation types mentioned above. In the match-making negotiation, the broker take inputs from the user (information about the submitted workflow and the QoS constraints) and service providers (information about the provided services and the instant capability of resources) to produce a mapping of the workflow which is agreeable to both the user side and the service provider side. In this way, the broker can free the user and service providers from the negotiation procedure. Moreover, the broker can make a decision within a reasonable time with an efficient workflow planning approach to cater to the real-time requirement of Grid computing. A single sequence of the modelled match-making negotiation is depicted in Figure 4.8. It should be noted that the broker just needs to ask the LRMs about dynamic information, such as the availability of a time slot according to the instant load of a particular resource, to know whether it is possible to run a service on this resource during a specific

period. Some relatively static information, e.g., which service is available from which resource and the service type etc, is always retained by the broker. It is also worth mentioning that when a negotiation fails, re-negotiation may be feasible since the instant capability of resources varies over time. However, for simplicity, a once-at-most negotiation is assumed in this thesis, namely, if the broker makes a positive decision in the first negotiation, the agreement is reached; otherwise, the user's request is rejected.

4.3.5 Pricing

In terms of pricing, two main factors, which may influence the final charge for each workflow done, were considered. One of them is how the price of task running is computed, and the other is user's constraint.

It should be noted that, when a user consumes a provided service, essentially, the user runs the task on the resource which provides the service. Therefore, the price of running a workflow task is calculated based on resource usage. Namely, given task i is successfully completed on resource m, the price p_i can be computed by

$$p_i = act_{i,m} \times \mu_m \tag{4.1}$$

where $act_{i,m}$ is the actual execution time of task i and μ_m is the price unit for resource m. To set μ_m, a *cost-plus pricing* method [Han92] was adopted, which is the primarily used pricing method in real business, since it is easy to calculate and requires little information. In the typical form of cost-plus pricing, the cost of the produce (or service) is calculated first, and then an additional amount is included to represent profit. Similarly, cost unit is defined as β_m $(0 < \beta_m < \mu_m)$ which denotes the cost per time unit for running the task on resource m. For a resource m, it is assumed that μ_m and β_m are associated with the power of m and can be pre-specified. Details of how to standardise resource power, μ_m and β_m will be provided in Section 5.5.1 and 6.5.2.

The final charge of a task also depends on whether or not the user's constraint is met. For simplicity, a rigid way was considered to compute the final price as below:

$$p_i = \begin{cases} act_{i,m} \times \mu_m & : \text{ If the user's QoS constraints are met} \\ 0 & : \text{ Otherwise} \end{cases} \tag{4.2}$$

Accordingly, the profit which can be earned from the task is computed by:

$$\rho_i = \begin{cases} p_i - act_{i,m} \times \beta_m & : \text{If the user's QoS constraints are met} \\ -act_{i,m} \times \beta_m & : \text{Otherwise} \end{cases} \quad (4.3)$$

4.3.6 SLA-based Scheduling Problem

Based on the model described above, the SLA-based scheduling problem which needs to be resolved is as follows: given a series of workflows that are submitted to the scheduling system at random times, the problem is how to plan, create and enforce an appropriate SLA for each workflow to tackle the grid uncertainties during the scheduling of workflow applications in dynamic Grid environments, so that not only the users' QoS constraints can be satisfied, but also the service providers' profit can be maximized.

In order to fulfill the complete Grid scheduling, there may be quite a few steps (e.g., application definition, information gathering, job submission etc.) to go through and many issues to address [Sch03]. This may result in a massive and complicated problem to resolve. The work present in this thesis does not cover all of these steps. It does, however, concentrate on three key issues, highlighting the features of an SLA-based scheduling system. These issues are: *planning*, *advance reservation* and *local scheduling*.

A diagram depicting what happens in the process of SLA-based scheduling is provided in Figure 4.9. Brief introductions of the three key issues, which will be further explored in the next 3 chapters, are presented as follows.

Planning

Having received a workflow application request from a user with a set of QoS constraints, to establish an SLA, whether or not it is feasible to fulfill the user's request must be pre-known. Thus, it is always necessary to seek a plan to allocate services for the execution of the application to ensure that all of the constraints from the user can be met at the same time. A plan is an initial schedule of the submitted workflow, in which the task-resource mapping, and the estimated start time and end time of every task is included. Given a set of pre-specified QoS constraints, the planning phase involves the procedure carried out by the broker to find such a plan where QoS metrics are optimized to such an extent that all the constraints can be met, whereas account is also taken of the load (on

	User		Broker		LRM
Compile Time (Before workflow execution)	Submit a workflow with QoS constraints	→	Receive inputs from User		
			Enquire LRM about dynamic information	→	Process enquiry
			Receive inputs from LRM	←	Respond to enquiry
			Plan task reservations of the workflow on resources (**Planning**)		
	Receive rejection	←	If it is impossible to meet user's constraints, decline user's request; otherwise, make advance reservations according to the planning result (**Advance Reservation**)	→	Process advance reservation requests
			Receive acknowledgement (or rejection)	←	If reservation succeeds, send acknowledgement, add reserved tasks to the execution queues of local resources; otherwise, send rejection
	Receive rejection Receive SLA offer	← ←	If advance reservation fails, decline user's request, otherwise, send and record SLAs for running the workflow	→	Get the signed sub-SLAs for the assigned tasks to run
Run Time (during execution)					If time comes to start a reserved task, run the task
			Receive on-time (or timeout)	←	When a task completes, if the actual end time is earlier to the end time of reserved duration, send on-time and launch rescheduling (**Local Scheduling**); if later, send timeout;
	Receive failure Receive success and make payment	← ←	If timeout is received, send failure; if on-time of the last task is received, send success		

Figure 4.9: SLA-based scheduling process of a single workflow

the execution of services) already present in the environment. If such a plan can be found, it is suggested that the user request can be accepted and the relevant SLAs can be agreed, otherwise, the request will be rejected.

Advance Reservation

When the planning phase approves that it is feasible to meet a user's QoS constraints, the broker moves on to make advance reservations for the workflow submitted by the user. First of all, the advance reservations need to contribute to guarantee the user's deadline requirement in a unpredictable grid environment. Introducing an extension into the estimated task execution time in advance reservation planning is a straightforward way to tackle the temporal uncertainties of grids in order to guarantee the user's QoS requirement for time-critical applications. However, this extension must be restricted with considering both the user's and the service provider's constraints in the context of a market-based Grid. Therefore, it is worth investigating how to extend the estimation of task execution time in advance reservation planning to strike a trade-off between different conflicting objectives from users and service providers. The aim of this advance reservation research is to determine 'how and to what extent the estimation of task execution time ought to be extended' in order to both guarantee users' QoS requirements and maximize service providers' benefits. Moreover, on the assumption that incoming workflows arrive sequentially and dynamically, the whole process of the advance reservation for the current workflow must be efficient to meet the real-time requirement of grid computing.

Local Scheduling

As a result of the aforementioned advance reservation, the reserved time for each task is normally longer than the estimated execution time of the task. It should be noted that a multiple-workflow scheduling scenario is considered, in which there are often multiple reserved tasks for each local resource, which do not have to be executed at a specific period, but must be completed before a certain point so that the deadline constraint for the relevant workflow can be satisfied. This provides flexibility for service providers to perform local scheduling, which reschedules the tasks assigned to local resources in order to maximize the utilization of resources without violating the user's constraints written in the SLA. Thus, it is necessary to develop novel SLA-based local scheduling strategies.

4.3.7 Summary

In terms of the taxonomy provided in Section 4.2.1, the proposed SLA-based scheduling system is summarized as follows: a commodity market is assumed, where users submit workflow applications and express their constraint-based QoS requirements on time and economic cost, and SLA-based scheduling approaches are developed in order to both meet the user's hard QoS constraint and maximize the service provider's profit. Multiple heterogeneous resources are assumed, which support single programming and advance reservation, and are decentralizedly controlled and managed by different local resource managers. A full-ahead planning scheme is used by the broker to determine whether or not the user's constraints can be met. The uncertainty of performance prediction and the existing loads of resources are taken into account in advance reservation and local scheduling (namely, schedule updating) during run-time.

4.4 Closing Remarks

This chapter has presented the migration from a performance-driven scheduling to an SLA-based market-driven scheduling, and revealed the significance of addressing uncertainties in SLA-based workflow scheduling. A generic SLA-based scheduling model is introduced and a state-of-the-art of the existing market-based Grid scheduling systems is reviewed. Due to the complexity, the SLA-based scheduling is broken into three sub-problems: SLA-based planning, SLA-based advance reservation and SLA-based local scheduling, and these will be resolved respectively in the next three chapters.

Chapter 5

SLA-based Planning

Supposing users want to make use of the services provided in the market-based grids to execute workflow applications within hard constraints on deadline and budget, the previous chapter designed the framework of the SLA-based scheduling system aiming at both guaranteeing users' QoS and maximizing service providers' benefits under grid uncertainties. This chapter focuses on Budget-Deadline-Constrained (BDC) planning issues for the proposed SLA-based scheduling system. The problem of concern is how the scheduling system can plan the submitted workflow to determine whether or not it is feasible to satisfy the user's constraints, and, therefore, reply accordingly to a user request to make an SLA.

The main contribution of this chapter is a novel BDC-planning heuristic which can efficiently find a plan which allocates services for the execution of the workflow so that both the pre-specified deadline and budget constraints can be met. Moreover, the load (on the execution of services) already present in the environment is also taken into account in the proposed planning heuristic.

5.1 Background

As mentioned in the previous chapter, the Service Level Agreement (SLA) plays a crucial role to guarantee the user's QoS requirements when a user requires the completion of the workflow before a hard deadline and within a certain budget in the context of a market-based grid. However, prior to establishing an SLA can be established, whether or not it is feasible to fulfill the user's request needs to be known. Therefore, it is necessary to seek a plan to allocate services for the execution of the application to ensure that all of the constraints from the user

can be met at the same time. If such a plan can be found, it is suggested that the user's request can be accepted and the relevant SLA can be achieved. Otherwise, the request will be rejected. This indicates the importance of the study to find a plan which meets the specified QoS constraints for a submitted workflow in the context of a market-based grid.

More specifically, the following planning scenario is considered: There are service providers who own resources and provide various computational services on these resources. A certain price will be charged by the service providers for successful service provision. There are users who need to use the service provided to compute their workflow applications. There is also a broker, which acts as the planner. The planning is considered to be online [CRH08], i.e., users submit workflows dynamically over time and the planner makes planning decisions based on the jobs which have been accepted and reserved, which are also perceived as the existing load of resources. Each workflow may start at a future time but must be completed before a certain deadline, and the total charge for workflow execution must be within the budget pre-specified by the user. A plan which meets both the budget and deadline constraints of a workflow is called a '*BDC-plan*'. Upon each user request, the planner must plan the submitted workflow on the provided services and then determine whether or not a BDC-plan can be found. If it can, the user request should be accepted; otherwise, it should be rejected. In such a scenario, several requirements must be considered in the planning:

- The dependencies among the multiple tasks of the workflow must be respected.
- The overall execution time (i.e., the makespan) and the total execution expense of the workflow must both be optimized so that the given deadline and budget are not exceeded.
- The planner, which does not really possess the resources, has to communicate with the owner, i.e., service provider, to make a scheduling decision based on the existing load of the resources;
- The planning process must be completed in a short time, because (i) the user normally expects a real-time response, and (ii) an evident planning latency may render the existing load information used in the planning out of date, since the load present in resources may vary over time.

Essentially, the problem considered in this chapter is bi-criteria constraint-based planning, which involves the planning process to optimize two metrics at the same time to meet the specified constraints (budget and deadline, here). A similar problem is optimization-based planning, which aims to find a solution which is optimal in terms of all metrics. An optimization-based planning solution can usually be applied to a constraint-based planning problem, whereas from the problem's point of view, the extensive attempt to optimize may not be of particular interest, since any solution meeting the constraints can be accepted by the problem. It should be noted that the application focus in this thesis is DAG-represented workflows. As the bi-criteria optimization problem for DAG is NP-complete [WPPF08], the planning problem is made a real challenge, and the usual solution adopted is the application of heuristics.

In the context of grid computing, several scheduling algorithms, which usually have a sophisticated design, have been proposed to acquire the bi-criteria optimization for DAG applications. It seems that these algorithms can easily be adapted to the aforementioned constraint-based planning problem. The question is "Are these sophisticated algorithms suitable for the constraint-based online workflow planning of the SLA-based scheduling system?". This chapter claims that, in the relevant scenario, simpler planning heuristics can be a better choice since they are as effective, more scalable and easier to implement.

To fulfill the aforementioned requirements of online workflow planning, a new low-cost heuristic is proposed, called Budget-constrained Heterogeneous Earliest Finish Time (BHEFT) to address the constraint-based optimization problem with two criteria (budget and deadline). A simulator is developed, where the planner can communicate with service providers via message passing to make the scheduling decision. It is demonstrated in the simulation that the proposed heuristic, which is albeit relatively simple, can be as effective as a sophisticated algorithm when finding a BDC-plan, but costs much less in terms of computation and communication overheads, and can therefore meet the real-time requirement of the online workflow planning.

The remainder of this chapter is structured as follows. The related works are reviewed in Section 5.2. The model assumed in this study and the problem definition are presented in Section 5.3. A novel BDC-planning heuristic (BHEFT) is described in Section 5.4. The experimental details and simulation results are discussed in Section 5.5. Finally, the chapter is concluded in Section 5.6.

5.2 Related Work

The problem of online job scheduling for achieving QoS in grid has been addressed in the literature. The work presented in [CRH07] and [CRH08] employs computational geometrical techniques in the designing of online scheduling algorithms for homogeneous and heterogeneous systems respectively. The algorithms are aimed to schedule multiple tasks in order to meet their start time and deadline constraints and maximize system utilization. In these studies, the application is modelled as a single task. Moreover, only time constraint is considered. In contrast, the work presented in this chapter focuses on the constraints of budget and deadline for DAG planning. Several budget-deadline constrained scheduling studies have been carried out in [BMAV05, FSZX04], whereas the application models take into account in these studies are still different from the DAG workflows. Workflows are considered in [SVF06], where a grid capacity planning approach based on 3-layered negotiation with advance reservation is proposed for a multi-criteria QoS expressed by utility functions. In this study, every workflow task has its own time constraint, and the allocator generates reservation offers for individual tasks via negotiation with resource providers and then the co-allocator launches the second layer negotiation between clients and allocators to build a co-allocation. The contentions between co-allocations are finally eliminated by the third layer negotiation. That is to say, this study mainly discussed negotiation mechanism rather than a bi-criteria planning heuristic. Moreover, our work consider deadline only for an entire workflow.

There is an abundance of literature about DAG scheduling heuristics. Being derived from earlier studies [SZ04a, KA99, BBR02b, THW02] of DAG scheduling on heterogeneous systems, several workflow management systems [TWML01, DBG+04, CDK+05] with scheduling heuristics have been developed to minimize the overall execution time of application, i.e., the makespan. These single-criteria heuristics are not suitable for being imposed in BDC-planning because the service prices and the relevant execution costs are not considered.

A multi-criteria scheduling problem may also be studied in a different context. The studies in [QK05, Qua06b] proposed mapping heuristics to meet deadline constraints while meanwhile minimizing the reservation cost of workflows, whereas they regarded workflow tasks as being multiprogramming, which is not commonly adopted in workflow scheduling studies [WHP08]. Based on the model of Utility Grids, the time-cost constrained optimization has been studied in

meta-scheduling [GKB08, GBS09], where the meta-broker (scheduler) is assumed to run in batch-mode, i.e., the meta-broker waits until the end of a certain time interval and batches all of the applications submitted during the interval into allocation. In contrast, this study considers the planning to be online, namely each new application is planned immediately after its submission.

Several efforts have been made to develop bi-criteria scheduling heuristics for workflow applications. Some of these, including [WPPF08], [SZTD05] and [DO05], do not consider the load on the existing load of resources in their assumption.

Wieczorek et al. [WPPF08] propose a two-phase algorithm (DCA) to address the optimization problem with two independent generic criteria for workflows in Grid environments. The algorithm optimizes the primary criterion only in the first phase, then optimizes the secondary criterion while keeping the primary one cost within the defined sliding constraint.

In [SZTD05] two scheduling heuristics based on guided local optimization, LOSS and GAIN, were developed to adjust a schedule, and these may be generated by a time-optimized heuristic or a cost optimized heuristic, to meet users budget constraints respectively.

The work presented in [DO05] focuses on a trade-off between execution time and reliability, and proposes two bi-criteria scheduling algorithms called BDLS and BGA. The former is an extension of the DLS algorithm [SL93], and the latter is a genetic algorithm. Except for BDLS, these heuristics are all sophisticated, since they all require a considerable number of searches to obtain a final result. BDLS is a list scheduling heuristic which adopts a dynamic priority to make scheduling decisions.

All of these heuristics are almost applicable to BDC-planning except that without considering the existing load of resources, they tend to produce plans which lead to reservation conflicts, i.e., the planned task may overlap the tasks of other workflows which have already been reserved, and this may mean an overload of resource. In order to eliminate this overlap, the heuristics can be modified in two ways, the first of which is enquiring of the service provider whenever necessary during scheduling. This approach, as will be demonstrated later, may introduce a heavy communication overhead in complex heuristics. Even for BDLS, which has to rely on the enquiry to compute dynamic priority repeatedly during scheduling, the communication overhead can also be significant. The other way is producing an initial schedule without considering the existing load, and then allocating each

task to the earliest possible available time slot of the mapped resource, while the task dependency constraint is not violated. However, this may lead to the degradation of heuristic performance.

The existing load of resources is considered in some studies [YB06b], [YB07], [TKB07] and [SKD07], where evolutionary algorithms (for example, Genetic Algorithms) based heuristics have been proposed to resolve the multi-objective (time and cost, commonly) optimization problem. Such algorithms may naturally be too time expensive for BDC-planning. Moreover, in these studies, the service providers usually allow the planner to retrieve free time slots of resources. This may not be acceptable for some service providers who do not want their workload, which may be commercially sensitive, to be exposed. Conversely, it is assumed in the present study that the planner asks for a certain length of time slot, and the service provider responds with the earliest availability.

Unlike the aforementioned sophisticated bi-criteria scheduling heuristics, BHEFT is an efficient planning heuristic which does not involve a significant amount of computation or heavy communication, but is as effective as the sophisticated heuristics for online BDC-planning. By applying BHEFT, the SLA-based scheduling system will be able to effectively determine whether a workflow request should be accepted or rejected in a real-time manner. Therefore, the establishment of an SLA is facilitated.

5.3 Problem Description

This section identifies the BDC-planning problem to be addressed in this chapter. An overview of the problem is provided in Section 5.3.1 and more details about problem model are presented in Section 5.3.2. Table 5.1 lists the notations used in this and the next two chapters.

5.3.1 Problem Overview

There are three key roles: *service provider*, *user* and *broker* in the BDC-planning model in the present study. The service providers administer multiple, heterogeneous resources which provide services of different capabilities and at different costs. The user wants to make use of the provided services to run a workflow application within a certain deadline and budget. The broker, which acts as an intermediary between the user and the service provider to hide the complexities

Symbol	Meaning
w	a submitted workflow
Sch_w	advance reservation planning result for w
Bgt_w	specified budget constraint for w
Ddl_w	specified deadline constraint for w
Stt_w	the earliest possible time to execute w
τ_i	a workflow task i
θ_i	advance reservation for task i
t_i^{stt}	start time of task reservation
t_i^{end}	end time of task reservation
c_i	estimated charge for task reservation
ϵ_i	the specified service type of task i
z_i	the specified task size of task i
γ_p	resource p
α_p	power ratio of resource p
$s_{i,p}$	a service which is provided in resource p and can be used by task i
$\mathcal{X}_p$	existing reservation load in resource p
μ_p	price unit for resource p
rd_i	reserved duration for task τ_i on resource p
$ert_{i,p}$	estimated task execution time of task i on resource p
$f_Q()$	the operation used by broker to enquire the earliest available time slot from the relevant LRM
γ^*	the benchmark resource
ϕ_d	constraint ratio for deadline
ϕ_b	constraint ratio for budget

Table 5.1: Notations

of Grid computing, is the planner which performs the planning for the execution of the workflow. In order to make planning decisions, the broker requires various inputs from the user and the service providers. These inputs are modelled as follows:

Let $\mathcal{R}$ represent the set of all resources and $\mathcal{S}$ denote the set of all provided services. Every workflow $w = (\mathcal{T}, \mathcal{D})$ consists of a group of tasks $\mathcal{T}$ and a set of data dependencies $\mathcal{D}$ among the tasks. Each task $\tau_i \in \mathcal{T}$ needs to specify the type of service it wants to use. In order to run a workflow, every task of the workflow must be mapped to a service belonging to the service type specified by the task. From the perspective of τ_i, there is a set of services (denoted by $\mathcal{S}_i = \{s_{i,p} : \tau_i \in \mathcal{T}, \gamma_p \in \mathcal{R}_i\}$), which can be used by the task τ_i, where $\mathcal{R}_i \subseteq \mathcal{R}$

and $\mathcal{S}_i \subset \mathcal{S}$

As a result of planning, the schedule of workflow w can be represented by $Sch_w = \{\theta_i : \tau_i \in \mathcal{T}\}$ consisting of sub-plans θ_i for workflow tasks, which is defined as

$$\theta_i = (t_i^{stt}, t_i^{end}, s_{i,p^*}, c_i) \tag{5.1}$$

where t_i^{stt} is the planned start time of the task, t_i^{end} is the planned end time of the task, s_{i,p^*} is the service to which the task is mapped, and c_i, as defined in Eq.(5.2), is the estimated charge for executing the task.

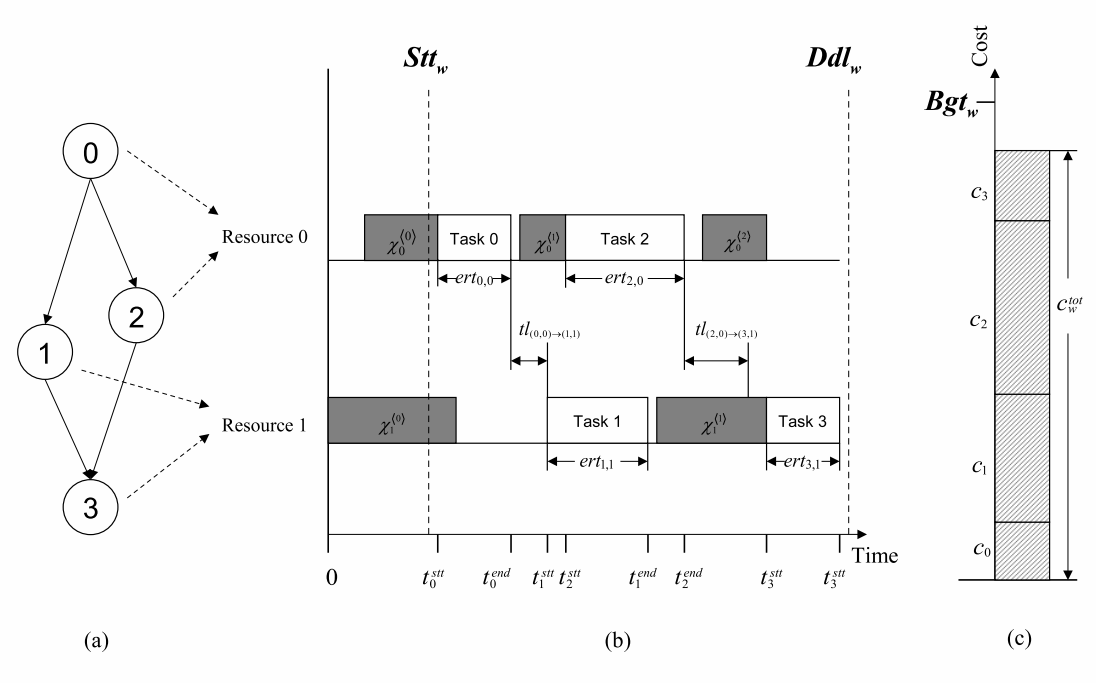


Figure 5.1: (a) Workflow w, (b) Plan of w on 2 resources as a timetable, and (c) Estimated reservation cost for w

Suppose workflow w is associated with a budget constraint Bgt_w and a deadline constraint Ddl_w, as shown in Figure 5.1, the *BDC-planning problem* is how to efficiently generate such a plan Sch_w to satisfy the following conditions:

- The planned reservation duration rd_i for each task τ_i is long enough to run the task. Namely, $\forall \theta_i \in Sch_w, rd_i = (t_i^{end} - t_i^{stt}) \geq ert_{i,p}$, where $ert_{i,p}$ is the estimated run time of task τ_i given that τ_i is mapped to service $s_{i,p}$. Concentrating on BDC-planning, an accurate estimation in this chapter is assumed for simplicity, i.e., $rd_i = ert_{i,p}$, while leaving the issue of prediction

uncertainty to the next chapter.

- The precedence constraint within the workflow is respected. Namely, $\forall(\tau_i \rightarrow \tau_j) \in \mathcal{D}, t_j^{stt} \geq t_i^{end} + tl_{i,j}$, where $tl_{i,j}$, as defined later in Eq.(5.3), is the data transmission latency from task τ_i to τ_j;

- The total estimated charge of advance reservation c_w^{tot} is not more than the budget. Namely, $c_w^{tot} = \sum_{\tau_i \in \mathcal{T}} c_i \leq Bgt_w$.

- The estimated end time of the workflow t_w^{end} is not later than the deadline. Namely, $t_w^{end} = \max_{\tau_i \in \mathcal{T}}\{t_i^{end}\} \leq Ddl_w$

- The resource must be available during the planned time slots for workflow tasks.

Given the same condition of budget-deadline constraints and the existing load of resources, different planning decisions may impact whether or not the planning result is a BDC-plan. The objective of BDC-planning heuristics is to maximize the likelihood that a BDC-plan can be successfully found for a submitted workflow request. Also, the time overhead caused by the heuristics is expected to be sufficiently low to cater to the real-time requirement of Grid environments.

5.3.2 Problem Modelling

Several concepts involved in the problem presentation of the previous section are modelled as follows:

- *Resource Modelling*. Multiple heterogeneous resources are exclusively owned by several Local Resource Managers (LRMs) and provide various types of services. Similar to the resource model in Chapter 2, all resources are fully connected, and the transmission rate (namely the time needed to transmit per data unit from one resource to another) between each two resources (say, γ_p and γ_q) is pre-known and denoted by $tr_{p,q}$; especially, $tr_{p,q} = 0$ when $p = q$.

- *Workflow Modelling*. One workflow is represented by a DAG as modelled in Chapter 2. Two parameters must be specified for each node τ_i, one of which is *service type* ϵ_i, which specifies the type of service the task wants to consume. Task τ_i can be assigned to resource γ_p only if γ_p implements

the service of type ϵ_i. The other parameter is *task size* z_i, which is a value quantifying the computational demand of the task. The execution time of task τ_i is proportional to z_i.

- *Service Modelling*. There is a set of all service types $\mathcal{K}$. Suppose that task τ_i specifies a service type $\epsilon_i = \kappa \in \mathcal{K}$, if κ is supported in resource γ_p, $s_{i,p}$ exists. For simplicity, it is assumed that every service type is supported in every resource. Thus, a task can be mapped to any resource, i.e., $\mathcal{S}_i = \{s_{i,p} : \tau_i \in \mathcal{T}, \gamma_p \in \mathcal{R}\}$. Mapping τ_i to $s_{i,p}$ means assigning τ_i to resource γ_p. To calculate the economic cost for a workflow task τ_i, the pricing scheme presented in Section 4.3.5 is adopted, and the price unit of resource γ_p is assumed to be known. Thereby, c_i can be computed by

$$c_i = \mu_p \cdot ert_{i,p} \tag{5.2}$$

  In addition, for each edge $i \rightarrow j$, where τ_i, τ_j are different nodes, a value $d_{i \rightarrow j}$, which denotes the data size transmitted from i to j, is associated. Similar to the definition in Chapter 2, given that task τ_i and τ_j are mapped to γ_p and γ_q respectively, the transmission latency $tl_{i,j}$ can be computed by

$$tl_{(i,p) \rightarrow (j,q)} = d_{i \rightarrow j} \times tr_{p,q} \tag{5.3}$$

- *Time Issues*. Generally, the time parameters involved in BDC-planning are similar to those defined in the deterministic scheduling model as described in Section 2.2.1. It is assumed that the estimated execution time of task τ_i on resource γ_p, i.e., $ert_{i,p}$, is known. The data available time ($dat_{i,p}$), which means the time at which all of the data required by node i arrives at resource p, can be computed by

$$dat_{i,p} = \begin{cases} Stt_w & : \quad i = entry \\ \max\limits_{k \in Pred(i)} \{t^{end}_{k,r(k)} + tl_{(k,r(k)) \rightarrow (i,p)}\} & : \quad i \neq entry \end{cases} \tag{5.4}$$

  where $Pred(i)$ denotes the set of all immediate predecessors (i.e., parent tasks) of task τ_i, $r(k)$ means the resource to which task τ_k has been assigned, and $t^{end}_{k,r(k)}$ means a finish time of τ_k. However, recalling the formula of the computing task start time defined in Eq.(2.3), the planner needs to know the

resource available time to determine the start and finish times of a specific task on a specific resource. Therefore, the planner will have to communicate with the relevant service provider every time the computation of task start and finish times is needed. This computation is significantly involved in the heuristics which focus on makespan optimization, since it is often used to calculate task priority when making planning decisions, and is always needed to evaluate the makespan of a given task-resource mapping.

It is necessary to model the existing load of resources before explaining how the planner retrieves the start and finish times of a task. The load on the execution of services in resources is virtually composed of the reservation of tasks. Given a specific moment, the existing load of resource p is defined as the union of time spans already reserved in the resource, i.e.,

$$\mathcal{X}_p = \bigcup_{j=0}^{n-1} (\mathcal{X}_p^{\langle j \rangle}) = \bigcup_{j=0}^{n-1} (t_{\langle j \rangle}^{stt}, t_{\langle j \rangle}^{end}) \tag{5.5}$$

where $(t_{\langle j \rangle}^{stt}, t_{\langle j \rangle}^{end})$ denotes a reserved time span, j is the index of the reservation in the resource, and n is the number of reserved time spans already present in resource p. It should be noted that it is assumed that a resource can only execute one task at a time without considering preemption [ZS06a].

It is assumed that the broker can communicate with any service provider via message passing. In order to obtain the start and finish times of a task on a particular resource, which is essentially a request for a available time slot, the planner needs to send a QUERY message to the service provider which owns the resource. From the message, the service provider should know that the broker is interested in the earliest available time slot with length d on resource γ_p and the time slot must be after time s, where d is usually considered to be the estimated execution time of task τ_i on resource γ_p and s is the data available time $dat_{i,p}$. Having received the message, the service provider looks through the existing load of resource γ_p, i.e., $\mathcal{X}_p$ to find the earliest available time slot and relays the result to the broker. To formulate this process, a function f_Q: $\mathcal{T} \times \mathcal{R} \times \mathbb{R}^+ \times \mathbb{R}^+ \mapsto \mathbb{R}^+$ is defined

as below:

$$f_Q(\tau_i, \gamma_p, s, d) = \min\{(a,b)|(a,b) \cap \mathcal{X}_p = \emptyset, a \geq s, b = a + d\} \quad (5.6)$$

where task $\tau_i \in \mathcal{T}$, resource $\gamma_p \in \mathcal{R}$, $a, b, s, d \in \mathbb{R}^+$, $\mathcal{X}_p$ is defined in Eq.(5.5). For concretization, the following example is provided: suppose that $\mathcal{X}_0 = \{(0,5),(8,12),(17,30)\}$ and for task 1, $dat_{1,0} = 3$ and $ert_{1,0} = 4$, then the earliest available time slot for allocating task 1 on resource 0 will be (12, 16). In this case, $f_Q(1,0) = (12,16)$. Finally, the whole procedure for the broker to retrieve the earliest available time slot is defined as a *Time Slot Query (TSQ)*.

5.4 A Proposed Heuristic

As introduced in Section 2.3.1, the Heterogeneous Earliest Finish Time (HEFT) algorithm is a well-known static list scheduling heuristic which has been developed to allocate a DAG application to a set of heterogeneous resources to minimize the overall execution time of the application [THW02]. While being powerful at optimizing makespan, the HEFT algorithm does not take into account the monetary cost and budget constraint when making mapping and scheduling decisions. In this section, the HEFT algorithm is extended in order to resolve the BDC-planning problem and the new algorithm is named the Budget-constrained Heterogeneous Earliest Finish Time (BHEFT). The outline of the BHEFT is presented in Figure 5.2.

Similar to the original HEFT algorithm, the BHEFT also has two major phases: *task prioritizing* and *service selection*.

In the task prioritizing phase, the priorities of all tasks are computed on an upward ranking. The rank of a task i is recursively defined by

$$rank_i = \overline{ert}_i + \max_{j \in Succ(i)} \{\overline{tl}_{i,j} + rank_j\} \quad (5.7)$$

where $Succ(i)$ is the set of the child tasks of task i, $\overline{ert}_i = \sum_{p=0}^{|\mathcal{R}|}(ert_{i,p})/|\mathcal{T}|$ is the average execution time of task i, $\overline{tl}_{i,j} = \sum(tl_{(i,p)\rightarrow(j,q)})/(|\mathcal{R}| \cdot |\mathcal{R}|)$ is the average transmission latency of edge $i \rightarrow j$. Especially, the rank of exit node $rank_{exit} = \overline{ert}_{exit}$.

Input: DAG G, Constraints C comprising Budget B and Deadline D;
Output: A BDC-plan

1. Compute *rank* (as defined in Eq.(5.7)) for all tasks.
2. Sort all tasks in a planning list in the non-ascending order of *rank*.
3. **for** $k := 0$ to $|\mathcal{T}| - 1$ **do** (where $|\mathcal{T}|$ is the number of tasks)
4. Select the kth task from the planning list.
5. Compute the Spare Budget for Application for task k (as defined in Eq.(5.8)).
6. Compute the Current Budget for Task for task k (as defined in Eq.(5.9)).
7. Construct the set of Affordable Services (as defined in Eq.(5.10)) for task k.
8. **for** each service which can be used by task k **do**
9. Compute the earliest finish time of mapping task k to the service based on the existing load of the resource providing the service (as described in Section 5.3.2).
10. **endfor**
11. Select service for task k in terms of one of the selection rules at the end of this section.
12. **endfor**

Figure 5.2: The BHEFT Heuristic

In the service selection phase, the tasks are selected in order of priority. Each selected task is allocated to its "best possible" service, of which the metric may change according to the circumstance of the spare budget which varies as the planning proceeds. The circumstances of the budget are considered in terms of two variables: the *Spare Budget for Application* (SBA) and the *Current Budget for Task* (CBT). Suppose that the kth task is being allocated, SBA_k and CBT_k are respectively computed by

$$SBA_k = B - \sum_{i=0}^{k-1} c_i - \sum_{j=k}^{n-1} \overline{c}_j \tag{5.8}$$

$$CBT_k = \begin{cases} \overline{c}_k + SBA_k \times \overline{c}_k / \sum_{i=k}^{n-1} \overline{c}_i & : \quad SBA_k \geq 0 \\ \overline{c}_k & : \quad SBA_k < 0 \end{cases} \tag{5.9}$$

where B is the given budget, c_i is the reservation cost of the allocated task i, $\overline{c}_j = (\sum_{i=p}^{|\mathcal{R}|} c_{j,p})/|\mathcal{R}|$ is the average reservation cost of the unallocated task j over different resource mappings, n is the number of tasks. Provided that $\epsilon_k = \kappa$, a set $\mathcal{S}'_\kappa$ is constructed consisting of an *affordable service* for task k, i.e.,

$$\mathcal{S}'_\kappa = \{s_{\kappa,p'} | c_{k,p'} \leq CBT_k\} \tag{5.10}$$

Then the "best possible" service is selected by the selection rules as follows:

1. If $\mathcal{S}'_\kappa \neq \emptyset$, the affordable service with the earliest finish time is selected;

2. If $\mathcal{S}'_\kappa = \emptyset$ and $SBA \geq 0$, the service with the earliest finish time selected;

3. If $\mathcal{S}'_\kappa = \emptyset$ and $SBA < 0$, the cheapest service is selected;

5.5 Performance Evaluation

5.5.1 Simulation Model

The performance of the proposed heuristic was compared with several sophisticated bi-criteria scheduling heuristics by simulation. The simulated resources and services were modelled with the following assumptions:

Standardisation of Resource Heterogeneity

In order to model the resource heterogeneity each resource γ_p is associated with a positive value named *power ratio* α_p to describe the resource power. It is also assumed that (i) there is a benchmark resource γ^* with power ratio $\alpha^* = 1.0$; (ii) to complete a given task τ_i, the estimated execution time of is in inverse proportion to its power ratio. Namely, we have:

$$ert_{i,p} = \frac{\alpha^*}{\alpha_p} ert_i^* = \frac{ert_i^*}{\alpha_p} \tag{5.11}$$

where $ert_{i,p}$ and ert_i^* are the estimated execution times of task τ_i on resource γ_p and γ^* respectively. Different service types may differ in complexity and require different processing times. For standardisation, it is assumed that it can be measured that the benchmark resource γ^* takes time t_κ^* (named *standard service time*) to complete a task specifying service type κ and task size equal to 1.0. Thereby, according to Eq.(5.11) and the definition of task size, the estimated execution time of task τ_i, which uses service type κ on resource γ_p can be computed:

$$ert_{i,p} = \frac{t_\kappa^* \cdot z_i}{\alpha_p} \tag{5.12}$$

Standardization of Price Setting

It is assumed that the more powerful resources cost more and earn more profit than less powerful resources. In order to standardize diverse price units for heterogeneous resources, price unit $\mu^* = 1.0$ is specified for the benchmark resource.

Then, the price unit μ_p for resource γ_p is correlated with the power ratio α_p and can be computed as follows:

$$\mu_p = \frac{\alpha_p(1+\alpha_p)}{2}\mu^* = \frac{\alpha_p(1+\alpha_p)}{2} \tag{5.13}$$

Random Generation of Existing Load of Resources

The existing load of resources was randomly generated for simulation. Given a specific period between time t_1 and t_2, the existing load of each resource p (i.e., $\mathcal{X}_p$) is parameterized by two pre-specified values: Utilization Rate (UR) and Average Task Load (ATL). The former is the ratio of the total reserved time to the whole period, and the later is the average number of tasks appearing in per time unit. Apparently, the average reserved duration is $\overline{RD} = UR/ATL$ and the average idle duration is $\overline{ID} = (1-UR)/ATL$. The following procedure describes how the existing load of resource p ($\mathcal{X}_p$) was constructed:

1. Set $\mathcal{X}_p = \emptyset$ and current time $CT = t_1$.
2. Randomly determine current state among RESERVED and IDLE.
3. If RESERVED:
   (a) randomly generate reserved duration RD by normal distribution with mean $\overline{RD}$ and standard deviation $\overline{RD}/6$ whereas only $RD > 0$ is adopted;
   (b) set $\mathcal{X}_p = \mathcal{X}_p \cup (CT, CT + RD)$;
   (c) set $CT = CT + RD$;
   (d) switch current state to IDLE.
4. If IDLE:
   (a) randomly generate idle duration ID by normal distribution with mean $\overline{ID}$ and standard deviation $\overline{ID}/6$ whereas only $ID > 0$ is adopted;
   (b) set $CT = CT + ID$;
   (c) switch current state to RESERVED.
5. Repeat Steps 3 and 4 till CT reaches t_2.

5.5.2 Experimental Setting

To run the experiments, a job planner (broker) and a set of resources were simulated by java programs distributed on computing nodes with 3.0 Ghz cpu, 1 GB memory and connection through Gigabit Ethernet. The communication between the broker and the service providers was implemented by socket programming. The software used in the experiment included Fedora release Core 5, MySQL 4.1.22 and JDK 1.5.0.

There were 2 service providers in the evaluation, each of which managed 3 resources, namely 6 resources in total. There were 4 service types having a standard execution time of 10, 15, 25 and 30 respectively. For each resource p, the capability ratio α_p was randomly generated from [0.5, 2.0]. The period considered for existing load modelling was [0, 5000].

As depicted in Figure 2.12, four types of DAGs were considered in the experiments, including fMRI [ZWF+04] with 17 nodes, Montage [BGL+04] with 34 nodes, AIRSN [HDW+05] with 53 nodes and LIGO [DKM+02] with 77 nodes. For each task i, ϵ_i was randomly selected from the provided service types, and z_i was randomly generated from [0.5, 2.0]. The communication computation ratio (CCR) was randomly selected from [0.1, 1.0].

Given a DAG, the deadline and budget constraints were considered as follows. For simplicity, a job was always assumed to start at time 0. The makespan M_{HEFT} was computed by applying the HEFT algorithm [THW02] to the DAG without considering the existing load of resources. The deadline constraint DC was considered to be located between the lower bound $LB_{dc} = M_{HEFT}$ and the upper bound $UB_{dc} = 5 \times M_{HEFT}$. A deadline ratio ϕ_d was used to depict the position of DC by $DC = LB_{dc} + \phi_d \times (UB_{dc} - LB_{dc})$, where $0 \leq \phi_d \leq 1.0$. For budget constraint, LB_{bc} was the lowest total cost obtained by mapping each task to the cheapest service, and UB_{bc}, the highest total cost obtained conversely. Similarly, a budget ratio ϕ_b was used to specify the possible budget constraint $BC = LB_{bc} + \phi_b \times (UB_{bc} - LB_{bc})$, where $0 \leq \phi_b \leq 1.0$.

BHEFT was compared with DCA [WPPF08], LOSS [SZTD05] and BDLS [DO05] in the experiments. As mentioned in Section 5.2, some modification is needed to adapt these heuristics, which do not consider the existing load of resources, to produce contention-free plan. According to the evaluation present in [WPPF08], in their original version, DCA, which is based on extensive local search, has the best optimization performance but the highest time overhead, as opposed to BDLS

which is a static list scheduling heuristic using a dynamic priority. Therefore, only a TS Query was introduced into LOSS and BDLS while DCA was modified in the other way mentioned in Section 5.2. When showing the experimental results, the suffix '_TSQ' was added to the names of the algorithms which used TS Queries, to distinguish them from DCA which do not consider TS Query, while the original names are used for short in the discussion. In terms of the configuration of DCA and BDLS, the same setting as used in [WPPF08] is adopted, i.e., LOSS3 is adopted to represent LOSS, a memorization table consisting of 100 cells with up to 10 intermediate solutions stored in each cell was used by DCA, and the parameter δ for BDLS was determined by a binary search which a maximum of 15 loops. Moreover, all heuristics terminate immediately when a BDC-plan is found.

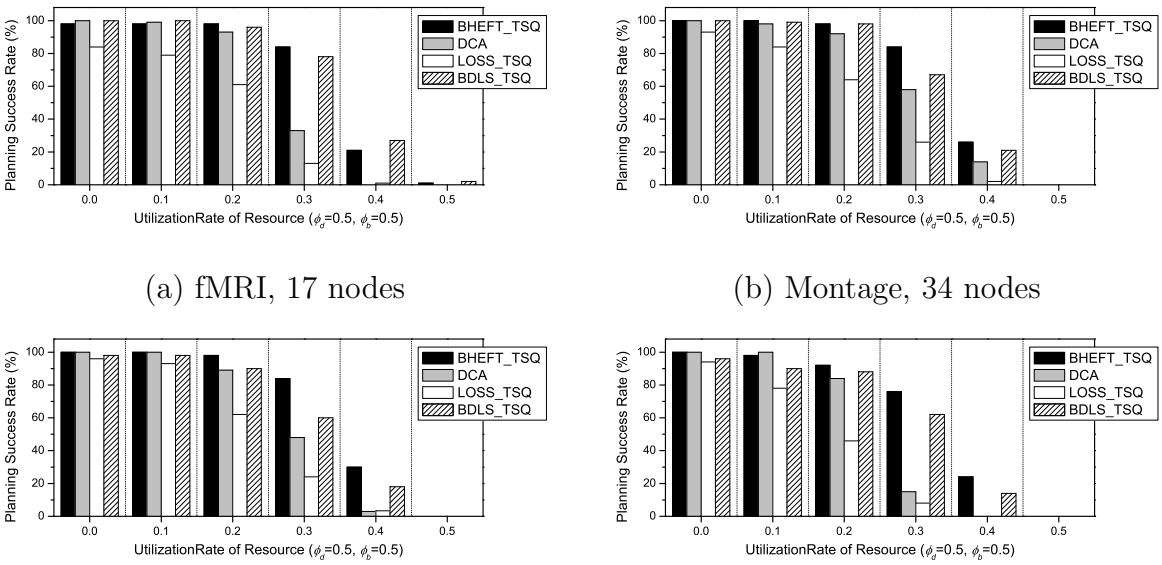
For each experiment, all of the parameters except for those which were given and fixed, were re-initialized at random with the above specifications. After a heuristic was run, if a BDC-plan was found, the planning succeeded, otherwise, a failure was reported. To analyze the performance of each heuristic, the experiment was repeated by multiple times and the metric *Planning Success Rate* (PSR) was used, as defined below:

$$PSR = 100 \times \frac{\text{the times for which a BDC-plan was found}}{\text{the total repeated times of experiment}} \tag{5.14}$$

Three sets of experiments were carried out. In the first one, ϕ_d and ϕ_b were fixed to be 0.5, while UR was varied for each resource from 0.0 to 0.6 in the step of 0.1 with the corresponding $ATL = 0.05 \times UR$. The experiment was repeated 500 times to learn how the existing load of resources affected the PSR of each heuristic. In the second set of experiments, UR was randomly generated in the interval of [0.1, 0.4], and the ATL was computed correspondingly. ϕ_d and ϕ_b were selected in the set of $\{0.25, 0.5, 0.75\}$ to form 9 combinations of constraints which covered a wide spectrum of diverse user requests, and the experiment was repeated 500 times for each combination of constraints. The result of PSR was investigated under various constraints (from tight to relaxed). At last, the average number of TS Queries needed was measured, as well as the average running time on planning for each heuristic. This experiment was repeated for 100 times for per workflow with various combinations of constraints.

5.5.3 Experimental Results

Figure 5.3 shows the results of the first set of experiments where the impact of the existing load of resources is investigated. Here, ϕ_d and ϕ_b are both fixed to be 0.5 to avoid unnecessary disturbance caused by setting the user constraints to be too tight or too relaxed.



(a) fMRI, 17 nodes

(b) Montage, 34 nodes

(c) AIRSN, 53 nodes

(d) LIGO, 77 nodes

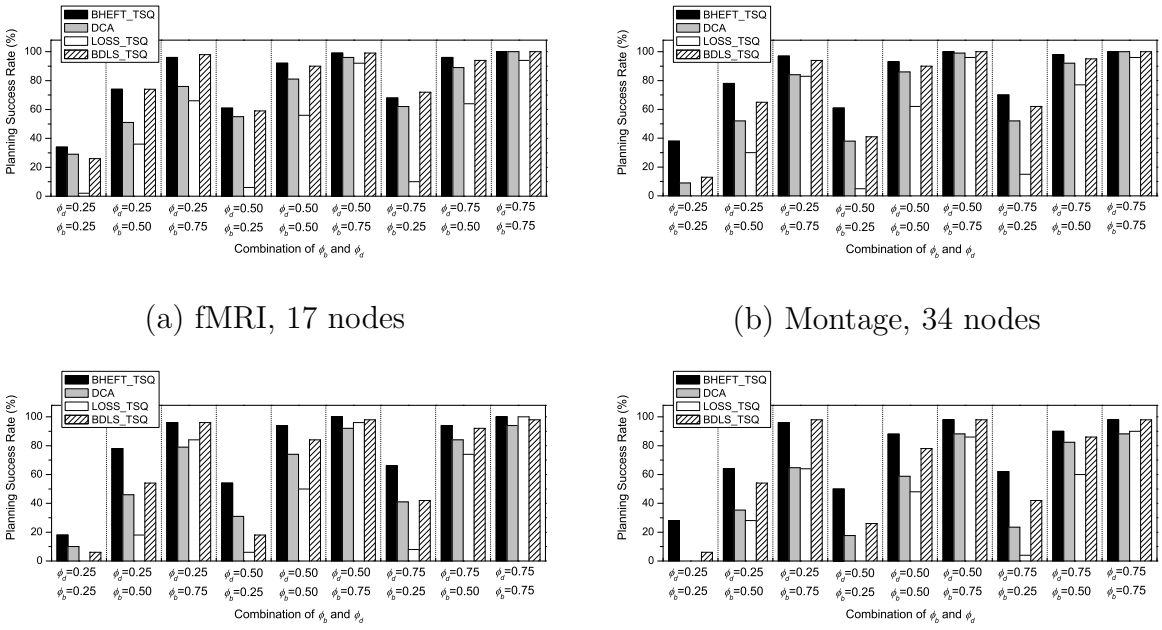
Figure 5.3: *PSR* results with the utilization rate of resource changes

It can be seen from Figure 5.3 that for all types of DAG, the comparison results of the compared heuristics of *PSR*s are similar. It is interesting to see that BHEFT performs the best over various settings of *UR*. The comparison results between LOSS and BDLS are a bit surprising. Both of these two algorithms consider the TS Query when making a scheduling decision, whereas BDLS, which is a list scheduling heuristic, evidently outperforms the LOSS algorithm based on a guided local search. However, according to the evaluation result in [WPPF08] where the existing load of resources is not considered, LOSS performs better than BDLS on optimization. This may be explained by the fact that LOSS is more sensitive to the impact of the existing load of resources, even though it uses TS Queries when making a scheduling decision. DCA proves to be a powerful algorithm at multi-criteria optimization. When *UR* is equals to 0 or 0.1, which means little existing load on resources, without using a TS Query DCA can achieve a result which is comparable to that of BHEFT and even slightly better than BDLS.

However, as UR increases, the performance of DCA unavoidably degrades due to the lack of the information about the existing load when scheduling. It can be observed how the performance of different algorithms evolves as UR changes. In the case $UR = 0.0$, this means that there is no existing load on resources and thus the consideration of existing load of resources in planning is not necessary, Almost all algorithms can obtain a PSR close to 100% except for LOSS in the case of fMRI DAG. As UR increases, the PSRs of all the compared heuristics degrade, while the PSR of BHEFT and BDLS evidently degrades less slowly than that of DCA and LOSS. The PSR of BHEFT manages to retain a pretty high PSR (above 80%) until UR reaches 0.3, and has a certain likelihood to obtain a BDC-plan when $UR = 0.4$. BDLS is the second-best performer, and can obtain a PSR which is at least 60% when UR is equal to 0.3. In contrast, in the majority of cases, the PSRs of DCA and LOSS drop to under 50% when UR reaches 0.3 and almost decline to below 5% when $UR = 0.4$. When $UR = 0.5$, every compared heuristic can hardly find any BDC-plan.

In the second set of experiments, more details were explored of the performance of each heuristic under various circumstances of user constraints. As illustrated in Figure 5.4, the evaluation results of PSR were collected and grouped into different types of DAG with different budget-deadline constraints.

The first focus is on the impact of the varying user constraints, and in extreme cases, where both deadline constraint and budget constraint are tight, for example, ϕ_d=0.25 and ϕ_b=0.25, all algorithms obtain low PSRs. Among them, BHEFT achieves the best PSR which is between 20% to 40%. When a small DAG (e.g., fMRI) is used, DCA and BDLS both obtain PSRs which are comparable to those of BHEFT. However, their PSRs turn significantly lower when the bigger DAG is used. Without considering the existing load of resources, DCA usually performs poorly when the deadline constraint is tight, i.e., ϕ_d=0.25. In contrast, the performance of LOSS is particularly poor when the budget constraint is tight. This is because the initial plan of LOSS constructed by using HEFT usually has a small makespan and an high monetary cost, which may make it difficult for LOSS to adjust the plan to meet budget constraint within a limited number of local searches. BDLS can be almost as effective as BHEFT in many cases, for example, when both budget and deadline constraints are above 50%. In the case where a small DAG (i.e., fMRI) is considered, BDLS can even obtain better PSR than BHEFT. However, in most cases, BHEFT performs better than BDLS. The



(a) fMRI, 17 nodes

(b) Montage, 34 nodes

(c) AIRSN, 53 nodes

(d) LIGO, 77 nodes

Figure 5.4: *PSR* results with the constraints change

advantage of BHEFT is especially significant when at least one of the constraints is tight and the used DAG is big.

In the third experiment, the number of TS Queries was measured, as well as the time cost needed by each algorithm to obtain a planning result. It should be noted that although DCA does not consider the existing load of resource to produce a plan, it still needs a few TS Queries to adjust the produced plan to be contention-free.

Figure 5.5 demonstrates the average number of TS Queries needed by each algorithm in each planning process. The DCA which does not consider the existing load of resource in the planning decision making, uses the fewest TS Queries. Out of all of algorithms which consider the existing load of resources, BHEFT needs the fewest TS Queries and the number is consistent over different settings of constraints. As another list scheduling heuristic, the BDLS uses more TS Queries to compute the dynamic priority which varies during scheduling. As a guided local search heuristic, LOSS normally requires search loops at a maximum of the production of the number tasks and the number of resources, which may be relatively fewer compared to DCA and some evolutionary algorithms with a configuration for extensive search. However, LOSS still needs many more TS

Queries than the list scheduling heuristics, and this, as will be shown in the time cost measure result, cause an unacceptable scheduling overhead, especially for large applications. It should also be noted that the number of TS Queries needed by BDLS and LOSS are both closely related to the tightness of the constraints. For relaxed constraints, these algorithms can normally terminate much earlier by successfully finding a BDC-plan.

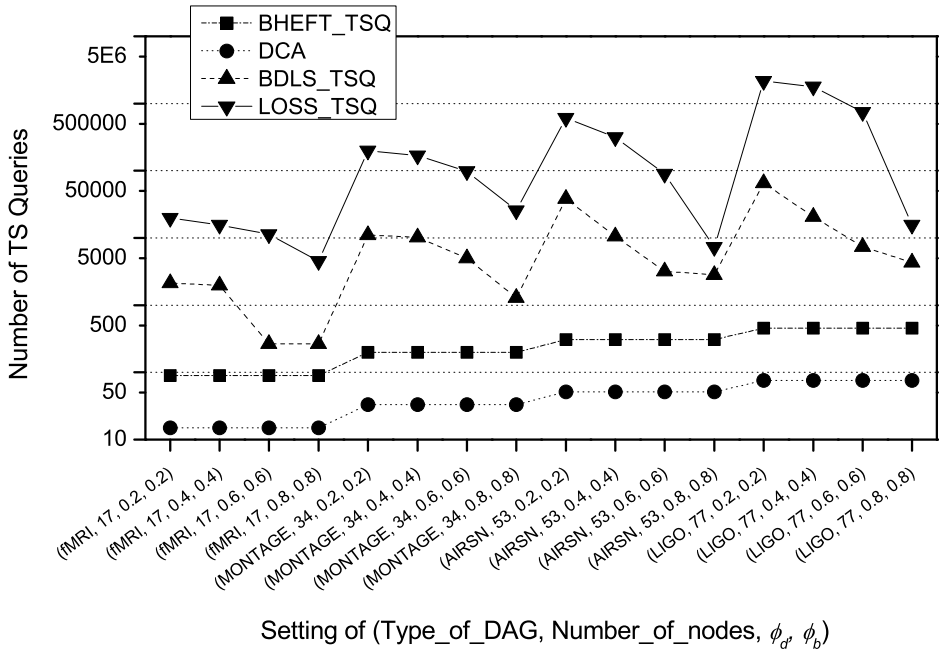


Figure 5.5: Number of TS Queries needed by each heuristic over diverse types and sizes of DAG and user constraints

Figure 5.6 shows how the running time of each heuristic varies over diverse cases of DAG and constraint settings. It is not surprising that, in most of the cases, LOSS has the highest time costs due to the communication overhead caused by numerous TS Queries. It can be easily imagined that some other sophisticated heuristics, such as DCA and genetic algorithm, if using the TS Query when scheduling, may need more time costs compared to LOSS. Actually, even LOSS is unscalable to large applications and too time consuming for on-line workflow planning. Although not using the TS Query, the DCA considered in the experiment still has a time cost comparable to BDLS, and this is significantly higher than

BHEFT. The latter two algorithms are both based on list scheduling, whereas BHEFT needs evidently less time costs than BDLS due to simpler computation and the fact that less communication is needed when making scheduling decisions. Moreover, BHEFT is the most scalable in terms of the growth of DAG size (and potentially the number of resources which is considered constant in this experiment). As can be seen in the graph, when scheduling LIGO application with 77 nodes on 6 resources, BHEFT only needs around 0.2 seconds on average. This validates the fact that BHEFT will be able to cater to the real-time requirements of online workflow planning.

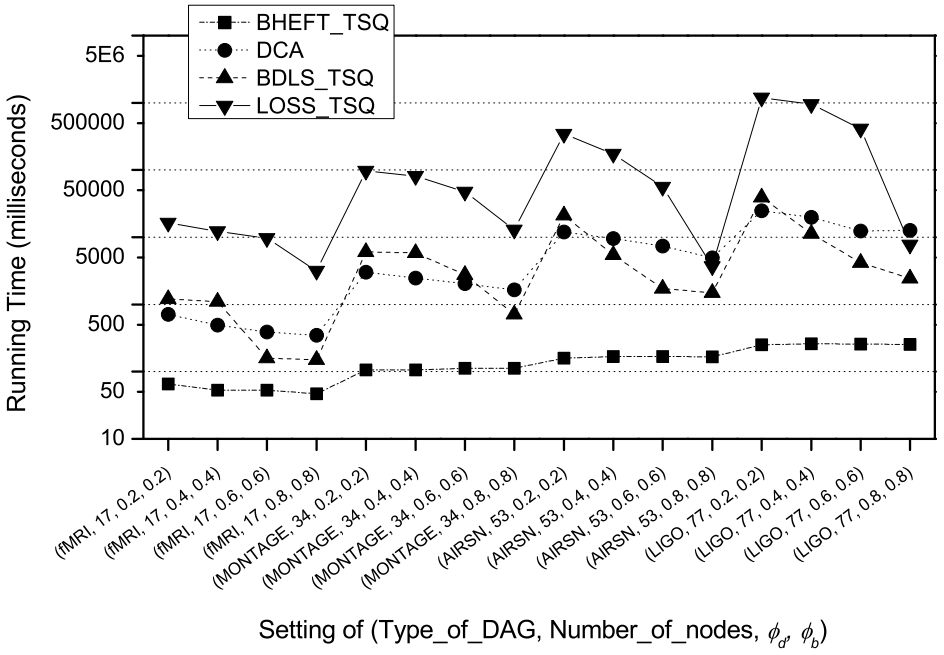


Figure 5.6: Time cost for each heuristic over diverse types and sizes of DAG and user constraints

Summary of observations. The following observations can be made from the above-described experimental results:

- The existing load of resources may have a significant impact on the BDC-planning. Directly applying those heuristics without considering the existing load of resources in job planning (for example, DCA) may result in

the unacceptable degradation of *PSR*. In contrast, BHEFT, which takes the existing load of resources into account in planning, is able to achieve a significant improvement on the success rate of finding a BDC-plan which simultaneously satisfies deadline and budget constraints.

- Some guided local search heuristics (for example, LOSS) may be too sensitive for the existing load of resources and cannot perform reasonably well for BDC-planning, even when the existing load of resources is considered when making the planning decision.
- In the context of BDC-planning, the simple list scheduling bi-criteria heuristics (for example, BHEFT and BDLS) may be as effective as the sophisticated heuristics based on an extensive local search, such as DCA.
- With low running costs, BHEFT is indicated to be competent for the real-time requirement of BDC-planning.

5.6 Closing Remarks

BDC-planning is required to establish an SLA in order to provide a certain level of QoS for workflow execution in the market-based Grid. This chapter proposes BHEFT, a novel low-cost bi-criteria heuristic based on HEFT, to fulfill the specific requirements of BDC-planning. The experimental results suggest that BHEFT can transcend the existing solutions on finding a BDC-plan. On the one hand, BHEFT requires little communication when making planning decisions based on the existing load of resources. In contrast, many of the complex algorithms (e.g., DCA) may either produce an unacceptable extra overhead to consider the existing load of resources for planning, or obtain a worse planning result than BHEFT without considering that. On the other hand, BHEFT exhibits a competitive performance on BDC-planning. Although some less complex planning algorithms (e.g., LOSS and BDLS) may cost acceptable communication overhead (but still significantly higher than BHEFT) to make use of the information about the existing load of resources, they do not perform as well as BHEFT to maximize the chance to find a BDC-plan. In summary, BHEFT appears to be at least as effective, or even more so than other sophisticated bi-criteria workflow scheduling heuristics, and has the lowest time costs and the best scalability. It is also indicated that BHEFT can effectively and efficiently find a BDC-plan under

various circumstances of constraints. This enables a quick judgement of whether or not the submitted user request is acceptable, and provides the feasibility of automating the creation of an SLA over diverse specified user constraints. Based on this chapter's work, the next chapter considers the overestimation of task execution time in BDC-planning to cope with prediction uncertainty, and makes advance reservations (as a term in the SLA) for the planned workflow according to the planning result.

Chapter 6

SLA-based Advance Reservation

The previous chapter provided an efficient BDC-constrained planning heuristic to help the SLA-based scheduling system to determine whether or not it is feasible to accept a workflow request with budget-deadline constraints. Using this knowledge, this chapter considers the problem of how to appropriately make an advance reservation for a dynamically submitted workflow if the workflow can be accepted. Given a set of workflows which may be submitted by different users over time, the advance reservation is aimed to not only enhance the guarantee of users' deadline requirement, but also maximize the profit service providers can earn from these workflows. Here, the advance reservation is considered at task level, i.e., one advance reservation is made for each individual task. This chapter investigates the performance of various advance reservation strategies which attempt to add some 'extra time' to each task reservation in order to improve service reliability in a situation in which actual task execution time is unpredictable. The key objective of this investigation is to evaluate the effectiveness of these advance reservation strategies on the maximization of SLA acceptance, resource utilization and service provider income, although these metrics may be conflicting. The evaluation results suggest how the 'extra time' ought be appended to task reservations in order to both provide the QoS guarantee for users and maximize the profit for service providers.

The main contribution of this chapter is the evaluation of several efficient advance reservation strategies which can automate the process including the planning and making of an appropriate advance reservation for a submitted workflow. The evaluation indicates the impacts these strategies may have on various aspects of the system performance and determine the strategy which should be adopted.

This is of great importance as a substantial step toward realizing an efficient and effective automatic generation of SLAs for an SLA-based scheduling system.

6.1 Background

As mentioned in Chapter 4, advance reservation is considered in the proposed SLA-based scheduling system as a viable alternative to the queue-based scheduler [MSK+04] which may lead to unacceptable uncertainty in guaranteeing users' QoS requirements. When a user submits a workflow request with hard budget and deadline constraints, before making advance reservation for the workflow, the SLA-based scheduling system needs to plan the workflow to see if it is possible to meet the user's constraints. If the planner manages to generate a plan which satisfies the constraints (i.e., BDC-plan), the workflow request will be accepted and the sequent advance reservation will be made. The BDC-planning problem has been addressed in Chapter 5, where planning decisions are made based on the estimated execution time of each task.

Given a plan for a workflow, it appears to be straightforward to make the counterpart advance reservations, i.e., each task is allocated a specific resource according to the mapping result in the plan and reserved with the time slot specified in the plan. Also, the reserved duration for each workflow task is considered to be the same as the estimated task execution time. However, if a task is reserved in such a manner, some issues may be raised in terms of meeting the user's deadline constraint due to the inevitable uncertainty of performance prediction. It can be imagined that when the execution time of a task turns out to be significantly longer than the static estimate (which is not unusual in practice), according to the agreed advance reservation, the task will probably be terminated if no re-negotiation or some pre-defined recovery mechanism is considered [ZS06a]. When this situation happens to a DAG application, terminating a task may mean the cancellation of all of its child tasks, which may lead to the failure of the whole application.

In order to minimize the risk of this kind of failure, extending some extra time to the advance reservation of a task can be a common and intuitive idea to improve the reliability of task execution. In an extreme case where only one single application is considered and no constraint is imposed, it may be trivial to investigate this extension since it can simply be enlarged as much as the user wants

to maximize the reliability of task execution. However, in the context of market-based grids, where there are multiple workflows being submitted dynamically over time with budget and deadline constraints, and competing for resources, extending the reserved duration of tasks may become a challenging issue. It can be imagined that this extension may have a significant impact on multiple functional objectives, such as the rate of accepting a workflow request, the rate of successful execution of an accepted workflow, and resource utilization, which may consequently result in unforeseen changes in the profits for the service providers.

There have been quite a few efforts to develop an advance reservation technique to achieve the QoS for grids [SVF06, NBB07, CRH07, CRH08]. However, to the best of the author's knowledge, none of these studies focus on how to appropriately increase the reliability of task execution for workflows which have hard constraints of budget and deadline and are executed in an environment where the performance prediction is inaccurate, so that not only the user's QoS requirements can be met, but also the profits for the service providers can be maximized. This problem is the main focus of this chapter. Based on the BDC-planning heuristics discussed in the previous chapter, several advance reservation strategies are designed in an attempt to reserve a time slot longer than the estimated execution time of each task. In simulation, these strategies are respectively applied to a set of multiple workflows which are sequentially submitted to the system at random times, and their impact on the system and user performance is investigated. The results suggest how, and to what extent, the reserved duration for each workflow task should be extended in various circumstances.

The remainder of the chapter is organized as follows. The related works on advance reservations for grid systems are first briefly described in section 6.2. Next, an overview of the problem is presented in section 6.3. Section 6.4 introduces the proposed advance reservation strategies. The simulation methods used in the evaluation are presented in section 6.5 and the results and discussion are provided in section 6.6. At last, section 6.7 concludes the chapter.

6.2 Related Work

Wide attention has been given to advance reservation studies in the grid community. One of the pioneering works is presented in [FKL+99] where a basic architecture with a set of APIs are defined for manipulating the advance reservations of

distributed resources. Quite a few resource management systems and schedulers have evolved to support advance reservation [Mac03], for instance LSF [LSF], Maui [JSC01], EASY [SCZL96] and COSY [CZ04]. Although advance reservation, in the state-of-the-art, may have a negative impact on system and user performance [SFT00], it can be seen that, during the recent years that advance reservation has been widely employed and investigated with different concerns, such as scheduling [SFT00, BV04, YS06], resource management [MKB02, ET05], provisioning for performance predictability [MAD+05, SKD05, WSV+06], optimizing or guaranteeing quality of service [SVF06, SKD07, NBB07, CRH07, CRH08] and exploiting the flexibility of advance reservation [RSW04, FMP05, RSR06] etc.

Essentially, this chapter focuses on improving reliability of task execution for workflows by extending the reserved duration for each task, in order to guarantee the user's deadline requirement, meet the budget constraint, and maximize the service providers' profits. In terms of optimizing multiple QoS metrics, the authors of [SVF06] define a utility function which provides a numerical expression of various QoS constraints, and propose an approach to generate advance reservation offers based on a 3-layer negotiation. In this study, each application task is considered to have its own constraints and the reservation offers of each task are generated separately by negotiation with the resource provider. In contrast, the individual workflow tasks considered in this chapter do not have separate constraints. Instead, the user just needs to simply specify the budget and deadline constraints for the whole workflow. Another negotiation-based advance reservation strategy is proposed in [MAD+05], where the focus is to optimize the time metric and the monetary cost is not considered. Another study of advance reservation focusing on multi-objective optimization can be found in [SKD07], where a workflow is considered to be reserved as whole. This differs from the work of this chapter which makes advance reservations separately for each individual task. The work presented in [CRH07] and [CRH08] employs computational geometry techniques to plan advance reservation for homogeneous and heterogeneous systems respectively. The algorithms are aimed to schedule multiple tasks in order to meet their start time and deadline constraints and maximise system utilization. In these studies, the application is modelled as a single task other than a DAG and, again, only time constraint is considered.

None of the aforementioned studies consider adding extra time to task reservation to increase task execution reliability. The most relevant work is available

in [ZS06a], where two policies are proposed to plan advance reservations for the individual tasks of a DAG on a heterogeneous platform with considering deadline constraint. The described approach consists of two phases, in the first of which, some makespan-optimized scheduling heuristic such as HEFT [THW02] and HBMCT [SZ04a] was used to generate an initial schedule based on a static task runtime estimate. In the second phase, the proposed policies were used to determine how to distribute the Application Spare Time (AST), i.e., the time difference between the expected finish time of the initial schedule and the deadline, into spare time for tasks, then extend the task spare time for each planned task respectively. It was demonstrated that such an approach managed to effectively reduce the risk of application failure with task runtime changes. In addition, it made advance reservation in the grain of the task instead of the whole application in order to avoid the waste of resource utilization in the latter case. However, the advance reservation policies introduced in [ZS06a] are not directly applicable in market-based Grids because these policies have three main limitations:

1. The existing load of the system is not considered. Suppose that some advance reservations have been made for other applications and a resource can only run one task at a time at most. Any advance reservation planning without considering the existing load will probably result in reservation conflict (i.e., it will overlap with the existing reservation).

2. The budget constraint is not considered. An initial schedule which meets the budget-deadline constraints may be obtained by using a multi-objective planning heuristic. Following the policies proposed in [ZS06a], the reserved time for every task may be extended without violating the deadline constraint. However, the time extension is unlikely to be free of charge in a practical business scenario. Therefore, this approach will probably introduce extra economic costs, and consequently exceed the budget constraint.

3. Only a single-workflow scenario is considered. Due to the competition of resource between applications, an excessively extended reservation for the current application will probably lead to a heavy existing load of resource for newly planned applications and make them more difficult to be successfully planned. The impact of introducing extended task reservation into planning among multiple workflows is not investigated in [ZS06a].

6.3 Problem Description

This section identifies the SLA-based advance reservation problem to be addressed in this chapter. An overview of the problem is provided in Section 6.3.1 and more details about the problem model are presented in Section 6.3.2. Table 6.1 as shown below, lists the notations used in this chapter in conjunction with Table 5.1.

Symbol	Meaning
p_i	final price for running task i
ρ_i	the profit that can be earned from running task i
β_p	cost unit for resource p
$\Delta_{i,p}$	extended duration in task reservation
χ	extension rate in task reservation
δ	quality of task execution time estimation
ET_w	information about the task execution time estimation for workflow w
$\mathcal{H}()$	a budget-deadline-constrained-planning heuristic
λ	extension price ratio
ϕ	constraint ratio

Table 6.1: Notations

6.3.1 Problem Overview

It is assumed that there is a set of workflows W, each of which (denoted by w) can be represented by a DAG. With a specified budget Bgt_w and a deadline Ddl_w, each workflow w is submitted at a random time to a system which, the same as depicted in Section 5.3.1, consists of multiple heterogeneous resources and provides different services. The estimation of the execution times of all tasks of w is known and denoted by ET_w, while an actual execution time may differ from its estimated value. As shown in Figure 6.1, the lifecycle of each workflow is depicted as follows.

After being *submitted*, the workflow is *planned* based on the estimation of task execution time and the existing load of resources to see if the specified budget and deadline constraints can be met. If no, the workflow is *rejected*. Otherwise, the workflow tasks are planned again by an advance reservation strategy which normally considers a reserved duration longer than the execution time of each task. If the strategy can obtain a reasonable advance reservation plan, by which

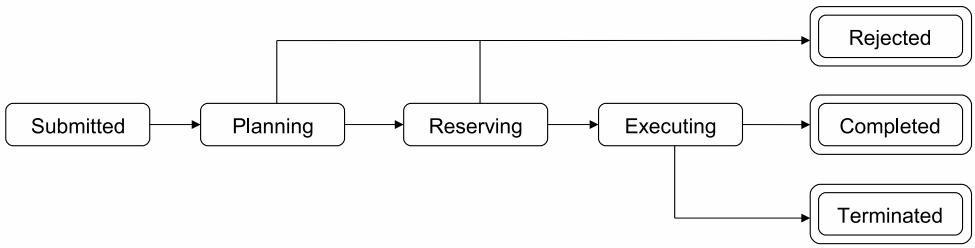


Figure 6.1: Lifecycle of a workflow in the SLA-based scheduling system model

both budget and deadline constraints are still met and to some extent, the reserved duration of each task is extended from the execution time estimation, the workflow is *reserved* according to the plan. Otherwise, it is *rejected*. Each task of the workflow begins to be executed at the start time specified in its advance reservation. If any of the tasks cannot be completed before the end of the reserved duration, the workflow is *terminated* and the advance reservations of the remaining tasks are cancelled. Otherwise, the workflow is successfully *completed*, and this usually means that the user's QoS requirements are satisfied. If a workflow is rejected or terminated, the user pays nothing. Otherwise, the user pays a price based on the resource usage, and the service providers can earn profits from this payment.

With the focus on such a scenario, the problem is how the advance reservation strategy should generate a reasonable advance reservation plan as mentioned before, so that the overall profits for the service providers can be maximized.

In order to ensure that the budget and deadline constraints can still be met after the extension of the reserved duration for each task, the BDC-planning heuristics can be a natural approach for the employment of the advance reservation strategy. As shown in the previous chapter, a BDC-planning heuristics attempts to plan a DAG based on a given estimation of task execution time to see if the specified constraints can be satisfied. Using such a heuristic, the mission of the advance reservation strategy is to determine an appropriate overestimation of the task execution time. To illustrate this, a motivation example is provided in Figure 6.2, where a simple workflow w with 4 tasks and QoS constraints are initially planned based on the estimated execution time of each task as demonstrated in Figure 6.2(a). The problem is how to determine the extension to the time estimation, which is denoted by $\Delta_{i,p}$ for task i on resource p. Figure 6.2(b) provides a possible extension without violating the budget and deadline constraints.

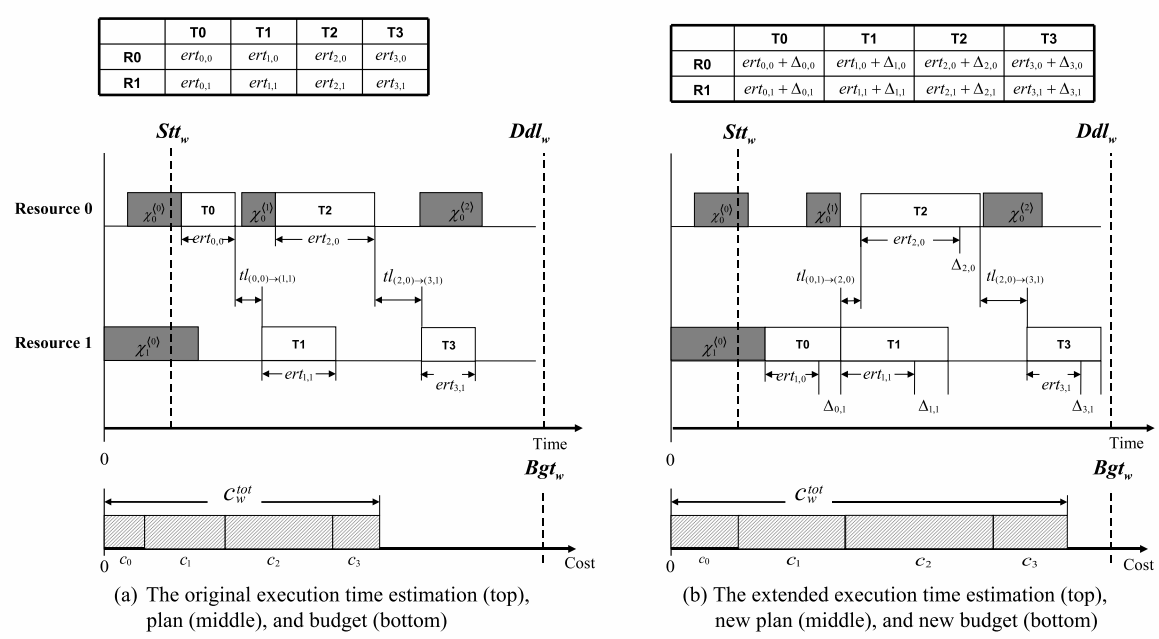


Figure 6.2: A Motivation Example

In order to formulate the above, a function $\mathcal{H}$ is defined to represent a Budget-Deadline Constrained (BDC) workflow planning heuristic which can generate a BDC-plan Sch_w for a workflow w based on a given estimation of task execution times (i.e., ET_w) and other needed information such an information about workflow w, resource set $\mathcal{R}$ and service set $\mathcal{S}$, the existing loads of resources etc. With the focus on the input of performance estimation, function $\mathcal{H}$ is denoted as below:

$$Sch_w = \mathcal{H}(ET_w) \tag{6.1}$$

If $\mathcal{H}$ successfully find a schedule meeting user's QoS requirements, $Sch_w \neq \emptyset$. Otherwise, $Sch_w = \emptyset$. As a successful planning result, each task τ_i is reserved with a reserved duration $rd_i = ert_{i,p}$ to the mapped resource γ_p. An extended estimation for task execution time is denoted by $ET_w^{\Delta} = \{est_{i,p} + \Delta_{i,p} : \tau_i \in \mathcal{T}, s_{i,p} \in \mathcal{S}_i\}$, where $\Delta_{i,p} > 0$ is the extended time for each task execution time prediction. Given that there are multiple workflows submitted by users with budget and deadline constraints over time, the problem this chapter considers is how to appropriately extend the task execution time estimation in BDC-planning (namely, to find an appropriate ET_w^{Δ} to generate the advance reservation schedule

Sch_w^{Δ})

$$Sch_w^{\Delta} = \mathcal{H}(ET_w^{\Delta}) \tag{6.2}$$

and then make advance reservations according to Sch_w^{Δ} (if $Sch_w^{\Delta} \neq \emptyset$) for each submitted workflow, so that the guarantee of users' QoS requirements can be enhanced and service providers' benefits can be promoted.

6.3.2 Problem Modelling

As an extension of the BDC-planning problem, most of the concepts modelled in Section 5.3.2 can be reused in this chapter. The only two distinctions in problem modelling are presented as follows:

- *Time Issues.* Suppose task τ_i is being mapped to resource γ_p. In the previous work of BDC-planning, it was assumed that the reserved duration for each task was the same as the estimation of task execution time, i.e., $rd_i = ert_{i,p}$. In this chapter, account is taken of the uncertainty in performance prediction, and thus, the actual task execution time is modelled as a random variable. For each task, the actual execution time is assumed to be randomly distributed within a certain bound around the estimated value. The notion of *Quality of Estimation* (QoE, denoted by δ) is used to describe the upper bounds of deviation which the actual task execution time may have in respect of the estimated value. Namely, the actual execution time $act_{i,p}$ falls within the range $[(1-\delta)ert_{i,p}, (1+\delta)ert_{i,p}]$, where $0 \leq \delta < 1.0$. As already mentioned, the reserved duration for a task is extended from the execution time estimate, i.e., $rd_i = ert_{i,p} + \Delta_{i,p}$. The successful running of a workflow is defined as being that all of the reserved tasks of the workflow begin at the planned start time and complete before the planned end time, i.e., $\forall \tau_i \in w, act_{i,p} \leq rd_i$.

- *Accounting Issues.* In the previous chapter where the overestimation of the task execution time is not considered in advance reservation planning, Eq.(5.2) is used to compute the economic cost of a planned task. In this chapter, due to the extension considered of the reserved duration for each task, double rates are adopted in the price setting. In detail, given a task τ_i with a reserved duration of $rd_i = ert_{i,p} + \Delta_{i,p}$ on resource γ_p, there is

$$c_i = ert_{i,p} \cdot \mu_p + \Delta_{i,p} \cdot \lambda \cdot \mu_p \tag{6.3}$$

where μ_p is the price unit accounting for the charge during the estimated execution time, $(\lambda \cdot \mu_p)$ is the price unit accounting for the charge during the extended time in the reserved duration, and the *extension price ratio* $0 < \lambda < 1$, which can be pre-specified, defines the ratio between these two price units. To compute the final charge for a workflow, the cost-plus pricing described in Section 4.3.5 is adopted. With overestimation, the final price for a successful service provision is computed by

$$p_i = \begin{cases} act_{i,p} \cdot \mu_p & : \quad 0 < act_{i,m} \leq ert_{i,m} \\ ert_{i,p} \cdot \mu_p + \lambda \cdot \mu_p (act_{i,p} - est_{i,p}) & : \quad est_{i,p} < act_{i,p} \leq rd_i \end{cases} \tag{6.4}$$

while the profit earned (i.e., ρ_i) according to different execution results is computed by

$$\rho_i = \begin{cases} p_i - act_{i,m} \times \beta_m & : \quad \text{If the workflow is successfully completed} \\ -act_{i,m} \times \beta_m & : \quad \text{Otherwise} \end{cases} \tag{6.5}$$

where β_m is the cost unit as defined in Section 4.3.5. It is assumed that for each resource γ_p, the μ_p and β_p are both pre-specified. The standardization of μ_p can be found in Section 5.5.1, i.e., $\mu_p = \frac{\alpha_p(1+\alpha_p)}{2}$, and in this chapter, it is assumed that $\beta_p = \frac{\alpha_p}{2}$.

6.4 Advance Reservation Strategies

This section discusses how to appropriately extend the task execution time estimation in BDC-planning and advance reservation for a submitted workflow. In terms of the problem defined in Section 6.3, this can be formulated as below:

Problem Formulation. Given the task execution time estimation $ET_w = \{ert_{i,p} : \tau_i \in w, \gamma_p \in \mathcal{R}\}$ and a BDC-planning heuristic $\mathcal{H}$, rather than computing $Sch_w = \mathcal{H}(ET_w)$, an appropriate $ET_w^{\Delta} = \{ert_{i,p} + \Delta_{i,p} : \tau_i \in w, \gamma_p \in \mathcal{R}, \Delta_{i,p} > 0\}$ needs to be determined to obtain $Sch_w^{\Delta} = \mathcal{H}(ET_w^{\Delta})$.

It is deemed to be unnecessary to prioritize different workflow tasks in the extension, since every task has the same probability to cause workflow running failure. This is suggested by two assumptions: (i) the Quality of Estimation is the same for all unpredictable task execution times; and (ii) any task running beyond the reserving duration will result in a workflow running failure. Therefore,

extending each $ert_{i,p}$ by the same percentage is considered, and the *extension ratio* χ is defined as below:

$$\chi = \frac{\Delta_{0,0}}{ert_{0,0}} = \frac{\Delta_{0,1}}{ert_{0,1}} = \cdots = \frac{\Delta_{j,p}}{ert_{j,p}} = \cdots = \frac{\Delta_{|\mathcal{T}|-1,|\mathcal{R}|-1}}{ert_{|\mathcal{T}|-1,|\mathcal{R}|-1}} \tag{6.6}$$

whereby the problem can be simplified. In order to find an appropriate ET_w^{Δ}, the corresponding extension ratio χ_w needs to be found.

Four advance reservation strategies are considered, i.e., *Greedy Extension*, *Rigid Extension*, *Conservative Extension*, and *Progressive Extension* to determine the extension ratio χ for a BDC-planning heuristic $\mathcal{H}$ to plan a single workflow and make advance reservations accordingly.

6.4.1 Rigid Extension

Input:
A submitted workflow application w.
The estimation for task execution times for the workflow ET_w.
Output:
Advance reservation result for w.

1: Obtain the evaluation of QoE, i.e., δ_{est};
2: Let the extension ratio $\chi_w = \delta_{est}$;
3: Obtain ET_w^{Δ} by extending ET_w using χ_w;
4: Compute $Sch_w^{\Delta} = \mathcal{H}(ET_w^{\Delta})$ as defined in Eq.(6.2);
5: **If** $Sch_w^{\Delta} \neq \emptyset$ *then*
6: Make advance reservation for each task of w according to Sch_w^{Δ};
7: **Return** Sch_w^{Δ}.
8: **Else**
9: **Return** $\emptyset$.
10: **Endif**

Figure 6.3: The outline of Rigid Extension Strategy

As described in Figure 6.3, the *Rigid Extension* (RX) is simple and totally relies on the estimation of QoE (as defined in Section6.3.2) to determine the extension ratio χ_w. Suppose that the RX strategy evaluates the QoE to be δ_{est}, which can be the same as, or different from, the actual QoE (i.e., δ_{act}). From RX's point of view, in order to ensure every task τ_i on resource γ_p can be successfully executed, the reserving duration rd_i should be at least $(1 + \delta_{est}) \cdot ert_{i,p}$. Thus, the RX strategy simply lets $\chi_w = \delta_{est}$ and obtains ET_w^{Δ} by extending the initial estimation of task execution times ET_w using χ_w. Namely, $ET_w^{\Delta} = \{ert_{i,p}^{\Delta} =$

$ert_{i,p}(1+\delta_{est}) : ert_{i,p} \in ET_w\}$. Then ET_w^{Δ} can be used by a BDC-planning heuristic to obtain the output schedule.

6.4.2 Greedy Extension

Input:
A submitted workflow application w.
The estimation for task execution times for the workflow ET_w.
Output:
Advance reservation result for w.

1: Obtain an initial schedule by $\mathcal{H}$, i.e., $Sch_w = \mathcal{H}(ET_w)$ as defined in Eq.(6.1);
2: **If** $Sch_w = \emptyset$ **then return** $\emptyset$
3: Let $\chi_{suc} = 0$;
4: Compute $\chi_{cur} = \frac{Ddl_w - \mathcal{M}(Sch_w)}{\mathcal{M}(Sch_w)}$, where $\mathcal{M}(Sch_w)$ means the planned end time of w according to Sch_w;
5: **Repeat**
6: Obtain ET_w^{Δ} by extending ET_w using the extension ratio χ_{cur},
7: Compute $Sch_w^{\Delta} = \mathcal{H}(ET_w^{\Delta})$ as defined in Eq.(6.2);
8: **If** $Sch_w^{\Delta} \neq \emptyset$ **then**
9: Let $\chi_{suc} = \chi_{cur}$;
10: Compute $\chi_{cur} = \frac{Ddl_w - \mathcal{M}(Sch_w^{\Delta})}{\mathcal{M}(Sch_w^{\Delta})} + \chi_{suc}$;
11: **Else**
12: Compute $\chi_{cur} = \frac{\chi_{cur} + \chi_{suc}}{2}$.
13: **Endif**
14: **Until** the specified maximum iteration is reached;
15: Make advance reservation for each task of w according to the latest $Sch_w^{\Delta} \neq \emptyset$.
16: **Return** the latest $Sch_w^{\Delta} \neq \emptyset$.

Figure 6.4: The outline of Greedy Extension Strategy

Without considering the QoE, the *Greedy Extension* (GX) strategy attempts to maximize the extension ratio for the currently planned workflow as long as the specified QoS constraints and the spare capability of the resources allow. The GX strategy proposed here (as shown in Figure 6.4) is inspired by the idea of a bisection method [Cor77] which repeatedly bisects an interval before selecting a subinterval for further processing. The GX strategy first tries to find a BDC-plan by applying a BDC-planning heuristic $\mathcal{H}$ without any extension to ET_w (Line 1). If such a plan cannot be found, the strategy stops, having failed. Otherwise, the strategy continues to initialize two extension rate variables χ_{suc} (Line 3) and χ_{cur} (Line 4) and starts iteration (Line 5-14). In each loop, the estimation of the task execution time is extended by χ_{cur} and the extended result is used in the BDC-planning heuristic to obtain a new schedule (Line 6-7). χ_{suc} and χ_{cur} are updated

accordingly depending on whether or not the new schedule is a BDC-plan (Lines 8-13). The iteration terminates when it reaches the specified maximum, and then the last found BDC-plan is returned as the output (Lines 14-15).

6.4.3 Conservative Extension

The *Conservative Extension* (CX) strategy is a mixture of the RX and the GX strategies. The CX strategy firstly employs the RX strategy to see if a BDC-plan can be found. If it can, the workflow is reserved by the planning result as the output. Otherwise, the GX strategy is invoked to obtain the output schedule. It can be imagined that, in this way, CX will never produce an overestimation rate higher than that of RX.

6.4.4 Progressive Extension

The *Progressive Extension* (PX) strategy is also a mixture of the RX and the GX strategies. The PX strategy firstly employs the RX strategy to see if a BDC-plan can be found. If not, the extension terminates with a workflow rejection. Otherwise, the GX strategy is invoked. Then, the overestimation ratios obtained by RX and GX are compared, and the planning result with the higher overestimation ratio is returned as the output. It can be imagined that, in this way, PX will never produce an overestimation rate lower than that of RX.

6.5 Experimental Methodology

In this section, the effectiveness of the four aforementioned advance reservation strategies is evaluated in a multiple-workflow scenario. Suppose that a set of workflows are sequentially submitted for processing during the period of $[a, b]$, four metrics are considered as follows:

- Acceptance Rate (AR): $AR = \frac{\text{Number of Accepted Workflows}}{\text{Number of Arrived Workflows}} \times 100\%$
- Success Rate (SR): $SR = \frac{\text{Number of Successfully Executed Workflows}}{\text{Number of Accepted Workflows}} \times 100\%$
- Utilization Rate (UR): $UR = \frac{\text{Sum of the Periods of Task Running}}{\text{Number of Resource} \times (b-a)} \times 100\%$
- Total Profit (ρ_W^{tot}): $\rho_W^{tot} = (\sum_{w \in W} \sum_{\tau_i \in w} \rho_i)$, where W denotes the set of multiple workflows; ρ_i is the profit service providers can earn from executing $\tau_i \in w$.

6.5.1 Simulator

In order to make the evaluation, a simulator was implemented which could emulate the arrival, planning, advance reservation and execution of a bunch of multiple workflows which were submitted sequentially with QoS constraints. It is worth mentioning that no rescheduling is considered in local resource, namely, the time at which a workflow task actually begins execution is considered to be the same as the reserved start time. The behaviour of the broker and several LRMs, each of which manages a set of heterogeneous resources, was also simulated in the simulator. The steps the simulator needs to follow to emulate the processing of a set of workflows W sequentially arriving during the time of $[a, b]$ are briefly described below:

1. *Step I: Constructing Environment.* A computing environment consisting of a certain number of LRMs, a set of heterogeneous resources managed by the LRMs, a set of provided services located on the resources, and a broker is constructed. A configuration of resources and services (for example, resource power ratio, transmission rate, service price, service type etc.) is randomly generated. For each simulated resource, a reservation queue is maintained to record the advance reservation made on this resource. Each simulated LRM can manipulate the reservation queues of the resources managed by this LRM. For example, the LRM can add a reservation, remove a reservation, lookup a reservation, or move a reservation (i.e., change the start time of the reservation) in the managed reservation queues. A broker is created, being able to communicate with all LRMs. The broker can send various requests (for example, querying free time slots, making advance reservation etc.) to any LRM, and the LRM receiving the request can manipulate the managed resources and their reservation queues accordingly.

2. *Step II: Generating Workflow Stream.* For each workflow $w \in W$, the workflow profile is randomly generated following a set of specified parameters (for example, DAG type, number of tasks etc.), the budget and deadline constraints associated with w are generated accordingly, and the arrival time of w is randomly generated and time-stamped.

3. *Step III: Processing Workflow Stream.* From time a through b, for each time unit t^*, the simulator firstly checks if the time-stamped event (I) occurs. If it does, the corresponding actions are taken. Then the simulator goes

through every resource and checks whether one of the time-stamped events (II)-(IV) occurs. If it does, the corresponding actions are taken:

- *Time Stamped Event (I): Workflow Arrival*, which is indicated by t^* equals the arrival time of one of the time-stamped workflows. If a new workflow arrives, one of the proposed advance reservation strategies is invoked by the broker to plan the workflow. If the planning succeeds, the workflow is accepted and its tasks are reserved according to the planning result. The existing reservations on relevant resources are updated accordingly. The start and end of task reservation are time-stamped. The actual end time of each task is randomly generated according to the actual QoE, and is also time-stamped.
- *Time Stamped Event (II): Execution Start*, which is indicated by t^* equals the start time of a task reservation in a resource. If this occurs, the state of the reservation is set to RUNNING.
- *Time Stamped Event (III): Early Completion*, which is indicated by t^* equals the actual end time of a task reservation in a resource. If this occurs, it is implied that the task has been completed earlier than the planned end time and thus it has been completed successfully. The task reservation is removed and the relevant metrics are measured and recorded.
- *Time Stamped Event (IV): Execution Failure*, which is indicated by t^* equals the planned end time but is less than the actual end time of a task reservation in a resource. If this occurs, it is implied that the task has terminated having failed. For each remaining task reservation for the workflow that this task belongs to, if the reservation is RUNNING, the running task is terminated (i.e., the reservation is removed) and the relevant statistic is recorded. Otherwise, the reservation is removed without updating the statistic.

4. *Step IV: Recording results.* The simulator collects the statistics on the above-mentioned four metrics.

In terms of the simulation, several assumptions are made, as follows:

- During one time unit, one workflow may arrive at most;

- The workflow arrival can be modelled as a Bernoulli Process;
- The earliest start time for every workflow (i.e., Stt_w) is at the next time unit after its arrival;
- All actions reacting to the time-stamped events as stated above can be completed within one time unit.

6.5.2 Experimental Setting

In this experiment, the simulator simulated 2 LRMs, each of which managed 3 resources, namely 6 heterogeneous in total. It should be noted that, although the simulated system has a small number of resources compared to a realistic Grid, it is indicated in [NWB08, NWB09] (where advance reservation is applied to a set of distributed HPC resources with multiple processors) that the present work could be extended to scheduling for a wider Grid. For the setting of resources, the power ratio α for each resource was randomly generated from a range of [0.5, 2.0], and the transmission rate between each two different resources was randomly generated from [0.5, 1.5]. The price unit of the benchmark resource was assumed to be $\mu^* = 1.0$, and then the price unit of all resources could be computed respectively with their power ratio as defined in Eq.(5.13). The existing load of resources before the simulation started was assumed to be zero. There were 4 service types having a standard service time of 10, 15, 25 and 40 respectively. It was assumed that the actual QoE $\delta_{act} = 0.5$ and the actual execution times are normally distributed within the range specified by δ_{act}.

The DAG parameters considered in this experiment are the same as those described in Section 5.5.2. Given a specific DAG type, a bunch of DAGs of the same type can be randomly generated. To generate the random arrival time of each DAG the parameter *Application Arrival Delay* (AD) was used, which describes the mean interval between application arrivals.

For each generated DAG, the deadline and budget constraints were considered in the same way as presented in Section 5.5.2. Specifically, Constraint Ratio ϕ ($0 \leq \phi \leq 1.0$) was defined, and let $\phi_d = \phi_b = \phi$. In addition, it was assumed that the earliest possible start time for a submitted workflow was the next time unit to its submission.

Among the numerous parameters involved in this experimental setting, how

the four parameters (i) Application Arrival Delay (AD, Section 6.6.1), (ii) Constraint Ratio (ϕ, Section 6.6.2), (iii) DAG Type (DAG, Section 6.6.3), and (iv) Extension Price Ratio (λ, Section 6.6.4) affect the performance of different advance reservation strategies was particularly investigated. According to the description in Section 6.5.1, these parameters configure the emulation of processing a set of workflows W sequentially arriving during the time of $[a, b]$ as follows:

- Application Arrival Delay parameterizes the random generation of arrival time of the workflow in Step II.
- Constraint Ratio parameterizes the generation of the QoS constraints associated with workflows in Step II.
- DAG type and the relevant parameters parameterize the random generation of workflows in Step II.
- Extension Price Ratio configures the price setting in Step I.

More specifically, the simulator is used to emulate the processing of multiple workflows during the virtual time from 0 through 10000. When investigating the impact of a varying parameter, all other parameters were allowed to stay in their default. By default, AD=100, $\phi = 0.8$, DAG='Montage with 34 nodes', and $\lambda = 0.2$. Using this default setting, a heavy workload scenario was modelled, where 34-node Montage workflows arrived at average interval of 100, from time 0 through 10000. Generally, the modelled workload exceeded the resource capability. However, for each workflow, it was not difficult to find a schedule meeting budget and deadline constraints, since the default setting of constraint ratio and extension price ratio was quite relaxed.

The four proposed strategies were examined in this experiment. All strategies employed a heuristic $\mathcal{H}$=BHEFT, which was proved to be an efficient BDC-planning heuristic in the previous chapter. It is worth mentioning that, for RX, PX and CX strategies which rely on the estimation of QoE, their estimated QoE δ_{est} may differ from the actual QoE δ_{act} which is assumed to be 0.5. Taking this into account, the situation was also examined of when QoE is under-estimated (denoted by RX-, PX-, and CX-, and $\delta_{est} = 0.5\delta_{act}$) or over-estimated (denoted by RX+, PX+, and CX+, and $\delta_{est} = 1.5\delta_{act}$). For strategies denoted by RX*, PX* and CX*, it is assumed that $\delta_{est} = \delta_{act}$. In short, RXs is used to denote the set of RX, RX+ and RX-, and a similar rule is applied to CXs and PXs.

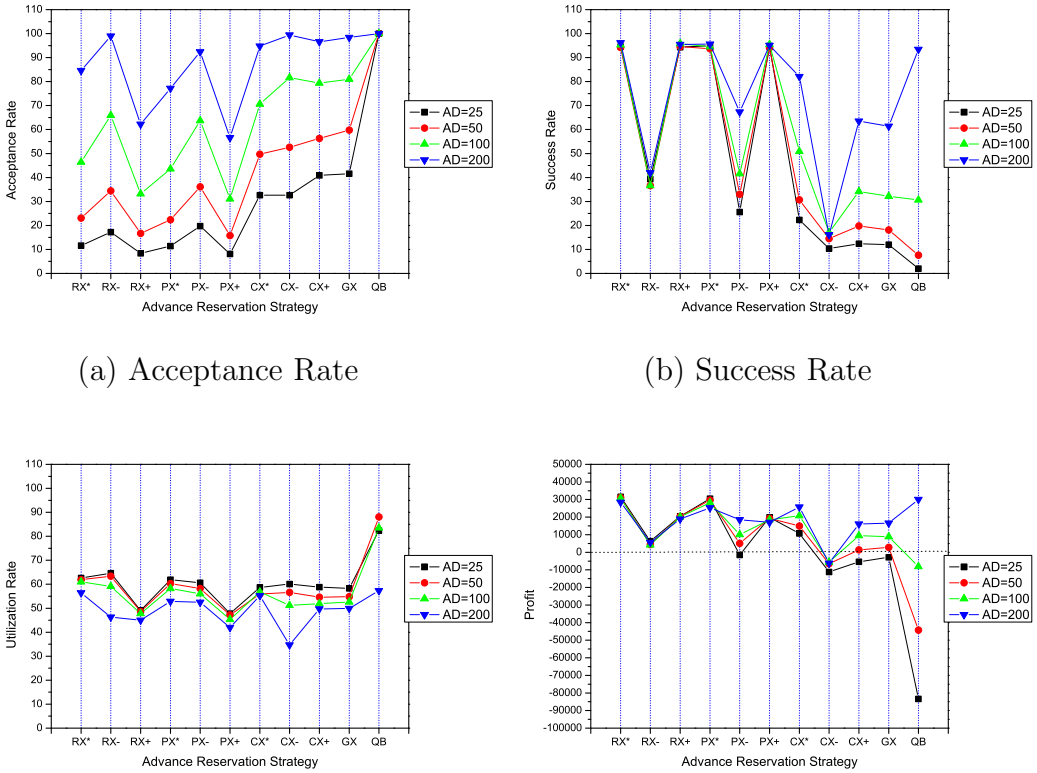
In addition, the performance of a Queue-based strategy (denoted by QB) is also examined, in which no advance reservation is considered. When QB is used, the queued task is allowed to sort by first-in-first-out. Moreover, when the simulator detects that a workflow exceeds its deadline, the workflow execution is judged to have failed and all of the remaining tasks are removed from the queue. Therefore, eleven competitors were examined in the simulation. In each simulation, the performance results for all examined competitors were recorded respectively. Due to the randomness of the experimental setting, in one experiment, the simulation was repeated 100 times and the average result was obtained for every aforementioned metric.

6.6 Evaluation Results

6.6.1 Impact of Application Arrival Delay

The application arrival delay was varied at 25, 50, 100 and 200 to model different workloads from heavy to light, and the evaluation results are presented in Figure 6.5.

Firstly, the impact of the varying AD on different metrics was observed. For the application acceptance rate (AR), the larger the AD was, the higher the AR of the majority of the advance reservation strategies would be. Actually, the total number of the accepted workflows, which was often limited by the capacity of the resources, did not change much. Therefore, the AR increased as the total number of submitted workflows decreased, which was indicated by a higher AD. QB was special. Without considering advance reservation, QB does not have to take into account the existing loads of resources. Therefore, the application acceptance rate for QB was always 100% since the constraint ratio was quite relaxed by default. For the success rate (SR), some strategies, including RXs, PX*, PX+ and CX-, were not as sensitive as others in respect of the variation of AD. As AD increased, there were fewer workflows for resource competition. Consequently, the extension ratio for PX-, CX+, and GXs may have become significantly larger and thus, the success rate rose. In the case of QB, the success rate increased because there were fewer tasks in the queue and therefore the delay in the queue was reduced. For the utilization rate, the value achieved by each advance reservation strategy varied slightly from AD=25 through AD=100, but decreased a bit when AD reached



(a) Acceptance Rate

(b) Success Rate

(c) Utilization Rate

(d) Profit

Figure 6.5: Impact of Application Arrival Delay (AD)

200. With advance reservation, the highest utilization rate could reach around 65%. In contrast, the utilization rate for QB was always higher than 80% when AD was equal to 25, 50 and 100, but dropped dramatically to 55% at AD=200. However, higher utilization rate did not necessarily mean higher revenue because the user's constraint was imposed in the calculation of payment. For the profit, the performance of RXs, PX* and PX+ remained almost the same from AD=25 through AD=100, and slightly degraded when AD=200. In contrast, PX-, CXs and GXs performed gradually better as AD became larger. QB suffered a big loss at AD=25, but the profit grew significantly as AD increased.

Secondly, the different profits earned by different strategies were compared. When AD was not greater than 100, all advance reservation strategies earned more profit than QB, and especially, with a small AD like 25 and 50, the performance of QB turned out to be a disaster. However, when AD reached 200, which meant fewer incoming workflows and fewer queued tasks, QB outperformed all

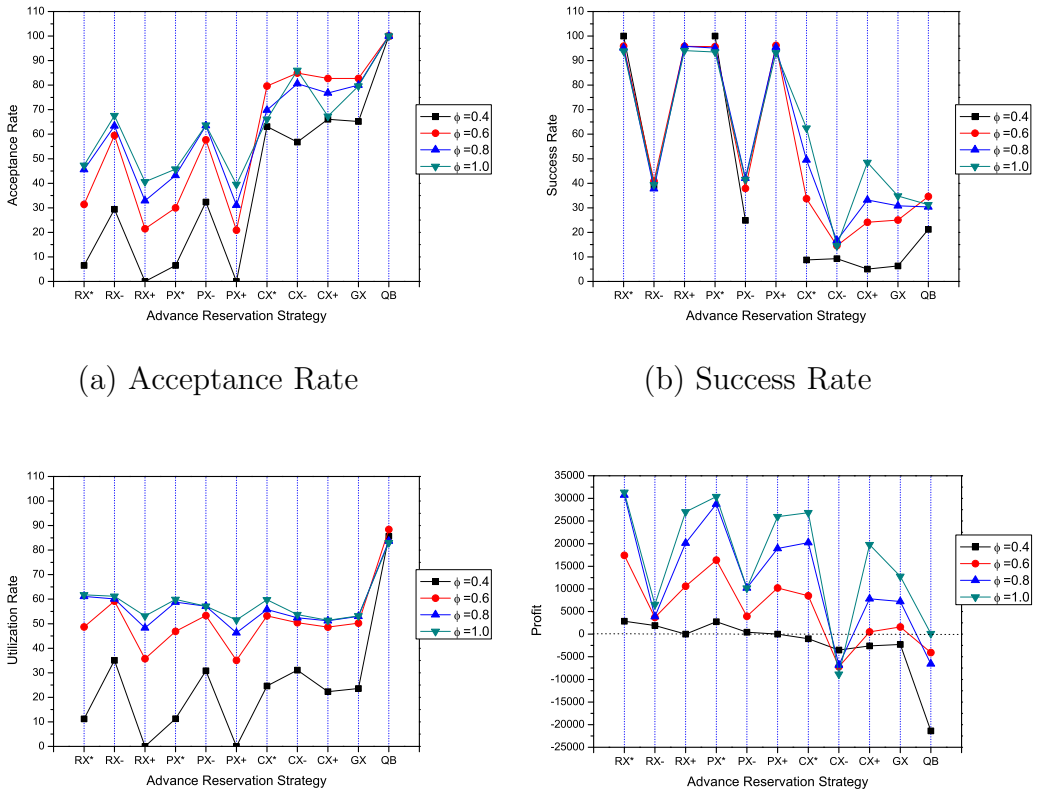
advance reservation strategies. This implies that when the workload is light to a certain extent, applying advance reservation may be unnecessary and may even be a disadvantage. The comparison results of all advance reservation strategies suggest that, when the estimated QoE was the same as the actual one, RX* performed better than other advance reservation strategies. When the QoE was underestimated, PX- performed slightly better than GX and evidently better than CX- in all settings of AD, and outperforms RX- when AD is equals to 50 or higher. When QoE was overestimated, the performance of RX+ and PX+ were comparable and better than CX+'s and GX's in all settings of AD, while the advantage weakened as AD decreased.

6.6.2 Impact of Budget and Deadline Constraint

The performance of different strategies under different QoS constraint conditions was investigated, which, from strict to relaxed, are respectively set by a constraint ratio (ϕ) equal to 0.4, 0.6, 0.8 and 1.0. The evaluation result is depicted in Figure 6.6. It should be noted in Figure 6.6(b) that, in an extreme case, the results of RX+ and PX+ may be invalid.

Firstly, the impact of the varying constraint ratio on different metrics was observed. For the application acceptance rate (AR), the RXs and PXs obtained higher AR as ϕ increased, while the CXs and GX hit the highest AR at $\phi = 0.6$. Every strategy obtained a significantly lower AR in the case of $\phi = 0.4$ (i.e., when the QoS constraint was quite strict) than in other cases, especially in this case RX+ and PX+ encountered an extreme situation in that almost no workflow could be accepted. The AR of QB remained 100%. For the success rate, the RXs was not affected by the varying ϕ, while the PXs, CXs and GX obtained higher SR as ϕ grew since they could achieve a higher extension ratio with more relaxed constraints. For the utilization rate and the profit, almost every strategy obtained an increased value as ϕ increased except for CX- which exhibited the contrary.

Secondly, the different profits earned by different strategies were compared. For all of the settings of ϕ, all of the advance reservation strategies could earn more profit than QB, especially when the constraints were tight (e.g., $\phi = 0.4$), the performance of QB turned out to be a big loss of profit. The focus then turned to comparing all of the advance reservation strategies. When the estimated QoE was the same as the actual one, RX* and PX* performed comparably and even better



(a) Acceptance Rate

(b) Success Rate

(c) Utilization Rate

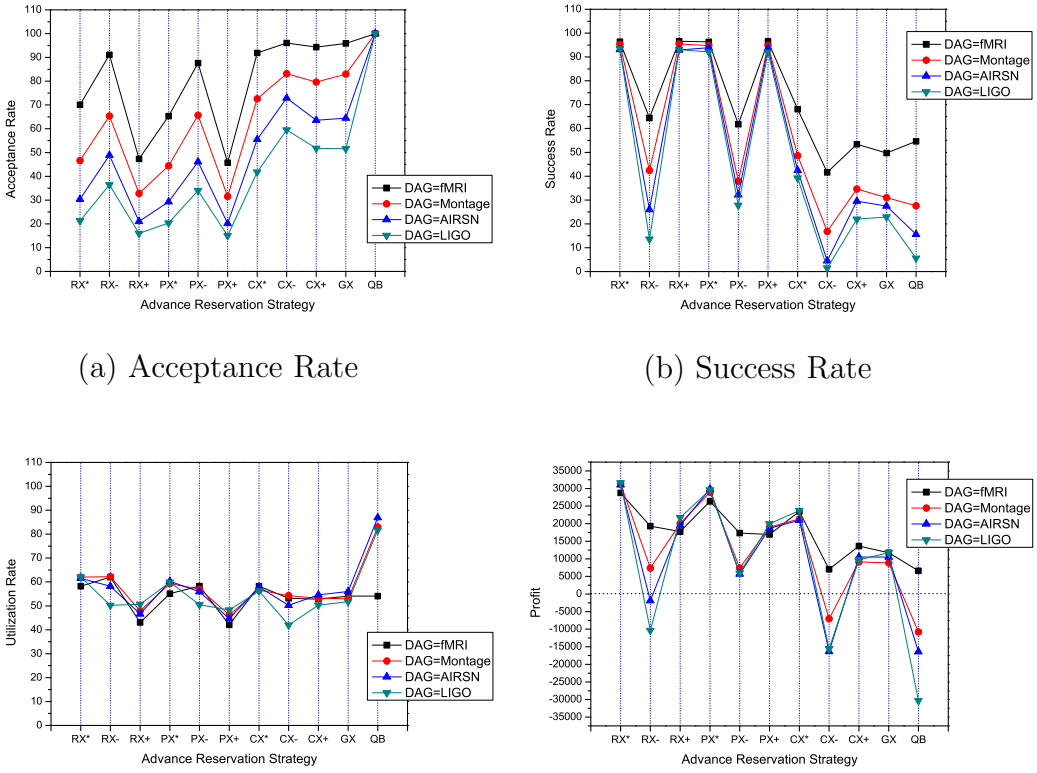
(d) Profit

Figure 6.6: Impact of Constraint Ratio (ϕ)

than other advance reservation strategies. When the QoE was overestimated, RX+ and PX+ still performed the best, although when $\phi = 0.4$ and using RX+ and PX+, the utilization rate and profit fell to zero, since there was almost no workflow being accepted. When the QoE was underestimated, the best strategy could be the RX- when the constraints were too tight, or the GX when the constraints were highly relaxed, or the PX- in the majority of the medium cases.

6.6.3 Impact of DAG Type

The performance of the examined strategies with different types of DAG workflows having various number of nodes was observed. The involved DAGs included fMRI with 17 nodes, Montage with 34 nodes, AIRSN with 55 nodes and LIGO with 77 nodes, which are depicted in Figure 2.12. The evaluation results are provided in Figure 6.7.



(a) Acceptance Rate

(b) Success Rate

(c) Utilization Rate

(d) Profit

Figure 6.7: Impact of DAG Type

Firstly, how the measured results of different metrics varied with different DAG types was observed. In terms of application acceptance rate (AR), for all of the advance reservation strategies, the bigger the DAG was, the less the AR achieved by each strategy. This was because bigger DAGs led to a heavier workload and made it more difficult for the planning of later arrived workflows to meet QoS constraints. For success rate (SR), RX*, RX+, PX* and PX+ were not sensitive to the variation of DAG type since the QoS can always be guaranteed by these four strategies. In contrast, for other strategies, in which successful task running was not guaranteed, the SR decreased as the DAG size grew. This was because the more nodes one DAG had, the more likely the occurrence of task running failure. For utilization rate, the impact caused by various DAG type settings on different advance reservation strategies was not apparent, while QB had significantly lower utilization for fMRI than other DAG types, because of the lighter workload resulting from the small application. For profit, the change

caused by various DAG type settings is similar to that for success rate.

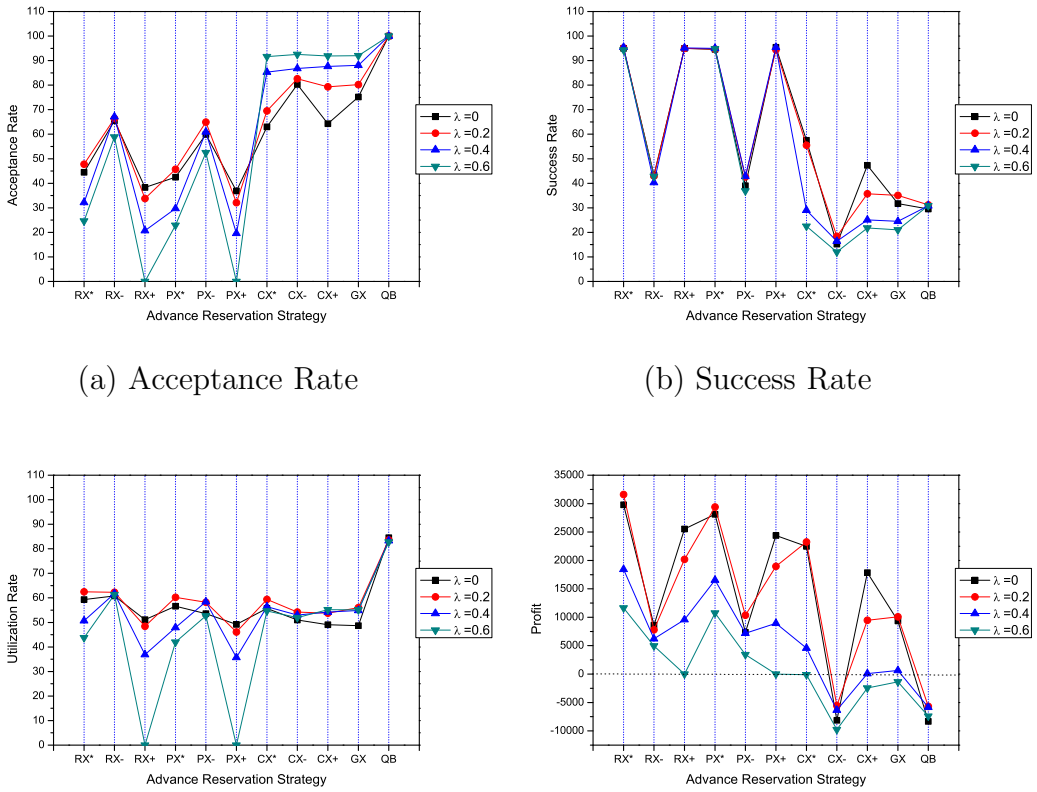
Secondly, the different profits earned by different strategies were compared. In most cases of DAG setting, the majority of the advance reservation strategies obtained a higher profit than QB except for CX-. With all DAG type settings, if the QoE could be perfectly estimated, RX* and PX* performed better than CX*, which in turn performed better than GX. However, when the QoE was underestimated, RX-, PX- and CX- usually performed worse than GX, except that when small DAG (for example, fMRI) was used, PX- may have performed better than GX. It is worth mentioning that when a big DAG was used (for example, AIRSN and LIGO), RX- may have led to a loss of profit.

6.6.4 Impact of Extension Price Ratio

How the examined strategies performed with different settings of extension price ratio at 0, 0.2, 0.4 and 0.6 was studied, and Figure 6.8 depicts the evaluation results.

Firstly, the impact of the varying extension price ratio (i.e., λ) on different metrics was observed. Since the advance reservation is not considered for QB, for all metrics, the performance of QB is insensitive to the variation of λ, and therefore QB is excluded from this discussion. For application acceptance rate (AR), RXs and PXs obtained lower AR with higher λ which made it more difficult to meet budget constraint in BDC-planning. In contrast, the AR values of CXs and GX grew as λ increased since the increased λ limited the extension ratio of the planning result and consequently lessened the resource competition. For success rate, RXs and PXs were not sensitive to the variation of λ, but other strategies obtained lower SR as λ grew since the extension ratio was restricted. For utilization rate, RX+ and PX+ were more sensitive to the variation of λ than other strategies and achieved lower resource utilization as λ increased. For profit, as λ evolved from 0 to 0.6, most of the advance reservation strategies earned less except for RX*, PX, PX- and GX which hit the highest profit at $\lambda = 0.2$.

Secondly, the different profits earned by different strategies were compared. In most cases of the DAG setting, the majority of the advance reservation strategies obtained higher profit than QB except for CX-. For all of the settings of λ, if the QoE could be perfectly estimated, RX* and PX* performed better than CX*, which in turn performed better than GX. However, when the QoE was underestimated, PX- usually performed the best. If the QoE was overestimated,



(a) Acceptance Rate

(b) Success Rate

(c) Utilization Rate

(d) Profit

Figure 6.8: Impact of Extension Price Ratio (λ)

RX+ and PX+ may have encountered an extreme situation in that no workflow was accepted, and therefore no profit was gained.

6.6.5 Summary

In summary, the evaluation examined the effectiveness of different advance reservation strategies with various simulation settings which could cover a wide spectrum of online workflow processing scenarios in the context of market-based grids. The results suggest the following:

- In most scenarios, the advance reservation strategies significantly outperform the queue-based strategy, which indicates the necessity of applying advance reservation to guarantee users' QoS requirements, when running workflows with hard constraints in the context of market-based grids;

- Advance reservation is not always necessary, and in some extreme case where the workload is highly light, a queue-based strategy may even perform better than the strategies using advance reservation;

- Accurately perceiving the range of task execution time changes in run-time (i.e., QoE) may play a crucial role in designing an efficient strategy to cope with the prediction uncertainty for running workflows in grids. As demonstrated in the evaluation, with a perfect estimate of QoE, the RX and PX strategies could significantly outperform the GX strategy, which does not consider QoE.

- There is no a single advance reservation strategy which can always obtain the highest profit in all circumstances, because extending the reserved duration for tasks may have various impacts on the system performance, which may result in an unforeseen influence of the revenue. Generally, the RX and PX strategies seem to perform best among the compared strategies, while the PX strategy is considered to be a better choice. Indeed, when the estimated QoE is not less than the actual one, the RX strategy can sometimes obtain a higher profit than the PX strategy. However, when the QoE is underestimated, the PX strategy can usually earn more than the RX strategy and avoid a loss of profit which may be encountered by the RX strategy.

6.7 Closing Remarks

This chapter presented four various advance reservation strategies which attempt to appropriately extend the reserved duration of workflow tasks by overestimating the estimation of task execution time and using a BDC-planning heuristic to obtain the advance reservation plan. Thus, not only can the reliability of the execution of a single workflow be increased, but also the profits that can be earned from a set of multiple workflow requests can be maximized. An evaluation of the designed advance reservation strategies was performed to investigate the impacts these strategies can have in the scenario of processing multiple workflows which have hard budget and deadline constraints and are submitted to an SLA-based scheduling system dynamically over time. The evaluation verifies the usefulness of advance reservation on both guaranteeing users' deadline requirements under

runtime changes and maximizing the overall income of the system, and suggests the proper strategies which can be used to automate the process of planning and making advance reservations for workflows in the SLA-based scheduling system.

It has been shown that the advance reservations generated by the strategies proposed in this chapter can enhance the guarantee of users' deadline requirements in an environment where the performance prediction is inaccurate. However, these reservations may probably lead to lower resource utilization and less system income due to runtime changes. This motivates the work in the next chapter to explore the investigation of applying local scheduling techniques (such as backfilling) to improve resource utilization and resource providers' benefit.

Chapter 7

SLA-based Local Scheduling

The previous chapter develops several "generous" strategies which overestimate task execution times in planning and making advance reservations for workflows to guarantee users' QoS requirements under the uncertainty of performance prediction. However, such a solution may lead to the problem of low resource usage when the actual task execution time is considerably shorter than the reserved duration. This chapter focuses on the problem of rescheduling the reserved workflow tasks on local resources according to runtime changes during workflow execution to improve the resource utilization and increase service providers' profits.

This chapter investigates the flexibility of an SLA-based workflow advance reservation, proposes a novel local scheduling policy which makes use of this flexibility, and applies a backfilling technique to reduce the fragments of resource utilization caused by advance reservation. The proposed policy, which complements the development of the proposed SLA-based scheduling system, is the main contribution of this chapter. A simulator is implemented to simulate the processing of a set of workflows, each of which goes through the planning, advance reservation and local scheduling in the SLA-based scheduling system. In simulation, the performance of the system is evaluated with applying different local scheduling policies. The experimental results not only demonstrate the increase of resource utilization and overall profits that the proposed local scheduling policy can gain, but also verify the effectiveness of the whole SLA-based scheduling system.

7.1 Background

In order to achieve a QoS guarantee for a user who requires to run a workflow in a market-based grid with hard deadline constraint, the SLA-based workflow scheduling system plans the workflow and makes advance reservations for individual workflow tasks to the planned allocated resources, where a certain amount of 'extra time' is considered in the reserved duration for each task to cope with uncertainty in estimating task execution time. These advance reservations are agreed between the user and the service providers in the form of an SLA, and every workflow task will be allowed to exclusively use the reserved resource within the reserved duration. In this way, the QoS requirements for grid users can be satisfied as promised. Chapters 5 and 6 addressed the problem of how the SLA for a submitted workflow can be appropriately generated via advance reservation.

Suppose that an SLA has been reached for a workflow application submitted to a grid system. At this stage, the user's and the involved service providers' obligations, such as completing the workflow before a hard deadline and the price to pay for successful execution, have been recorded and will be enforced in the forthcoming workflow running. Moreover, each workflow task has been allocated to the planned resource and is waiting to start the execution at the planned start time. Nevertheless, the planned start time for each individual workflow task is not a term of the SLA. Thus, the workflow task can actually start at any moment as long as the constraints on task dependencies and the specified deadline are not violated, since the user does not necessarily care about the detail of execution. All he/she needs to be concerned about is whether his/her QoS requirements will be satisfied. This implies the flexibility of an SLA-based advance reservation, which allows the local scheduler to flexibly arrange the actual start time of the allocated tasks. The procedure to do this is called 'local scheduling'.

One of the easiest ways of determining when the task should actually start running is to schedule the start time of task execution as planned. However, due to prediction uncertainty, the actual execution time of a task can often be shorter than the reserved duration, especially when the task execution time is overestimated. If workflow tasks are executed as planned, (namely, during a specific duration, a resource can only be exclusively used by the task associated with the reservation), this will normally lead to redundant reservation, i.e., a reservation which leaves the resources idle and consequently degrades the resource usage. This is obviously unacceptable for the resource owner, (i.e., service provider), to

whom keeping the resource utilization reasonably high and consequently maximizing profit is always important. Therefore, there must be some alternative technique for local scheduling to improve the resource utilization. To achieve this, it is believed that it is worth exploring the aforementioned flexibility in SLA-based advance reservation to fully exploit a backfilling technique's talents in local scheduling.

This chapter focuses on applying a backfilling technique based on flexible advance reservations of workflows, which are considered to be DAGs, to minimize utilization fragments caused by advance reservation. Although backfilling techniques have been widely used to improve resource utilization, the majority of them consider only independent tasks, each of which has a specific ready time and deadline, and therefore the runtime changes in one local resource have no impact on the scheduling in any other local resource. It is believed in this study that, in the case of workflow tasks which often have dependencies between them, a new local scheduling scheme which enables the interaction between local schedulers is necessary, in order to maximize the benefit of using a backfilling technique. This motivates the proposal of a novel local scheduling policy with backfilling. Simulation evaluations are carried out to investigate the performance of the proposed policy and its variants under various circumstances of grid computing scenarios. It should be noted that the task advance reservations of local resources are viewed as being dynamic, since new workflow requests are continuously submitted as the existing advance reservations are scheduled. As multiple workflow requests come in over time, the aim of this approach is to maximize resource utilization and service providers' profits, as well as meeting the user's QoS requirements.

The rest of this chapter is organized as follows. Section 7.2 briefly reviews the related work. Section 7.3 describes the problem to be resolved. Section 7.4 analyzes the flexibility of SLA-based advance reservation. Section 7.5 explains how the proposed model is used in the scenario of processing multiple workflow requests. Section 7.6 describes the steps of the simulated evaluation and discusses the evaluation results. Finally, Section 7.7 concludes the chapter.

7.2 Related Work

The backfilling technique was initially adopted in queue-based schedulers to increase the efficiency of resource usage. In order to increase resource utilization,

backfilling approaches allow jobs from the back of the execution queue to be executed before the jobs proceed in the queue, thus utilizing the idle resources while the latter are waiting for some precondition to be satisfied to begin execution. Typically, the jumping jobs are not allowed to disturb the execution of some reserved jobs. Therefore, this scheduling scheme requires that the execution time estimated for any involved job must be known in advance. Since backfilling was first implemented on a production system in the "EASY" scheduler [SCZL96], many variants have been developed including aggressive [SCZL96], conservative [MF01], selective [SKSS02], slack-based [TF99], relaxed [JMW02], multiple-queue [LS02] and LOS [SF05] backfilling.

Advance reservations with flexible parameters have been introduced into Grid computing environments to provide QoS for users. Kaushik et al [KFC06] investigate the impact of a flexible time interval, which they call time-window size, on the request waiting time, request blocking probability and resource utilization. Castillo et al [CRH07] use concepts from computational geometry to cope with the resource fragmentation caused by advance reservation, and design online scheduling algorithms to provide a QoS guarantee. Farooq et al [FMP06] evaluate a set of application-to-resource mapping algorithms with flexible advance reservation and propose an algorithm called Minimum Laxity Impact, which performs rescheduling when a new job arrives but minimizes the extent to which existing jobs are pushed toward their deadlines. In [RSR06, RSW04], authors explore the fuzziness in reservation requests to address the drawback of advance reservation. However, backfilling is not considered in these aforementioned works on advance reservation.

The impact of advance reservation on different backfilling algorithms is investigated in [LZ07, MDM08]. Netto and Buyya [NBB07] consider backfilling when rescheduling the existing advance reservations with flexible and adaptive time QoS parameters. These works apply backfilling when rescheduling queued jobs when new a job arrives. In the work of the present study, local scheduling with backfilling is triggered when a reserved task completes earlier than the expected end time.

The task execution time estimation inaccuracy is taken into account in [NB08], where backfilling is used in rescheduling co-allocation requests based on flexible advance reservation and processor remapping. However, in [NB08], the job is viewed as being a bunch of parallel tasks requiring multiple resources, namely

workflow applications are not considered. In the case of the present study, the focus is on local scheduling based on the flexibility arising from SLA-based workflow advance reservation, and task-resource remapping is not considered during local scheduling.

Unlike the above-mentioned studies, the focus of this study is applying backfilling in local scheduling for workflow tasks, each of which has the ready time and deadline constraints on its execution. Most existing backfilling studies focus on independent tasks and consider that the local scheduler performs backfilling in isolation. Therefore, the ready time and deadline associated with each task have to be fixed. However, in the case of executing workflows, these two parameters, which essentially bind the maximum range within which a task can be backfilled, are likely to vary due to runtime changes and consequently enlarge the range of the backfilling of the task. Without considering this variation, the performance of backfilling may be limited. Motivated by this observation, this chapter proposes a new local scheduling policy, which take into account the variation of the ready times and deadlines for tasks during runtime, in order to fully exploit the potential of backfilling so that the resource utilization can be maximized and the service providers' profits can be promoted.

7.3 Problem Description

To execute an accepted workflow request, the broker makes advance reservations for individual workflow tasks across multiple administrative domains managed by Local Resource Managers (LRM). Each local resource has its own *advance reservation queue* (*reservation queue* for short, hereafter) consisting of reserved tasks, which are initially generated according to the planning result of some SLA-based advance reservation strategy (such as those proposed in Chapter 6). Since the planning results are normally produced based on inaccurate task runtime estimation, the schedulers of local resources may update their reservation queues to reduce fragments in resource utilization.

In this chapter, the notations defined in Chapters 5 and 6 are reused. Similar to the existing load of resources defined in Eq.(5.5), at any time t^*, the reservation queue of resource γ_p can be represented by

$$\mathcal{Q}_p(t^*) = \{(t^{stt}_{\langle 0\rangle}, t^{end}_{\langle 0\rangle}), (t^{stt}_{\langle 1\rangle}, t^{end}_{\langle 1\rangle}), \cdots, (t^{stt}_{\langle j\rangle}, t^{end}_{\langle j\rangle}) \cdots\} \tag{7.1}$$

where t^{stt} is the planned start time of a task, t^{end} is the planned end time of a task, $\langle j \rangle$ means the index of the advance reservation in resource γ_p, $(t^{stt}_{\langle j \rangle}, t^{end}_{\langle j \rangle})$ denotes the jth advance reservation, especially, $(t^{stt}_{\langle 0 \rangle}, t^{end}_{\langle 0 \rangle})$ is the earliest reservation after t^* in resource γ_p. In addition, it is held that $t^{stt}_{\langle 0 \rangle} > t^*$ and $\forall j > 0$, $t^{end}_{\langle j-1 \rangle} \leq t^{stt}_{\langle j \rangle}$.

As time t^* goes on, multiple workflow requests arrive dynamically. The local scheduling problem considered in this chapter is to update the head reservation in the reservation queue for each resource and determine the actual start time of the head reservation in order to minimize resource utilization fragments and consequentially maximize resource utilization and the overall profits the service providers can earn from these workflows, while still meeting the constraints of task dependencies and workflow completion deadline. It should be noted that the head reservation may not necessarily be $(t^{stt}_{\langle 0 \rangle}, t^{end}_{\langle 0 \rangle})$ due to the use of the backfilling technique in local scheduling.

7.4 Flexibility of SLA-based Advance Reservation

This section attempts to illustrate and formulate the flexibility of an SLA-based advance reservation.

First of all, a motivation example to explore flexibility in SLA-based advance reservation is provided in Figure 7.1, where a simple workflow w consisting of four nodes has been planned and reserved for two resources. As illustrated in Figure 7.1, where Stt_w is the earliest possible time to start executing the workflow and Ddl_w is the specified deadline, the advance reservation for task 3 can be placed at any position between t_a and t_b without violating the task dependencies and deadline constraints. In fact, such flexibility exists for every task of the workflow.

Based on the notation defined in Chapter 6, the following parameters formulate the flexible advance reservation for task τ_i which is mapped to resource γ_p:

- $ert_{i,p}$: the estimated running time of task τ_i on resource γ_p;
- t^{stt}_i: the planned start time of task τ_i;
- rd_i: the reserved duration for task τ_i, $rd_i \geq ert_{i,p}$;
- t^{end}_i: the planned end time of task τ_i, defined as $t^{stt}_i + rd_i$;

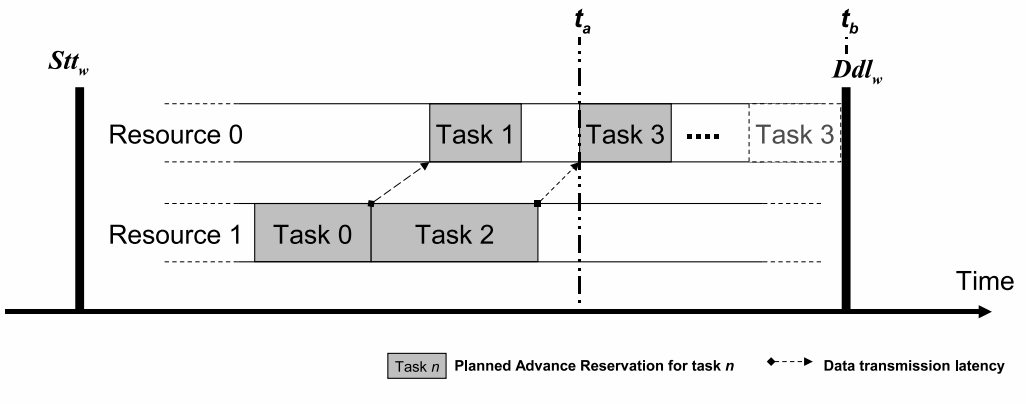


Figure 7.1: A motivation example of flexibility in SLA-based advance reservation for workflow

- $drt_{i,p}$: the data ready time of task τ_i on resource γ_p;
- $let_{i,p}$: the latest end time of task τ_i on resource γ_p;
- $\hat{t}_i^{stt}$: the actual start time (i.e., the last updated t_i^{stt}) of task τ_i;
- art_i: the actual runtime of task τ_i on resource γ_p;
- act_i: the actual completion time of task τ_i;
- $\hat{t}_i^{end}$: the last updated planned end time of task τ_i, defined as $\hat{t}_i^{stt} + rd_i$;

The computations of $drt_{i,p}$ and $let_{i,p}$ are respectively defined as below.

$$drt_{i,p} = \begin{cases} Stt_w & : \quad i = entry \\ \max\{\max\limits_{j \in Pred_c(i)}\{t_j^{end} + tl_{(j,r(j))\rightarrow(i,p)}\}, & \\ \quad \max\limits_{k \in Pred_u(i)}\{act_k + tl_{(k,r(k))\rightarrow(i,p)}\}\} & : \quad i \neq entry \end{cases} \tag{7.2}$$

$$let_{i,p} = \begin{cases} Ddl_w & : \quad i = exit \\ \max\limits_{j \in Succ(i)}\{t_j^{stt} - tl_{(i,p)\rightarrow(j,r(j))}\} & : \quad i \neq exit \end{cases} \tag{7.3}$$

where $entry$ and $exit$ represent the entry node and exit node of workflow w, $Pred_c(i)$ denotes the set of all completed parent tasks of τ_i and $Pred_u(i)$ represents the set of all uncompleted parent tasks, tl quantifies the transmission latency between two allocated tasks, and $r(j)$ means the resource where task τ_j is mapped.

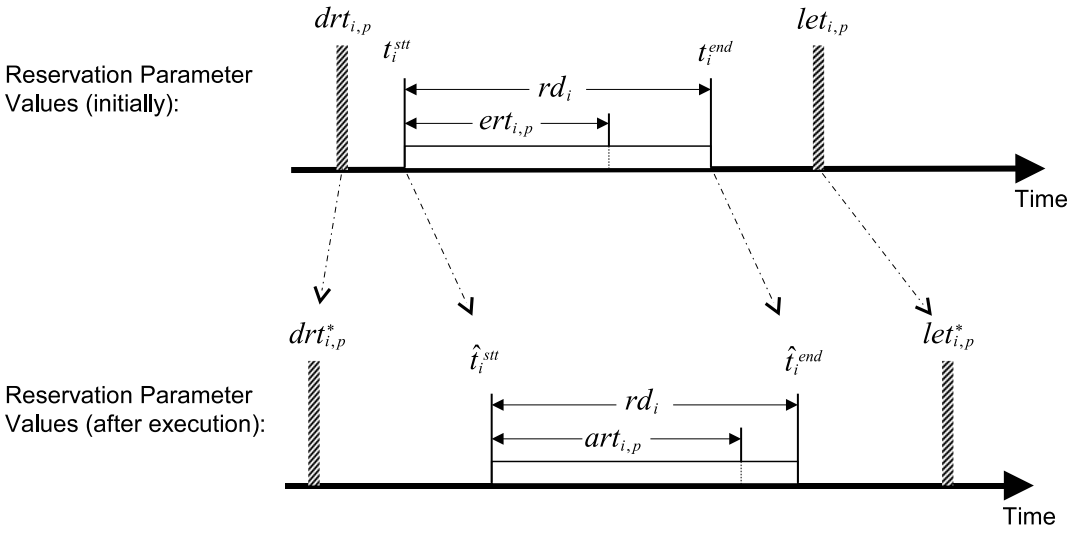


Figure 7.2: Parameters related to a task reservation

As illustrated in Figure 7.2, $drt_{i,p}$ and $let_{i,p}$ actually depicts the range in which the advance reservation for task τ_i on resource γ_p can be placed without violating task dependencies and deadline constraints. In terms of Eq.(7.2), $drt_{i,p}$ specifies the earliest possible time for task τ_i to obtain all the necessary data from its parent tasks. That is to say, the task dependency will not be violated as long as task τ_i starts execution later than $drt_{i,p}$ (i.e., $t_i^{stt} \geq drt_{i,p}$). According to Eq.(7.3), $let_{i,p}$ specifies the latest possible time that task τ_i has to complete to avoid delaying the planned start time of its child tasks, especially for the exit node, $let_{exit,r(exit)}$ is the hard deadline for the whole workflow. Assuming that tasks can always complete within the reserved duration, it can easily be imagined that the deadline constraint can always be met as long as the planned end time of τ_i is not later than $let_{i,p}$ (i.e., $t_i^{end} \leq let_{i,p}$). It should be noted that because of local scheduling during runtime, for each task τ_i, $drt_{i,p}$, $let_{i,p}$, t_i^{stt} and t_i^{end} may vary until τ_i starts running. Also, act_i cannot be known until τ_i completes its run. Figure 7.2 highlights the variation of these parameters from their initial state (as a result of planning) to the state after task execution, such as $drt_{i,p} \mapsto drt^*_{i,p}$, $let_{i,p} \mapsto let^*_{i,p}$, $t_i^{stt} \mapsto \hat{t}_i^{stt}$ and $t_i^{end} \mapsto \hat{t}_i^{end}$. This indicates the flexibility of an SLA-based advance reservation. Nevertheless, no matter how these parameters vary, it must be satisfied that $drt_{i,p} \leq t_i^{stt} < t_i^{end} \leq let_{i,p}$ for each τ_i during local scheduling.

7.5 Process Workflows with Local Scheduling

The focus is on such a scenario in which multiple workflow requests, each of which consists of a workflow and the associated budget and deadline constraints, are dynamically submitted over time, and meanwhile, other submitted workflows are being executed with possible local scheduling. The scheduling process is considered to be on-line, where workflow requests are submitted over time and the planner (broker) make planning decisions for each submitted workflow based on only the current existing load of resources (i.e., the existing task reservations in resources). If the planning result suggests the user's budget and deadline constraints can be met, a SLA is reached between the user and service providers is reached and advance reservations are made for individual workflow tasks to guarantee the user's QoS constraints, albeit with prediction uncertainty. These task reservations are flexible and may be rescheduled to start at some time different from the initially planned start time by local scheduling.

7.5.1 Overview from the Perspective of a Single Workflow

As mentioned in Chapter 4, the SLA-based scheduling of a submitted workflow consists of planning, making advance reservation and local scheduling, and there are three key roles: *service provider*, *user* and *broker* in the SLA-based scheduling model. The service provider administers multiple, heterogeneous resources which provide services of different capabilities and at a different costs. The users send job requests with the objective of making use of the provided services to run a workflow application within a certain deadline and budget. The broker is responsible for carrying out BDC-planning and advance reservation before workflow execution (as presented in the previous chapters) and coordinate local scheduling for the flexible reserved tasks during the runtime.

For each newly arriving workflow request, the whole processing can be divided into two phases: 'before-execution' and 'run-time', as illustrated in Figure 7.3.

In the before-execution phase, the submitted workflow is firstly planned to examine whether or not the user's budget and deadline constraints can be met. If yes, the workflow request is accepted, the advance reservations for individual workflow tasks are made with the planned resources, the task-resource mapping results are recorded by the broker and the SLA is reached. Otherwise, the workflow request is rejected.

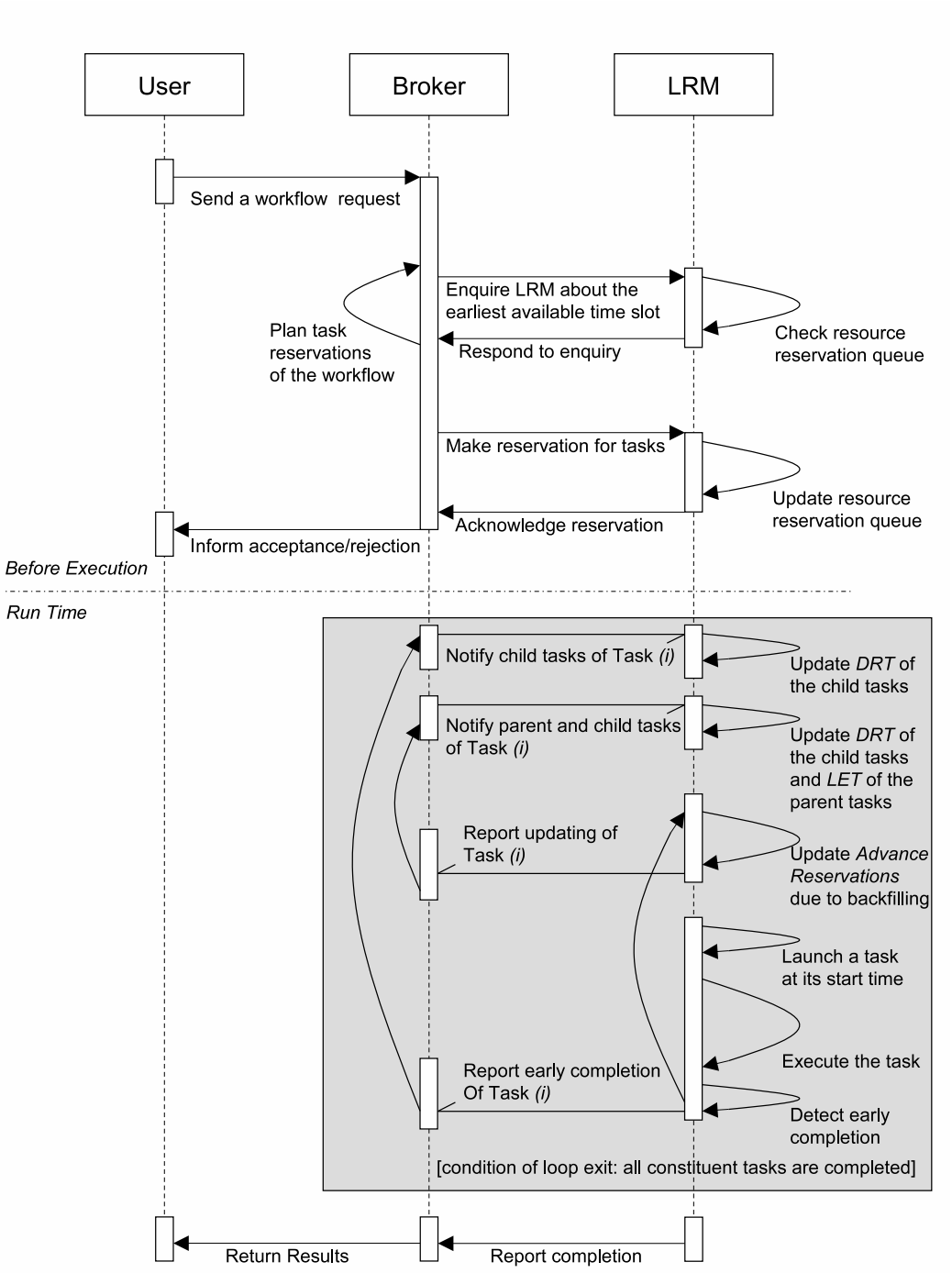


Figure 7.3: Sequence diagram for processing a workflow request in the SLA-based scheduling system

The run-time phase starts as the workflow execution begins. Each workflow task actually begins running at the last updated planned start time which may vary during runtime, as explained in Section 7.4. Pre-emption is not considered during task execution. When the task completes running earlier than the end of the reserved duration on the allocated resource, the local scheduling consisting of two operations, namely updating and backfilling, is launched on this resource. The details of these two operations are presented in Section 7.5.3.

7.5.2 Initial Planning and Advance Reservation

For each accepted workflow, the initial advance reservations are generated by following the four steps:

1. The broker plans the submitted workflow using one of the advance reservation strategies proposed in Chapter 6, where overestimation of task execution time and the current existing load of resources are taken into account;

2. According to the planning result, the broker makes advance reservations for individual tasks to the service providers, and the initial values of *drt* and *let* for each task reservation are computed;

3. Local schedulers update their advance reservation queues, which is perceived by the broker as existing load of resources, respectively;

4. The broker records the task-resource mappings and the SLA agreed upon between the user and the service providers.

7.5.3 Local Scheduling Operations

There are two operations in the proposed SLA-based local scheduling: (i) *Updating*: recompute *drt* and *let* of the reserved tasks according to runtime changes; and (ii) *Backfilling*: move a particular reserved task forward to the head of reservation queue of a local resource. It should be noted that the local schedulers perform these operations only for the reserved task waiting for execution, not during execution. The idea here is to find some flexible task reservation waiting for execution to fill the extra utilization gap caused by the early completion of an executed task, in order to reduce resource utilization fragments.

Updating

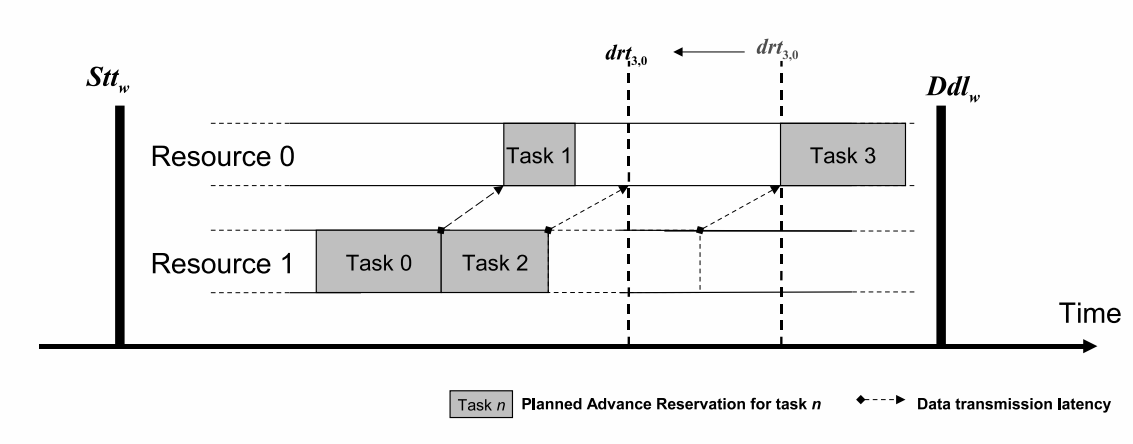


Figure 7.4: Example of Updating Operation

As previously mentioned, the dynamic variables $drt_{i,p}$ and $let_{i,p}$ bind the range within which the advance reservation for task τ_i should be placed. Therefore, it is necessary to update of these variables to ensure that the reserved task can be moved to the right place without violating task dependencies and deadline constraints. An updating operation is the procedure that changes the *drt* and/or *let* of the child and/or parent tasks, which have not begun to execute yet, of a task whose reserved start time or actual completion time varies due to local scheduling. As illustrated in Figure 7.3, an updating operation can be invoked by early completion and/or the moving of reserved tasks. And the interaction between local schedulers is enabled by the coordination of the broker. In the case of early completion of a task, for each child task of the task, the *drt* needs to be updated, while in the case of a task moving, both the *let* of each non-started parent task and the *drt* of each child task need to be updated. Suppose that the communication between the broker and local schedulers relies on message passing, and one message is needed to update the *drt* or *let* of a task at most, then the maximum number of messages needed to complete an updating operation is the sum of in-degree and out-degree of the involved task node plus one (the message passed from a local scheduler the broker to report an early completion of a task or a change of a local task reservation). Here, the in-degree and out-degree means the number of the parent and child tasks respectively. As can be seen from the DAG examples depicted in Figure 2.12, for typical DAGs, on average, the sum

of in-degree and out-degree of a task is not big. Therefore, the communication overhead introduced in the updating operation should not be high. Figure 7.4 provides an example of updating drt due to earlier completion, where the $drt_{3,0}$ of Task 3 becomes earlier due to the early completion of Task 2.

Backfilling

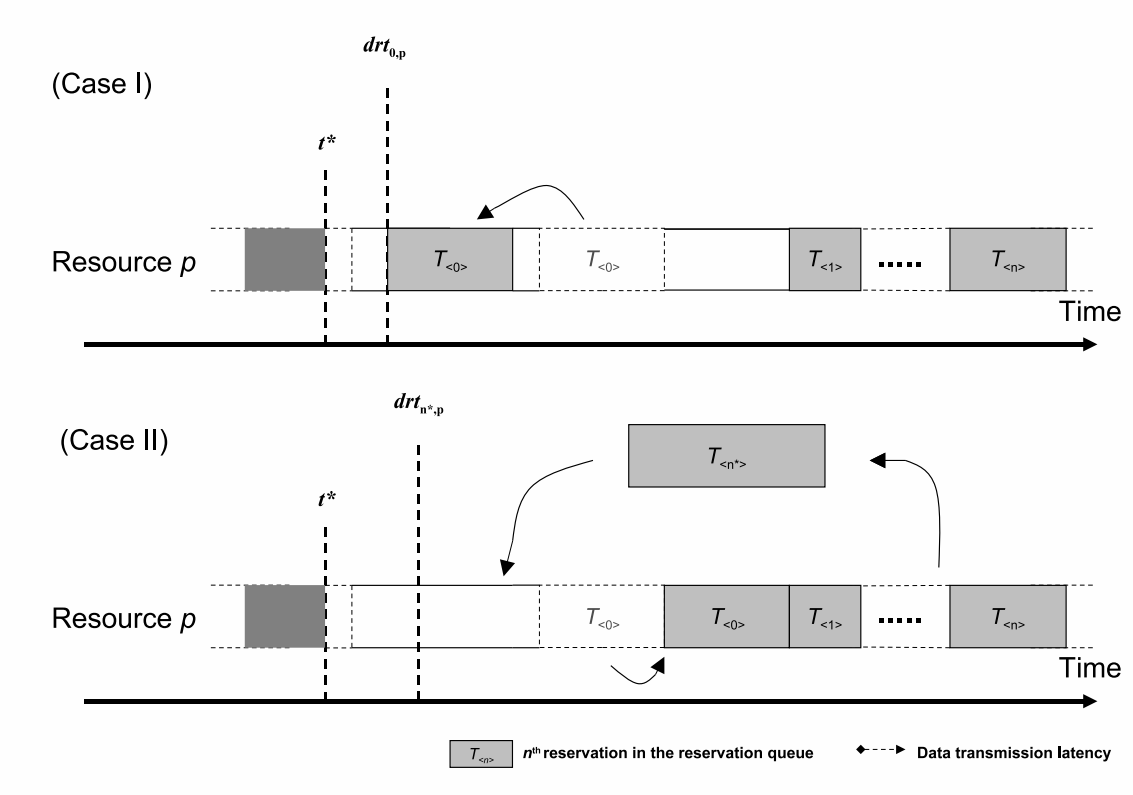


Figure 7.5: Example of Backfilling Operation

When a task completes before the reserving duration runs out, this leaves an extra gap in resource utilization. The idea of backfilling is to move a selected task from the reservation queue to fill the gap to utilize resource, but the moving of the task must be within the range bounded by its drt and let. Therefore, backfilling can only occur when a task is completed earlier than the end time of the reserved duration. Since most of the parameters relating to workflow task advance reservation such as c_i, t_i^{stt}, rd_i, t_i^{end}, $drt_{i,p}$ and $let_{i,p}$, are resource dependent, the migration of reserved tasks across different resources is not considered to avoid extra computational overhead, which may be significant due to the complexity of

rescheduling dependent tasks. Moreover, only a maximum of two reserved tasks are moved, one of which is the current head of the reservation queue, and the other is the selected task for backfilling (if the head is not selected). This is because (i) at any given moment, the only thing to consider is the execution of the current head reservation in the reservation queue. The remainder will be considered again upon the completion of the current head; and (ii) the more reserved tasks are moved, the more computational overheads will be introduced due to the updating operation, which may conflict with the real-time requirement of grid computing. Based on the above discussion, given that one task on resource γ_p just completes at time t^* which is before the reserved duration, and thus backfilling is invoked, there are two different possible cases of backfilling (as shown in Figure 7.5) which need to be taken into account:

- **Case I:** The task in the head position of the reservation queue (denoted by $\tau_{\langle 0\rangle}$) is considered for backfilling. In this special case, the following steps will be carried out:

  1. Compute $nt^{stt}_{\langle 0\rangle} = \max\{t^*, drt_{\langle 0\rangle,p}\}$;
  2. If $(nt^{stt}_{\langle 0\rangle} < t^{stt}_{\langle 0\rangle})$
     (a) Let $t^{stt}_{\langle 0\rangle} = nt^{stt}_{\langle 0\rangle}$ and $t^{end}_{\langle 0\rangle} = nt^{stt}_{\langle 0\rangle} + rd_{\langle 0\rangle}$, which means $\tau_{\langle 0\rangle}$ is moved ahead;
     (b) Update *let* of each non-started parent task of $\tau_{\langle 0\rangle}$ and *drt* of each child task.

- **Case II:** One task other than the head in the reservation queue (denoted by $\tau_{\langle i\rangle}$) is considered for backfilling. In this case, $\tau_{\langle i\rangle}$ is first removed from the reservation queue. Then $\tau_{\langle 0\rangle}$ is tentatively moved backward as much as possible without delaying any other tasks. The idea of doing this is to enlarge the utilization gap so that there may be more tasks *eligible* for backfilling. Finally $\tau_{\langle i\rangle}$ is checked to see if it is eligible for backfilling. If yes, $\tau_{\langle i\rangle}$ is inserted into the gap between t^* and $\tau_{\langle 0\rangle}$. In detail, the following steps will be carried out:

  1. Compute $nt^{end}_{\langle i\rangle} = \min\{let_{\langle 0\rangle,p}, t^{stt}_{\langle 1\rangle,p}\} - rd_{\langle 0\rangle}$;
  2. Compute $nt^{stt}_{\langle i\rangle} = \max\{t^*, drt_{\langle i\rangle,p}\}$;
  3. If $\tau_{\langle i\rangle}$ is *eligible* for backfilling

(a) Let $t^{stt}_{\langle i \rangle} = nt^{stt}_{\langle i \rangle}$ and $t^{end}_{\langle i \rangle} = nt^{stt}_{\langle i \rangle} + rd_{\langle i \rangle}$, thereby $\tau_{\langle i \rangle}$ becomes the new $\tau_{\langle 0 \rangle}$;

(b) Update *let* of each non-started parent task of $\tau_{\langle i \rangle}$ and *drt* of each child task.

(c) If the old $\tau_{\langle 0 \rangle}$ overlap with the new $\tau_{\langle 0 \rangle}$

i. move the old $\tau_{\langle 0 \rangle}$ backward till the overlap is eliminated;

ii. update *let* of each non-started parent task of the old $\tau_{\langle 0 \rangle}$ and *drt* of each child task.

It should be noted that in Case II, $\tau_{\langle i \rangle}$ is *eligible* for backfilling if, and only if, the following two conditions are both satisfied:

- $\tau_{\langle i \rangle}$ is not a successor of $\tau_{\langle 0 \rangle}$;
- $nt^{stt}_{\langle i \rangle} + rd_{\langle i \rangle} < nt^{end}_{\langle i \rangle}$.

7.5.4 Local Scheduling Policy

Based on existing task reservations on local resources, which are initially derived from workflow planning and advance reservation, the aim of a local scheduling policy is to find a reserved task to fill the fragment generated by the early completion of a running task. Whenever a task is completed before its reserved duration runs out, the local scheduler uses the algorithm described in Figure 7.6 to select a task from the reservation queue as the next task for execution.

To complete the description of the local scheduling algorithm, it is still necessary to explain how the priority of task reservations is computed to determine the particular task for backfilling. Here, various kinds of prioritization are considered, as follows:

- ***Shortest Task First* (STF)**: The task with the shortest reserved duration and which is eligible for backfilling is selected as the task with the highest priority, i.e., $h^* = \min_{v_{\langle k \rangle} \in \mathcal{Q}_p} \{rd_{\langle k \rangle} | \tau_{\langle k \rangle} \text{ is eligible for backfilling}\}$.
- ***Longest Task First* (LTF)**: The task with the longest reserved duration and which is eligible for backfilling is selected as the task with the highest priority, i.e., $h^* = \max_{v_{\langle k \rangle} \in \mathcal{Q}_p} \{rd_{\langle k \rangle} | \tau_{\langle k \rangle} \text{ is eligible for backfilling}\}$.

Input: The task τ_i on resource γ_p which is just completed at time t^* and before the end of reserving duration; and the current reservation queue $\mathcal{Q}_p(t^*)$ of γ_p.
Output: The next task to be executed in resource γ_p.

1. **For** each child task τ_j of task τ_i
2. Update $drt_{j,r(j)}$.
3. **Endfor**
4. **For** each task reservation $v_{\langle k \rangle} = (t^{stt}_{\langle k \rangle}, t^{end}_{\langle k \rangle}) \in \mathcal{Q}_p(t^*)$
5. Compute priority $h_{\langle k \rangle}$ of $v_{\langle k \rangle}$ using one of the prioritization methods described at the end of the section.
6. **Endfor**
7. Select v^* with the highest priority h^*.
8. **If** v^* is the head of $\mathcal{Q}_p(t^*)$ **then**
9. Schedule v^* with backfilling of Case I and possible updating as described in Section 7.5.3.
10. **Else**
11. Schedule v^* with backfilling of Case II and possible updating as described in Section 7.5.3.
12. **Endif**

Figure 7.6: Procedure for local scheduling, which is executed on the local schedulers when a task is completed before the reserved duration ends

- ***Earliest Task First* (ETF)**: The task with the earliest possible start time and which is eligible for backfilling is selected as the task with the highest priority, i.e., $h^* = \max_{v_{\langle k \rangle} \in \mathcal{Q}_p} \{nt^{stt}_{\langle k \rangle} = \max\{t^*, drt_{\langle k \rangle,p}\} | \tau_{\langle k \rangle} \text{ is eligible for backfilling}\}$.

7.6 Evaluation

In this section, the effectiveness of the proposed local scheduling policy is evaluated in a multiple-workflow scenario which is similar to the one described in Section 6.5. However, compared to the experiments of Section 6.5 where no rescheduling is considered in local resources, this experiment considers different local scheduling policies to reschedule the reserved workflow tasks during runtime. As the proposed local scheduling polices complement the design of the SLA-based workflow execution system, this experiment can also be regarded as an evaluation of the whole system. To highlight the impact of local scheduling which is only invoked when a task is completed before the end of the reserved duration, an ideal situation in which all reserved workflow tasks can always be completed within the reserved duration is assumed. In order to construct such a situation, the Rigid Extension (RX) advance reservation strategy presented in the previous chapter is used for the local scheduling model, and it is assumed that the actual QoE is less than, or equal to, the estimated QoE of RX.

The aim of the evaluation is to examine the performance of different variations of the proposed local scheduling policy in the light of the workflow acceptance rate, resource utilization and the total profit earned from all successfully executed workflows. Here, a workflow is considered to have been successfully executed if the budget and deadline constraints specified by its SLA are both met. Suppose that a set of workflows are sequentially submitted for processing during the period of $[a, b]$, the three metrics focused on are listed as follows:

- Acceptance Rate (AR): $AR = \frac{\text{Number of Accepted Workflows}}{\text{Number of Arrived Workflows}} \times 100\%$
- Utilization Rate (UR): $UR = \frac{\text{Sum of the Periods of Task Running}}{\text{Number of Resource} \times (b-a)} \times 100\%$
- Total Profit (ρ_W^{tot}): $\rho_W^{tot} = (\sum_{w \in W} \sum_{\tau_i \in w} \rho_i)$, where W denotes the set of multiple workflows; ρ_i is the profit service providers can earn from executing $\tau_i \in w$.

7.6.1 Simulator

The simulator described in Section 6.5.1 was extended to support the local scheduling model and an evaluation was carried out to see the effects of the proposed SLA-based local scheduling policy. The main changes made to the simulator presented in the previous chapter are (i) local scheduling is added as the action which reacts to the event of early completion and (ii) the event of execution failure is omitted. The new design of the simulator is briefly described below. With local scheduling, the simulator emulates the processing of a set of workflows W which sequentially arrive during the time of $[a, b]$:

1. *Step I: Constructing Environment.* The same as the description in Section 6.5.1.
2. *Step II: Generating Workflow Stream.* The same as the description in Section 6.5.1.
3. *Step III: Processing Workflow Stream.* From time a through b, for each time unit t^*, the simulator first checks if the time-stamped event (I) occurs. If it does, the corresponding actions are taken. Then the simulator goes through every resource and checks to see if one of the time-stamped events (II)-(III) occurs. If it does, the corresponding actions are taken.

- *Time Stamped Event (I): Workflow Arrival*, which is indicated by t^* equals to the arrival time of one of the time-stamped workflows. If a new workflow arrives, one of the proposed advance reservation strategies is invoked by the broker to plan the workflow. If the planning succeeds, the workflow is accepted and its tasks are reserved according to the planning result. The existing reservations on relevant resources are updated accordingly. The start and end of task reservation are time-stamped. The actual end time of each task is randomly generated according to the actual QoE and time-stamped as well.
- *Time Stamped Event (II): Execution Start*, which is indicated by t^* equals to the start time of a task reservation in a resource. If this occurs, it is implied that the task starts running.
- *Time Stamped Event (III): Early Completion*, which is indicated by t^* equals the actual end time of a task reservation in a resource. If this occurs, it is implied that the task has completed earlier than the planned end time and thus the task has been successfully completed. The task reservation is removed and the relevant metrics are measured and recorded. In addition, the local scheduling operations are invoked. One of the priorities defined in Section 7.5.4 is used in the proposed local scheduling algorithm to decide the head of the reservation queue of this resource, and the *drt* and *let* of the relevant tasks are updated.

4. *Step IV: Recording results*. The simulator collects the statistics on the above-mentioned three metrics.

In terms of the simulation, the same assumptions are made as listed in Section 6.5.1.

7.6.2 Experimental Setting

Different performance results were examined for the situation without applying local scheduling (i.e., the reserved tasks are launched at the same time as planned) and the situation where local scheduling is used. The former case was denoted as 'AsPlan'. In the latter case, three variants of the local scheduling policy, which respectively use one of the prioritization methods defined in Section 7.5.4, were considered. These variants were respectively denoted by 'STF_UXBF', 'LTF_UXBF'

and 'ETF_UXBF'. In order to demonstrate the significance of using the proposed local scheduling operations (as described in Section 7.5.3), two variants of the widely used Earliest Deadline First (EDF) algorithm with backfilling is also considered in the comparison. One variant is denoted by 'EDF_XBF', which does not consider the updating operation in the backfilling, and the other is denoted by 'EDF_BF', in which the updating and the moving of the head of the reservation queue in the backfilling of Case II (as described in Section 7.5.3) are both ignored. Therefore, there were six competitors in total in the evaluation.

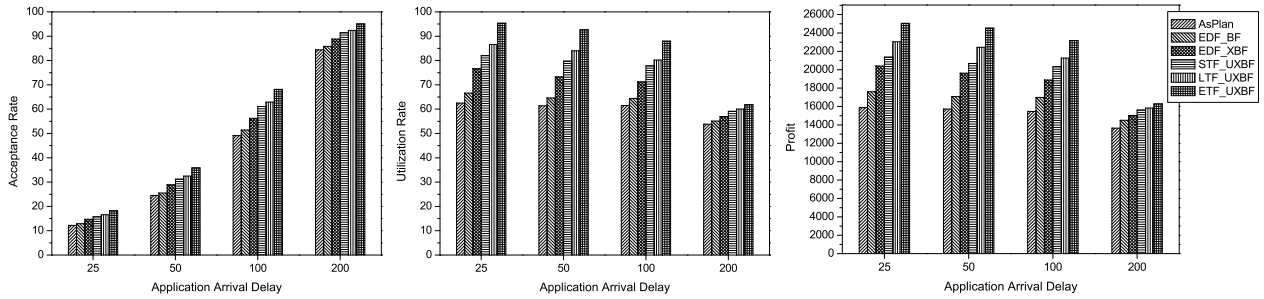
All of the competitors were evaluated under various simulation scenarios. For the configuration of the simulation, a parameter setting almost the same as that specified in Section 6.5.2 was adopted. The main difference is that in this experiment, it was assumed that the RX advance reservation strategy employed in local scheduling policies had a fixed estimated QoS, $\delta_{est} = 0.5$. Moreover, a varying actual QoE (δ_{act}) was assumed instead of a fixed one, whereas $\delta_{act} \leq \delta_{est}$. In addition to Application Arrival Delay (AD), Constraint Ratio (ϕ), DAG Type (DAG), and Extension Price Ratio (λ), the impact of actual QoE on the performance of different competitors was also investigated. In terms of the description of the simulation in Section 7.6.1, the actual QoE configures the simulation by affecting the generation of the actual task execution time in the actions to Time Stamped Event (I).

When investigating the impact of a varying parameter, all other parameters were allowed to stay in their default. By default, AD=100, $\phi = 0.8$, DAG='Montage with 34 nodes', $\lambda = 0.2$, and $\delta_{act} = 0.5$. In each simulation, the performance results for all competitors are recorded respectively. Due to the randomness of experimental settings, in one experiment, the simulation was carried out 100 times and the average result was obtained for every aforementioned metric.

7.6.3 Experimental Results

Firstly, the result of the impact of Application Arrival Delay (AD), which was varied at 25, 50, 100 and 200 to model different workloads from heavy to light, is investigated. As can be seen in Figure 7.7, for all the settings of Application Arrival Delay, both the acceptance rate and the utilization rate can be improved to a certain extent by applying the present study's local scheduling policies. The

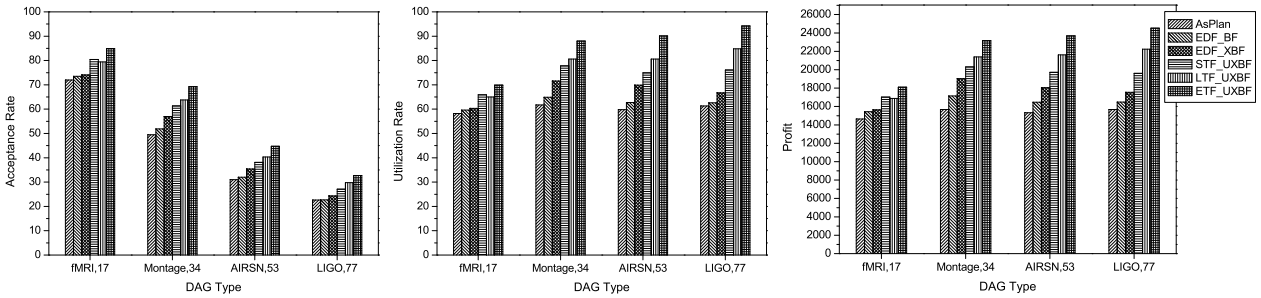
profit is also improved as a result. Among the policy variants, ETF_UXBF performs the best in all settings of AD. When AD equals 25, 50 and 100, ETF_UXBF manages to improve the profit by around 50% with respect to AsPlan. This improvement receives the main contribution from the improvement of the utilization rate, while the increase of the acceptance rate is less significant. As AD increases, which implies that there are fewer reserved tasks in the reservation queues of different resources, the improvement achieved by local scheduling decreases, and the policy variants exhibit a similar performance. However, when AD equals 200, around 20% of profit improvement can still be achieved. It is worth mentioning that, in all of the settings of AR, EDF_BF performs worse than EDF_XBF, which in turn performs worse than the proposed local scheduling policies.



(a) Acceptance Rate (b) Utilization Rate (c) Profit

Figure 7.7: Impact of Application Arrival Delay (AD)

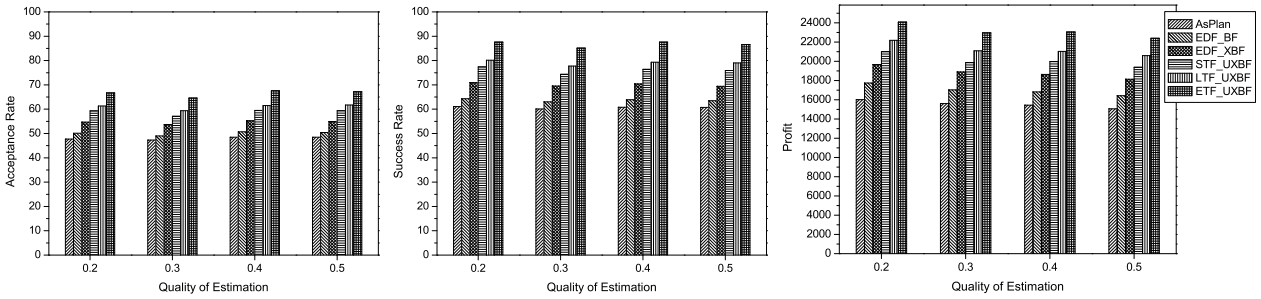
Secondly, with four different DAGs including fMRI with 17 nodes, Montage with 34 nodes, AIRSN with 53 nodes and LIGO with 77 nodes (as depicted in Figure 2.12) used in the evaluation, the observation of how various DAG types affect the evaluated metrics is presented in Figure 7.8. The results show that the proposed local scheduling policy can improve all measured metrics with respect to AsPlan for all settings of DAG. Again, ETF_UXBF outperforms other competitors. Among the various DAG types considered here, the proposed local scheduling policies provide the highest advantage to AsPlan in the case of LIGO with 77 nodes. This is because LIGO has the more nodes than other DAGs used here, which results in the most crowded reservations in the reservation queue and consequently, there is more likelihood of the occurrence of backfilling. The comparison results between EDF_BF, EDF_XBF and the proposed policies are the same as that mentioned in the analysis of the results of the varying Application Arrival Delay.



(a) Acceptance Rate (b) Utilization Rate (c) Profit

Figure 7.8: Impact of DAG type

Thirdly, attention is turned to the impact of actual QoS as shown in Figure 7.9. By the ideal assumption in this experiment, the actual QoE (δ_{act}) varies from 0.2 to 0.5 with a step of 0.1, which is never greater than the estimated QoE ($\delta_{est} = 0.5$) of the employed RX strategy. It is quite amazing to see that, given by using the RX strategy with a fixed δ_{est}, the performance of the RX strategy, which is suggested by the result of AsPlan, remains stable no matter how the actual QoE varies below δ_{est}. The local scheduling policy variants employing the RX strategy also exhibit a stable performance. In all cases of δ_{act}, ETF_UXBF performs the best and improves the resource utilization and profit by about 45% compared to AsPlan, while the performance comparison among other competitors retains the same results as above mentioned.

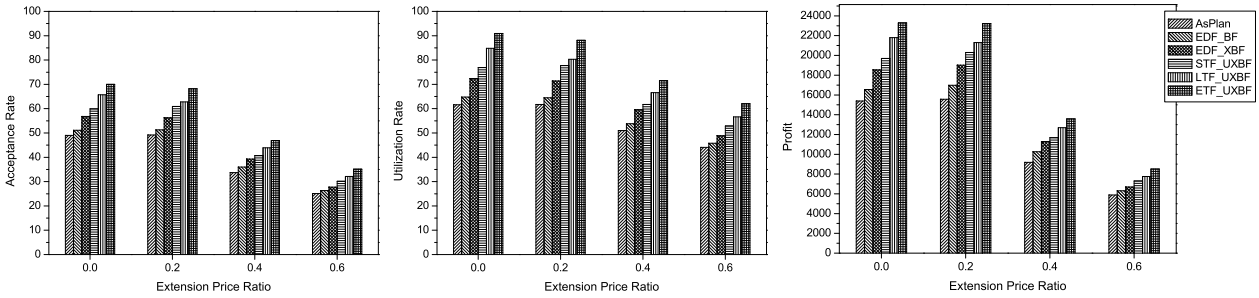


(a) Acceptance Rate (b) Utilization Rate (c) Profit

Figure 7.9: Impact of actual QoE

Fourthly, evolution through different settings of the Extension Price Ratio, which was varied from 0 to 0.6 with the step of 0.2, is investigated, and it is apparent that an appropriate setting of reservation price is important for the profit

earned by resource owners. If the price is set too low, the resource owner may earn less. On the other hand, if the price is set too high, it makes it more difficult to satisfy the budget constraint to successfully plan a new-coming workflow. This is clearly demonstrated in the results presented in Figure 7.10. As the reservation price ratio increases from 0 to 0.6, the job acceptance rate, the resource utilization and the profit decrease for all competitors, and the improvement achieved by the local scheduling policy also drops. However, no matter how much the reservation price ratio changes, the proposed local scheduling policy variants can always obtain an improvement compared to AsPlan. Again, ETF_UXBF performs the best.

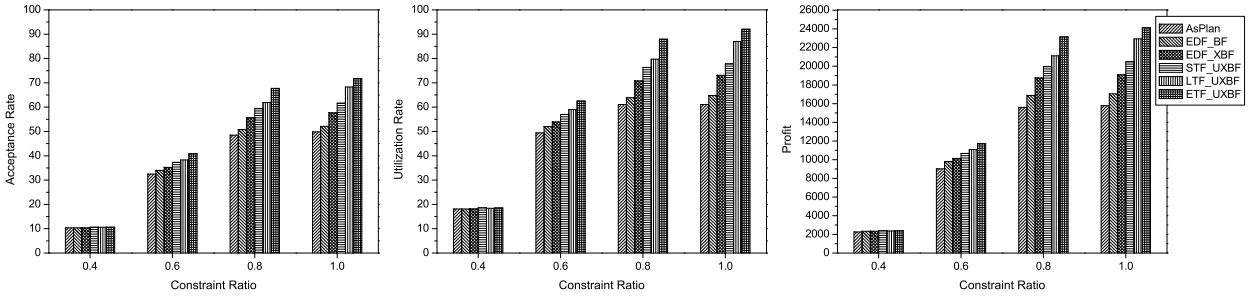


(a) Acceptance Rate (b) Utilization Rate (c) Profit

Figure 7.10: Impact of Extension Price Ratio

Finally, the varying results in terms of different settings of budget and deadline constraints are observed. Intuitively, the more strict these constraints are set, the more difficult it becomes for the planner to successfully plan the new-coming workflows. As can be seen in Figure 7.11, with the default setting of other parameters, if relaxed constraints are given, ETF_UXBF, which performs the best, can achieve a resource utilization as high as around 92% and improve the profit by about 50% with respect to AsPlan. This is because the relaxed constraints enable more flexibility to schedule the reserved tasks in terms of their SLAs. In most cases, the local scheduling policies can make an improvement no matter how the constraint setting changes. It can also be seen that, when the constraints are set too strictly (e.g., CR=0.4), the job acceptance rate, the resource utilization and the profit for all competitors are quite low, and the improvement made by local scheduling policies becomes slight or even negligible.

In summary, the experimental results indicates the following:



(a) Acceptance Rate (b) Utilization Rate (c) Profit

Figure 7.11: Impact of Constraint Ratio

- The proposed local scheduling policy variants can significantly increase resource utilization and service providers' profit in most of the scenarios simulated in the experiment, although the improvement may sometimes be limited due to strict constraint setting and sparse reservations in the reservation queue.

- The proposed local scheduling operations can make a significant improvement of the performance of the backfilling for the workflow tasks with ready time and deadline constraints. This is suggested by the comparison results of the performance of EDF_BF, EDF_XBF and the proposed local scheduling policies.

- The Earliest Task First (ETF) is the most appropriate prioritization method to be used in the proposed local scheduling policy variants to obtain best possible performance.

- In most settings of the various parameters, which represent different scenarios of running workflows with budget and deadline constraints in grids, the proposed SLA-based workflow execution system can both effectively guarantee users' QoS requirements and maximize service providers' profits, even though the task execution time may be unpredictable.

7.7 Closing Remarks

This chapter has considered the SLA-based local scheduling problem, which focuses on how to adjust the reservations of workflow tasks on local resources during

workflow execution to minimize the utilization fragments caused by advance reservation. The flexibility in SLA-based advance reservation was investigated, and two important notions relating to task advance reservation were defined to specify the boundary of the space where backfilling can be applied in local scheduling to maximize resource utilization. Some flexible advance reservation mechanisms have been proposed for grid computing, whereas none of these considers workflow applications with task dependencies.

The main contribution of this chapter was the proposed local scheduling approach which allows the interaction between local schedulers when they perform local scheduling with backfilling, i.e., to select one appropriate reserved task from the reservation queue to fill the extra utilization gap caused by early completion of the running task. Simulation experiments were carried out to emulate the processing of multiple workflow requests with or without using local scheduling, and in the case that local scheduling is used, various backfilling patterns were applied. The experimental results show that, by using the proposed local scheduling policy, resource utilization and service provider's profit could both be significantly improved. It is also indicated by the evaluation that, when running workflows with budget and deadline constraints in grids, the proposed SLA-based workflow execution system, as a combination of the BDC-planning, SLA-based Advance Reservation and SLA-based local scheduling techniques, can be a promising solution to both effectively guarantee users' QoS requirements and maximize service providers' profits, even though the task execution time may be unpredictable.

Chapter 8

Conclusions and Future Work

This chapter summarizes the thesis in Section 8.1, points out its limitations in Section 8.2 and finally lays out the directions for future work in Section 8.3.

8.1 Summary of the thesis

The ultimate aim of this thesis is to advance the state-of-the-art of grid workflow scheduling techniques assuming an idealized grid scenario. Although these observations do not relate to a real grid testbed, they can still provide insight that could be used to improve the efficiency of grid systems under various grid uncertainties. This aim has been achieved by developing new algorithms and novel scheduling mechanisms with relevant techniques in two different grid scheduling models as follows:

- Performance-driven model, which is a traditional scheduling model where the optimization of performance metrics is of the main concern, whereas users' QoS and the economic cost for application running are not considered.
- QoS-driven model, which has been widely used in recently emerged market-based grids where both the guarantee of users' QoS and the maximization service providers' profit are required.

The work presented in this thesis was thus constituted by the efforts of addressing issues caused by grids' dynamic features for the scheduling in a performance-driven model, and attempts to tackle various uncertainties for the scheduling in a QoS-driven model. In the performance-driven model, two uncertainties, i.e., the inaccuracy in task execution time prediction and the varying availability of

grid resources, were addressed, since these may affect the optimization of the scheduling performance. These uncertainties, as well as the uncertainty resulting from the queue-based scheduling, were also taken into account in the QoS-driven model, since they may all affect the guarantee of users' QoS and the maximization of the service provider's profit. The performance metric considered in the performance-driven model was makespan, while the QoS constraints considered in QoS-driven model were budget and deadline. Both scheduling models specifically concentrated on those workflows which can be represented by DAG.

The development of new algorithms started from the performance-driven model where two commonly used scheduling schemes—full-ahead and just-in-time—coexist. The former scheduling scheme is preferred if a performance prediction can be obtained, otherwise the latter. These two schemes were both covered in the work of this thesis. Then, the attention was turned to the scheduling in a QoS-based model, which is the latter part of the thesis. In this part, a number of approaches were proposed, which, when combined, addressed the uncertainties affecting the workflow scheduling in a QoS-based model. The major results obtained in this thesis include:

- A novel Monte Carlo-based scheduling approach was developed. By employing one of the existing static full-ahead DAG scheduling heuristics, the approach can improve the average makespan in various cases in which the task execution time changes stochastically (Chapter 2).

- A novel priority-based just-in-time workflow scheduling heuristic was proposed. The proposed heuristic manages to maximize the parallelism of ready tasks of DAG during execution and consequently improves the average makespan at low cost (Chapter 3).

- A novel workflow planning heuristic was presented in the context of a QoS-driven model. This heuristic can efficiently plan a DAG application with hard budget and deadline constraints specified by a user and can thus determine whether the pre-specified constraints can be met in terms of the existing load in resources (Chapter 5).

- A number of novel advance reservation strategies have been developed and evaluated in a QoS-driven scheduling model to automate the advance reservation of workflow applications in the context of market-based grids in order

to guarantee users' QoS requirements under task execution time changes (Chapter 6).

- A novel local scheduling policy has been designed to reduce the utilization fragments caused by advance reservation to maximize resource utilization and service provider's profit (Chapter 7).

These results were achieved by different portions of this work, which are respectively described in the next three subsections.

8.1.1 Full-ahead Scheduling

Based on a performance-driven model, there is an abundance of deterministic full-ahead DAG scheduling heuristics in literature, which focus on minimizing the makespan in heterogeneous systems based on a static prediction of task execution time. To tackle the unavoidable prediction uncertainty which may probably degrade the performance of deterministic heuristics, a new full-ahead scheduling approach based on Monte-Carlo method was developed in Chapter 2. This approach employs one of the existing heuristics to generate a specific full-ahead schedule in the fashion of the Monte-Carlo method. It is expected that the generated schedule can minimize the average makespan over runtime changes. The experimental results show that by applying the proposed approach, the scheduling performance of a static full-ahead heuristic, which is reflected by the average makespan, can be significantly improved. In some cases, this improved average makespan can even be better than the scheduling result obtained by the static heuristic with rescheduling, or by running the static heuristic in a post-mortem manner. Moreover, the schedule obtained by the proposed approach is full-ahead, and therefore its performance in runtime may be further improved by rescheduling. It was also indicated in the experiments that applying rescheduling to this schedule usually results in a better average makespan than applying rescheduling to the schedule which has been directly obtained by the static heuristic.

Apart from the two selective full-ahead scheduling heuristics employed in the implementation and evaluation of this approach, the proposed approach can be applied to any other existing static full-ahead heuristics. It is also worth mentioning that, although computational cost is normally not a fundamental obstacle for a full-ahead scheduling scheme, the parameters of the proposed Monte-Carlo

scheduling approach can be carefully configured to achieve competitive scheduling performance with an acceptable overhead.

8.1.2 Just-in-time Scheduling

Other than the full-ahead scheme which greatly relies on performance prediction, just-in-time scheduling is commonly adopted to cope with grid uncertainty in a situation in which there is no means to obtain information about the underlying computing platform. A novel just-in-time scheduling heuristic, the Priority-Based heuristic, was proposed in Chapter 3. This heuristic aims at maximizing the number of ready tasks at each execution step of a DAG application. When such a scheduling goal is achieved, the application is expected to perform efficiently on the remote resources even though their behavior changes unpredictably over time. The empirical results show that, in the comparison with other existing just-in-time scheduling heuristics, PB can either make a significant improvement by optimizing the application performance, or obtain a comparable or better performance with much lower complexity and wider applicability. This suggests that PB can be a promising solution to deal with grid uncertainties in a performance-driven model in the situation where performance prediction cannot be obtained.

8.1.3 SLA-based Scheduling

The thesis turned the scheduling objective from performance-driven to QoS-driven at Chapter 4, and focused on SLA-based scheduling in the context of market-based grids thereafter. The state-of-the-art of the existing market-based grid scheduling systems was reviewed. It is indicated in the review that the workflow scheduling problem with hard budget and deadline constraints has not been sufficiently explored, especially when performance prediction uncertainty is considered. With the aim of addressing this problem, a novel SLA-based scheduling model was designed and briefly presented. The SLA-based scheduling model presented in Chapter 4 is intended to make use of a Service Level Agreement in the scheduling of market-based grids, in which users require their applications to be completed before a specific deadline and within a certain budget, in order to tackle various uncertainties to guarantee users' requirements and maximize the benefits of the system. The development of the SLA-based scheduling model, as a complex challenge, was divided into three sub-problems: BDC-planning,

SLA-based advance reservation, and SLA-based local scheduling.

BDC-planning is needed to facilitate the establishment of an SLA. When a workflow request is submitted with the associated budget and deadline constraints, the planner must plan the workflow to examine whether or not these constraints can be met. If such a plan (i.e., BDC-plan) can be found, this is indicates that an SLA can be reached; otherwise, the request should be rejected. BDC-planning requires that the planned tasks must not overlap with the existing reserved tasks. Moreover, the planning must cost negligible time overhead due to the real-time requirement of grid computing. Chapter 5 contributed a novel BDC-planning heuristic, named Budget-constrained Heterogeneous Earliest Finish Time (BHEFT), which meet the aforementioned requirements. The aim of BHEFT is to maximize the likelihood of finding BDC-plans which subsequently results in the maximum chance to create SLAs for various DAGs with different constraints. Although there have been several low-cost multi-objective scheduling heuristics which can be cast for BDC-planning, BHEFT outperformed these heuristics in the study's simulated evaluation in which various scenarios with different settings of existing loads of resources and constraints were examined.

Based on the planning result of BHEFT, advance reservation can be made to avoid the uncertainty caused by queue-based scheduling which may ruin the QoS guarantee. However, the uncertainty in performance prediction can still raise problems if the reserving duration of tasks is not sufficiently considered. An investigation was carried out in Chapter 6 to examine how to plan and make an advance reservation with an appropriate amount of 'extra time' for workflow tasks to strike a balance between guaranteeing multiple users' QoS and maximize the service provider's profit. Based on the study of BDC-planning, three novel advance reservation strategies have been proposed. The proposed strategies are as follows:

- Rigid Extension (RX), which totally relies on an estimation of the inaccuracy of performance prediction;
- Greedy Extension (GX), which tries to expand the extra time as much as the QoS constraints permit;
- Conservative extension (CX), which attempts to expand the extra time to an extent not higher than a specified level;

- Progressive extension (PX), which attempts to expand the extra time to an extent not lower than a specified level;

All of these strategies can automate the process of planning and making advance reservation for workflow in market-based grids. As illustrated in the chapter, in order to run a workflow in the modelled market-based grid, all the user needs to specify is the budget and deadline constraints. The remainder of the workflow processing can be automated using one of the proposed strategies without user intervention. To evaluate the performance of the proposed strategies, a simulator, which can emulate the processing of multiple sequentially arriving workflows requests, was implemented. The processing includes planning, making advance reservation and task execution. A number of simulations were set to examine the effectiveness of different strategies. Generally, different strategies may exhibit advantages in different cases. However, if a close estimation of the inaccuracy of performance prediction can be obtained, the RX strategy will be the most profitable.

Although users' QoS can be guaranteed by the above strategies with sufficient 'extra time' for task reservation, such an approach usually results in low resource utilization, which may become even more serious due to prediction uncertainty, e.g., the early completion of tasks. This situation motivates the work in Chapter 7, which complements the development of SLA-based scheduling by providing a novel local scheduling policy to increase resource utilization. In Chapter 7, the flexibility of SLA-based advance reservation was explored, and two important notations, which depict the range in which a flexible task reservation can be placed, were defined. Using backfilling, the proposed local scheduling policy tries to adjust the flexible reservations on local resources during workflow execution to minimize the utilization fragments caused by advance reservation. The simulator implemented for the evaluation of advance reservation strategies was extended to support the emulation of local scheduling. The simulation results showed that, in major cases, the proposed local scheduling policy could achieve significant improvement of resource utilization and profit.

8.2 Limitations

Although the effectiveness and efficiency of this work on tackling various uncertainties for grid workflow scheduling have been shown via extensive simulation-based evaluation, it is believed that it can be further improved if some limitations in the following aspects can be addressed.

- Assumption. Although several unrealistic assumptions of static scheduling have been released in this dissertation, the proposed approaches still make some assumptions which may not always be valid. For instance, it is assumed that the data transmission latency between dependent tasks is constant, which may actually vary drastically due to Internet uncertainty. In addition, it is assumed in the QoS-driven model that, at most, one workflow request may come in a time unit.

- Resource mapping. Only the workflow tasks which require one resource to be used are considered. Nevertheless, some practical applications, one of which may be a single task of a workflow, may require multiple resources to be used, such as parallel MPI and PVM programs.

- Economic setting. Some simplistic setting in the economic model may limit the performance of the SLA-based scheduling system. For example, only fixed price is considered in price setting and the negotiation between users and the broker may be too rigid.

- Failure rescue. In the SLA-based scheduling model, there is lack of effective rescue operations when a task cannot be completed within its reserved duration. Terminating the whole workflow immediately may exclude some more efficient means to cope with this uncertainty issue.

- Scalability. Acting as a centralized planner and coordinator, the broker will be a bottleneck hindering the SLA-based scheduling model from addressing grid uncertainties for large systems.

- Idealized System. In this thesis, there is no study of real system case. The realistic system may be quite different with the assumed system. For instance, the simulated system has a small number of resources compared to a realistic Grid.

8.3 Future Work

Based on the limitations recognized in Section 8.2, the work in this thesis can be extended in the directions below. It should be noted that there have been extensive studies on workflow scheduling for performance-driven models, while the research into QoS-driven is still in its infancy. Thus, the main focus of the future work is on the latter model.

- The impact of uncertain network connections on performance of workflow may be worth investigating in both the full-ahead scheduling model and the SLA-based scheduling model in order to provide a more realistic scheduling solution for workflows in girds. It may also be necessary to support more practical workflow models which are not covered in this thesis. Some workflows consist of moldable tasks, the use of each of which requires multiple resources. Moreover, the task execution time is viewed as a function of the number of resources to be used. This may be interesting to be considered in both performance-driven and QoS-driven scheduling models.

- In order to strike a better balance on between the conflicting objectives of users and service providers, the development of renegotiation mechanism may be important. In a renegotiation model, users may specify flexible budget or deadline constraints, the price setting may be adaptive and a sophisticated compensation policy may be considered. All of these elements may result in greater flexibility during run-time to cope with grid uncertainties, and may consequently enhance the QoS guarantee and maximize service providers' profit.

- A more sophisticated local scheduling policy may be needed to analyze the risk that a whole workflow may miss its deadline when one of its tasks fails to complete before the reserved duration ends. Rather than canceling all of the remaining tasks, some evaluation and rescheduling technique may be applied to rescue the QoS so that the user's deadline constraint can still be satisfied. The aim of such an approach is to add more robustness to advance reservation under run-time changes. Moreover, the service providers may also benefit from a more successful service provision.

- A hierarchical scheduling scheme may be necessary to prevent the centralized broker from being overloaded. Multiple brokers can be coordinated

by a metaplanner, and each broker can keep connected with a set of resources of reasonable size and processes only a portion of a workflow, which is appropriately decomposed.

- It may be possible to extend BHEFT and apply it in a situation in which multiple DAGs require to be planned at the same time. This may occur when the randomly arriving workflow requests come in the same time unit. In such a situation, it is unlikely that all of these workflows can be successfully planned. Thus an appropriate priority may be needed for prioritizing the competing workflows to maximize job acceptance rate and service providers' profit.

- It will be of great interest to extend the theoretical work present in this thesis to the realistic grid systems.

Bibliography

[AAE+06] O. Ardaiz, P. Artigas, T. Eymann, F. Freitag, L. Navarro, and M. Reinicke. The catallaxy approach for decentralized economic-based allocation in grid resource and service markets. *Applied Intelligence*, 2(25):131–145, 2006.

[ABC+09] A. AuYoung, P. Buonadonna, B. N. Chun, C. Ng, D.C. Parkes, J. Shneidman, A. C. Snoeren, and A. Vahdat. Two auction-based resource allocation environments: Design and experience. In R. Buyya and K. Bubendorfer, editors, *Market Oriented Grid and Utility Computing*. Wiley Press, 2009.

[ABG02] D. Abramson, R. Buyya, and J. Giddy. A computational economy for grid computing and its implementation in the Nimrod-G resource broker. *Future Generation Computer Systems*, 18(8):1061–1074, 2002.

[ABJ+04] I. Altintas, C. Berkey, E. Jaeger, M. Jones, B. Ludaescher, and S. Mock. Kepler: Towards a grid-enabled system for scientific workflows. In *Proceedings of the Workflow in Grid Systems Workshop in GGF10*. GGF: Berlin, 2004.

[ADUM97] I. Ahmad, M. K. Dhodhi, and R. Ul-Mustafa. DPS: dynamic priority scheduling heuristic for heterogeneous computing systems. *IEE Proceedings of Computers and Digital Techniques*, 6(145):411–418, 1997.

[AG91] I. Ahmad and A. Ghafoor. Semi-distributed load balancing for massively parallel multicomputer systems. *IEEE Transactions on Software Engineering*, 17(10):987–1004, 1991.

[AHK98] R. Armstrong, D. Hensgen, and T. Kidd. The relative performance of

various mapping algorithms is independent of sizable variances in runtime predictions. In *Proceedings of the 7th Heterogeneous Computing Workshop*, pages 79–81, 1998.

[ASGH95] D. Abramson, R. Sosic, J. Giddy, and B. Hall. Nimrod: A tool for performing parametrized simulations using distributed workstations. In *Proceedings of the 4th International Symposium on High Performance Distributed Computing*, pages 112–121, 1995.

[ASL+08] L. Amar, J. Stosser, E. Levy, A. Shiloh, A. Barak, and D. Neumann. Harnessing migrations in a market-based grid OS. In *Proceedings of the 2008 9th IEEE/ACM International Conference on Grid Computing*, pages 85–94, 2008.

[ASMH00] S. Ali, H. J. Siegel, M. Maheswaran, and D. Hensqen. Task execution time modeling for heterogeneous computing systems. In *Proceedings of the 9th Heterogeneous Computing Workshop*, pages 185–199, 2000.

[BA09] R. Buyya and D. Abramson. The Nimrod/G grid resource broker for economic-based scheduling. In R. Buyya and K. Bubendorfer, editors, *Market-Oriented Grid and Utility Computing*. Wiley Press, 2009.

[BAGS02] R. Buyya, D. Abramson, J. Giddy, and H. Stockinger. Economic models for resource management and scheduling in grid computing. *Concurrency and Computation: Practice and Experience*, 14(13-15):1507–1542, 2002.

[BB07] J. Broberg and R. Buyya. A multi-commodity flow approach to maximising utility in linked market-based grids. In *MGC '07: Proceedings of the 5th international workshop on Middleware for grid computing*, pages 1–6. ACM, 2007.

[BBES04] I. Brandic, S. Benkner, G. Engelbrecht, and R. Schmidt. VGE-a service-oriented grid environment for on-demand supercomputing. In *Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing*, pages 11–18, 2004.

[BBES05] I. Brandic, S. Benkner, G. Engelbrecht, and R. Schmidt. QoS support for time-critical grid workflow applications. In *Proceedings of the First*

International Conference on e-Science and Grid Computing, pages 108–115, 2005.

[BBR02a] O. Beaumont, V. Boudet, and Y. Robert. The ISO-level scheduling heuristic for heterogeneous processors. In *Proceedings of 10th Euromicro Workshop on Parallel, Distributed and Network-Based Processing*, pages 335–350, 2002.

[BBR02b] O. Beaumont, V. Boudet, and Y. Robert. A realistic model and an efficient heuristic for scheduling with heterogeneous processors. In *Proceedings of the 16th International Parallel and Distributed Processing Symposium*, pages 88–101, 2002.

[BCD+08] S. Bharathi, A. Chervenak, E. Deelman, G. Mehta, M.-H. Su, and K. Vahi. Characterization of scientific workflows. In *Proceedings of the 3rd Workshop on Workflows in Support of Large-Scale Science (WORKS 2008)*, pages 1–10, 2008.

[BCT95] C. Boeres, G. Chochia, and P. Thanisch. On the scope of applicability of the ETF algorithm. In *Parallel Algorithms for Irregularly Structured Problems*, volume 980 of *Lecture Notes in Computer Science*, pages 159–164. Springer Berlin / Heidelberg, 1995.

[BD05] S. Baskiyar and C. Dickinson. Scheduling directed a-cyclic task graphs on a bounded set of heterogeneous processors using task duplication. *Journal of Parallel and Distributed Computing*, 8(65):911–921, August 2005.

[Be09] R. Buyya and K. Bubendorfer (eds.). *Market Oriented Grid and Utility Computing*. Wiley Press, 2009.

[BGA00] R. Buyya, J. Giddy, and D. Abramson. An evaluation of economy-based resource trading and scheduling on computational power grids for parameter sweep application. *Kluwer International Series in Engineering and Computer Science*, pages 221–230, 2000.

[BGL+04] G. B. Berriman, J. C. Good, A. C. Laity, A. Bergou, J. Jacob, D. S. Katz, E. Deelman, C. Kesselman, G. Singh, M.-H. Su, and R. Williams. Montage: A grid enabled image mosaic service for the

national virtual observatory. *Astronomical Society of the Pacific Conference Series*, 314:593–596, 2004.

[BJD+05] J. Blythe, S. Jain, E. Deelman, Y. Gil, K. Vahi, A. Mandal, and K. Kennedy. Task scheduling strategies for workflow-based applications in grids. In *Proceedings of the 5th IEEE International Symposium on Cluster Computing and the Grid*, volume 2, pages 759–767, 2005.

[BMAV05] R. Buyya, M. Murshed, D. Abramson, and S. Venugopal. Scheduling parameter sweep applications on global grids: a deadline and budget constrained cost-time optimization algorithm. *Software—Practice and Experience*, 35:491–512, Jan 2005.

[BSB+01] T. D. Braun, H. J. Siegel, N. Beck, L. L. Boloni, M. Maheswaran, A. I. Reuther, J. P. Probertson, M. D. Theys, and B. Yao. A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems. *Journal of Parallel and Distributed Computing*, 61:810–837, 2001.

[BV04] R. Buyya and S. Venugopa. The Gridbus toolkit for service oriented grid and utility computing: An overview and status report. In *Proceedings of 1st IEEE International Workshop on Grid Economics and Business Model (GECON'04)*, pages 19–36, 2004.

[Cam03] Cambridge. *Cambridge Advanced Learner's Dictionary*. Cambridge University Press, 2003.

[CDK+05] K. Cooper, A. Dasqupta, K. Kennedy, C. Koelbel, A. Mandal, G. Marin, M. Marin, M. Mazina, J. Mellor-Crummey, F. Berman, H. Casanova, A. Chien, H. Dail, X. Liu, A. Oluqbile, O. Sievert, H. Xia, L. Johnsson, B. Liu, M. Patel, D. Reed, W. Deng, C. Mendes, Z. Shi, A. Yarkhan, and J. Dongarra. New grid scheduling and rescheduling methods in the GrADS project. *International Journal of Parallel Programming*, 21(33):209–229, 2005.

[CFW98] H. Chen, N. S. Flann, and D. W. Watson. Parallel genetic simulated annealing: A massively parallel SIMD approach. *IEEE Transactions on Parallel and Distributed Systems*, 2(9):126–136, 1998.

[Che05] H. Chen. On the design of task scheduling in the heterogeneous computing environments. In *the 2005 IEEE Pacific Rim Conference on Communications, Computers and signal Processing*, pages 396–399, 2005.

[CJ01] B. Cirou and E. Jeannot. Triplet: A clustering scheduling algorithm for heterogeneous systems. In *Proceedings of the International Conference on Parallel Processing*, pages 231–236, 2001.

[CJS+02] J. Cao, S. A. Jarvis, S. Saini, D. J. Kerbyson, and G. R. Nudd. ARMS: An agent-based resource management system for grid computing. *Scientific Programming*, 10(2):135–148, 2002.

[CJSN03] J. Cao, S. A. Jarvis, S. Saini, and G. R. Nudd. GridFlow: workflow management for grid computing. In *Proceedings of the 3rd International Symposium on Cluster Computing and the Grid*, pages 198–205, 2003.

[CJSZ08] L.-C. Canon, E. Jeannot, R. Sakellariou, and W. Zheng. Comparative evaluation of the robustness of DAG scheduling heuristics. In *Grid Computing*, pages 73–84. Springer US, 2008.

[CLZB00] H. Casanova, A. Legrand, D. Zagorodnov, and F. Berman. Heuristics for scheduling parameter sweep applications in grid environments. In *Proceedings of the 9th Heterogeneous Computing Workshop*, pages 349–363, 2000.

[CMR06] G. Cordasco, G. Malewicz, and A. L. Rosenberg. On scheduling expansive and reductive DAGs for internet-based computing. In *26th IEEE International Conference on Distributed Computing Systems (ICDCS 2006)*, pages 29–37, 2006.

[CMR07a] G. Cordasco, G. Malewicz, and A. L. Rosenberg. Advances in IC-scheduling theory: Scheduling expansive and reductive DAGs and scheduling DAGs via duality. *IEEE Transactions on Parallel and Distributed Systems*, 18(11):1607–1617, 2007.

[CMR07b] G. Cordasco, G. Malewicz, and A. L. Rosenberg. Applying IC-scheduling theory to familiar classes of computations. In *Proceedings*

of the 21st IEEE International Parallel and Distributed Processing Symposium, pages 1–8, 2007.

[Cof76] E. G. Coffman. *Computer and job-shop scheduling theory.* John Wiley and Sons, 1976.

[Cor77] G. Corliss. Which root does the bisection algorithm find. *SIAM Review*, 2(19):325–327, 1977.

[CP96] M. Coli and P. Palazzari. Real time pipelined system design through simulated annealing. *Journal of Systems Architecture*, 6-7(42):465–475, December 1996.

[CR09] G. Cordasco and A. L. Rosenberg. On scheduling dags to maximize area. In *Proceedings of IEEE International Symposium on Parallel and Distributed Processing*, pages 1–12, 2009.

[CRH07] C. Castillo, G. Rouskas, and K. Harfoush. On the design of online scheduling algorithms for advance reservations and QoS in grids. In *Proceedings of the 21st IEEE International Parallel and Distributed Processing Symposium*, pages 1–10, 2007.

[CRH08] C. Castillo, G. N. Rouskas, and K. Harfoush. Efficient resource management using advance reservations for heterogeneous grids. In *Proceedings of IEEE International Symposium on Parallel and Distributed Processing*, pages 1–12, 2008.

[CST+02] J. Cao, D. P. Spooner, J. D. Turner, S. A. Jarvis, D. J. Kerbyson, S. Saini, and G. R. Nudd. Agent-based resource management for grid computing. In *Proceedings of the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid*, pages 350–350, 2002.

[CYL04] M. Chen, G. Yang, and X. Liu. Gridmarket: A practical, efficient market balancing resource for grid and p2p computing. In Minglu Li, Xian-He Sun, Qianni Deng, and Jun Ni, editors, *Grid and Cooperative Computing*, volume 3033 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 2004.

[CZ04] J. Cao and F. Zimmermann. Queue scheduling and advance reservation with COSY. In *Proceedings of the 18th IEEE International Parallel and Distributed Processing Symposium*, pages 63–71, 2004.

[DA] DagMan: `http://www.cs.wisc.edu/condor/dagman/`.

[DBG+04] E. Deelman, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, S. Patil, M. H. Su, K. Vahi, and M. Livny. Pegasus: Mapping scientific workflows onto the grid. In *Grid Computing*, volume 3165 of *Lecture Notes in Computer Science*, pages 131–140. Springer Berlin / Heidelberg, 2004.

[DGST09] E. Deelman, D. Gannon, M. Shields, and I. Taylor. Workflows and e-Science: An overview of workflow system features and capabilities. *Future Generation Computer Systems*, 5(25):528–540, 2009.

[DK08] M. I. Daoud and N. Kharma. A high performance algorithm for static task scheduling in heterogeneous distributed computing systems. *Journal of Parallel and Distributed Computing*, 4(68):399–409, April 2008.

[DKM+02] E. Deelman, C. Kesselman, G. Mehta, L. Meshkat, L. Pearlman, K. Blackburn, P. Ehrens, A. Lazzarini, R. Williams, and S. Koranda. GriPhyN and LIGO, building a virtual data grid for gravitational wave scientists. In *High Performance Distributed Computing (HPDC 02)*, pages 225–234, 2002.

[DÖ01] A. Doğan and F. Özgüner. Stochastic scheduling of a meta-task in heterogeneous distributed computing. In *Proceedings of the International Conference on Parallel Processing*, pages 369–374, 2001.

[DO02] A. Dŏgan and R. Özqüner. LDBS: a duplication based scheduling algorithm for heterogeneous computing systems. In *Proceedings of the International Conference on Parallel Processing*, pages 352–359, 2002.

[DÖ04] A. Doğan and F. Özgüner. Genetic algorithm based scheduling of meta-tasks with stochastic execution times in heterogeneous computing systems. *Cluster Computing*, 7:177–190, 2004.

[DO05] A. Dŏgan and R. Özqüner. Biobjective scheduling algorithms for execution time-reliability trade-off in heterogeneous computing systems. *The Computer Journal*, 3(48):300–314, 2005.

[EG] Eurogrid project: `http://www.eurogrid.org/`.

[EM] EMAN: `http://blake.bcm.tmc.edu/eman/`.

[EP] e-Protein: `http://www.e-protein.org/`.

[ERA+03] T. Eymann, M. Reinicke, O. Ardaiz, P. Artigas, F. Freitag, and L. Navarro. Decentralized resource allocation in application layer networks. In *Proceedings of the 3rd International Symposium on Cluster Computing and the Grid*, pages 645–650, 2003.

[ERL90] H. El-Rewini and T. G. Lewis. Scheduling parallel program tasks onto arbitrary target machines. In *Journal of Parallel and Distributed Computing*, volume 9, pages 138–153, 1990.

[ERS+05] T. Eymann, M. Reinicke, W. Streitberger, O. Rana, L. Joita, D. Neumann, B. Schnizler, D. Veit, O. Ardaiz, P. Chacin, I. Chao, F. Feitag, L. Navarro, M. Catalano, M. Gallegati, G. Guilioni, R. C. Schiaffino, and F. Zini. Catallaxy-based grid markets. *Multiagent Grid Systems*, 4(1):297–307, 2005.

[ES01] D. W. Erwin and D. F. Snelling. UNICORE: a grid computing environment. *Euro-Par 2001 Parallel Processing*, 2150:825–834, 2001.

[ET05] E. Elmroth and J. Tordsson. A grid resource broker supporting advance reservations and benchmark-based resource selection. *Lecture Notes in Computer Science*, 3732:1077–1085, 2005.

[FJP+05] T. Fahringer, A. Jugravu, S. Pllana, R. Prodan, C. S. Junior., and H.-L. Truong. ASKALON: A tool set for cluster and grid computing. *Concurrency and Computation: Practice and Experience*, 17:143–169, 2005.

[FK99] I. Foster and C. Kesselman. *The Grid: Blueprint for a Future Computing Infrastructure*. Morgan-Kaufmann, 1999.

[FK03] I. Foster and C. Kesselman. *The Grid 2: Blueprint for a Future Computing Infrastructure*. Morgan-Kaufmann, 2003.

[FKL+99] I. Foster, C. Kesselman, C. Lee, R. Lindell, K. Nahrstedt, and A. Roy. A distributed resource management architecture that supports advance reservations and co-allocation. In *Proceedings of the International Workshop on Quality of Service*, pages 27–36, 1999.

[FMP05] U. Farooq, S. Majumdar, and E. W. Parsons. Impact of laxity on scheduling with advance reservation in grids. In *Proceedings of the 13th IEEE International Symposium on Modelling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS'05)*, pages 319–322, 2005.

[FMP06] U. Farooq, S. Majumdar, and E. W. Parsons. A framework to achieve guaranteed QoS for applications and high system performance in multi-institutional grid computing. In *Proceedings of the International Conference on Parallel Processing*, pages 373–380, 2006.

[Fos02] I. Foster. What is the gird? a three point checklist. *http://www-fp.mcs.anl.gov/foster/Articles/WhatIsTheGrid.pdf*, 2002.

[FSZX04] H. Feng, G. Song, Y. Zheng, and J. Xia. A deadline and budget constrained cost-time optimization algorithm for scheduling dependent tasks in grid computing. In *Grid and Cooperative Computing*, volume 3033 of *Lecture Notes in Computer Science*, pages 113–120. Springer Berlin / Heidelberg, 2004.

[GBS09] S. K. Garg, R. Buyya, and H. J. Siegel. Scheduling parallel applications on utility grids: Time and cost trade-off management. In *Thirty-Second Australasian Computer Science Conference (ACSC 2009)*, volume 91, pages 139–147, 2009.

[GEJ+08] P. Gardfjäll, E. Elmroth, L. Johnsson, O. Mulmo, and T. Sandholm. Scalable grid-wide capacity allocation with the swegrid accounting system(SGAS). *Concurrency and Computation: Practice and Experience*, 18(20):2089–2122, 2008.

[GJ79] M. R. Gary and D. S. Johnson. *Computer and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Co., 1979.

[GKB08] S. K. Garg, P. Konugurthi, and R. Buyya. A linear programming driven genetic algorithm for meta-scheduling on utility grids. In *Proceedings of the 16th International Conference on Advanced Computing and Communication (ADCOM 2008)*, pages 19–26, 2008.

[GP] GriPhyN: `http://www.griphyn.org/`.

[GR] GrADS: `http://www.hipersoft.rice.edu/grads/`.

[GY92] A. Gerasoulis and T. Yang. A comparison of clustering heuristics for scheduling directed acyclic graphs on multiprocessors. *Journal of Parallel and Distributed Computing*, 16:276–291, 1992.

[Han92] W. Hanson. The dynamics of cost-plus pricing. *Managerial and Decision Economics*, 13:149–161, 1992.

[HDW+05] J. V. Horn, J. Dobson, J. Woodward, M. Wilde, Y. Zhao, J. Voeckler, and I. Foster. Grid-based computing and the future of neuroscience computation. *Methods in Mind*, pages 141–170, 2005.

[HH75] J. M. Hammersley and D. C. Handscomb. *Monte Carlo Methods*. London: Methuen, 1975.

[Hil93] A. Hiles. *Service level agreements: measuring cost and quality in service relationships*. Chapman & Hall, 1993.

[HJ03] T. Hagras and J. Janecek. A simple scheduling heuristic for heterogeneous computing environments. In *Proceedings of the Second International Symposium on Parallel and Distributed Computing*, pages 104–110, 2003.

[HRV07] R. Hall, A. L. Rosenberg, and A. Venkataramani. A comparison of DAG-scheduling strategies for Internet-based computing. In *Proceedings of the 21st IEEE International Parallel and Distributed Processing Symposium*, pages 1–9, 2007.

[IOF95] M. A. Iverson, F. Ozguner, and G. J. Follen. Parallelizing existing applications in a distributed heterogeneous environment. In *Proceedings of the 4th Heterogeneous Computing Workshop*, pages 93–100, 1995.

[JFI+07] S. A. Jarvis, B. P. Foley, P. J. Isitt, D. P. Spooner, D. Rueckert, and G. R. Nudd. Performance prediction for a code with data-dependent runtimes. *Concurrency and Computation: Practice and Experience*, 19:1–12, 2007.

[JHSN05] S. A. Jarvis, L. He, D. P. Spooner, and G. R. Nudd. The impact of predictive inaccuracies on execution scheduling. *Performance Evaluation*, 1-4(60):127–139, 2005.

[JMW02] W. A. Ward. Jr., C. L. Mahood, and J. E. West. Scheduling jobs on parallel systems using a relaxed backfill strategy. *Job Scheduling Strategies for Parallel Processing*, 2537:88–102, 2002.

[JSC01] D. Jackson, Q. Snell, and M. Clement. Core algorithms of the Maui scheduler. In *Job Scheduling Strategies for Parallel Processing*, volume 2221 of *Lecture Notes in Computer Science*, pages 87–102. Springer Berlin / Heidelberg, 2001.

[KA96] Y. Kwok and I. Ahmad. Dynamic critical-path scheduling: an effective technique for allocating task graphs to multiprocessors. *IEEE Transactions on Parallel and Distributed Systems*, 5(7):506–521, May 1996.

[KA99] Y. K. Kwok and I. Ahmad. Static scheduling algorithms for allocating directed task graph to multiprocessors. *ACM Computing Surveys*, 4(31):406–471, December 1999.

[KFC06] N. R. Kaushik, S. M. Figueira, and S. A. Chiappari. Flexible time-windows for advance reservation scheduling. In *Proceedings of the 14th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*, pages 218–225, 2006.

[KKP+04] L. V. Kalé, S. Kumar, M. Potnuru, J. DeSouza, and S. Bandhakavi. Faucets: Efficient resource allocation on the computational grid. In *Proceedings of the International Conference on Parallel Processing*, volume 1, pages 396–405, 2004.

[KL05] A. Kamthe and S.-Y. Lee. A stochastic approach to estimating earliest start times of nodes for scheduling DAGs on heterogeneous distributed computing systems. In *Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium*, pages 121b–121b, 2005.

[KL07] A. Kamthe and S.-Y. Lee. Stochastic approach to scheduling multiple divisible tasks on a heterogeneous distributed computing system. In *Proceedings of the 21st IEEE International Parallel and Distributed Processing Symposium*, pages 1–11, 2007.

[LAH+04] G. Laszewski, K. Amin, M. Hategan, N. J. Zaluzec, S. Hampton, and A. Rossi. GridAnt: a client-controllable grid workflow system. In *Proceedings of the 37th Hawaii International Conference on System Science*, pages 1–10, 2004.

[LCWZ03] G. Li, D. Chen, D. Wang, and D. Zhang. Task clustering and scheduling to multiprocessors with duplication. In *Proceedings of the 17th International Parallel and Distributed Processing Symposium*, pages 1–8, 2003.

[LE] LEAD: `http://portal.leadproject.org/`.

[LHS06] M. M. Lòpez, E. Heymann, and M. A. Senar. Analysis of dynamic heuristics for workflow scheduling on grid systems. In *The Fifth International Symposium on Parallel and Distributed Computing*, pages 199–207, 2006.

[LKB77] J. K. Lenstra, A. H. G. Rinnooy Kan, and P. Brucker. Complexity of machine scheduling problems. *Annals of Discrete Mathematics*, 1:343–362, 1977.

[LPX05] G. Q. Liu, K. L. Poh, and M. Xie. Iterative list scheduling for heterogeneous computing. *Journal of Parallel and Distributed Computing*, 5(65):654–665, May 2005.

[LS02] B. Lawson and E. Smirni. Multi-queue backfilling scheduling with priorities and reservations for parallel systems. In *Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing*, pages 72–87, 2002.

[LSF] Load Sharing Facility platform: `http://www.platform.com/products/LSF/`.

[LZ07] B. Li and D. Zhao. Performance impact of advance reservations from the grid on backfill algorithms. In *Proceedings of the 6th International Conference on Grid and Cooperative Computing (GCC07)*, pages 456–461, 2007.

[Mac03] J. MacLaren. Advance reservations: State of the art. *http://www.ggf.org/Meetings/ggf7/drafts/sched-graap-2.0.pdf*, 2003.

[MAD+05] S. McGough, A. Afzal, J. Darlington, N. Furmento, A. Mayer, and L. Young. Making the grid predictable through reservations and performance modelling. *The Computer Journal*, 48(3):358–368, 2005.

[MDM08] H. R. Moaddeli, G. Dastghaibyfard, and M. R. Moosavi. Flexible advance reservation impact on backfilling scheduling strategies. In *Proceedings of the 7th International Conference on Grid and Cooperative Computing (GCC08)*, pages 151–159, 2008.

[MF01] A. W. Mu'alem and D. G. Feitelson. Utilization, predictability, workloads, and user runtime estimate in scheduling the IBM SP2 with backfilling. *IEEE Transaction on Parallel and Distributed Systems*, 12(6):529–543, 2001.

[MFR07] G. Malewicz, I. Foster, and A. L. Rosenberg. A tool for prioritizing DAGMan jobs and its evaluation. *Journal of Grid Computing*, 5(2):197–212, 2007.

[MKB02] M. Maheswaran, K. Krauter, and R. Buyya. A taxonomy and survey of grid resource management systems for distributed computing. *Software: Practice and Experience*, 2(32):135–164, 2002.

[MKK+05] A. Mandal, K. Kennedy, C. Koelbel, G. Marin, J. Mellor-Crummey, B. Liu, and L. Johnsson. Scheduling strategies for mapping application workflows onto the grid. In *IEEE International Symposium on High Performance Distributed Computing (HPDC 2005)*, pages 125–134, 2005.

[MO] Montage: `http://montage.ipac.caltech.edu/`.

[MR05] G. Malewicz and A. L. Rosenberg. On batch-scheduling DAGs for Internet-based computing. In *Euro-Par 2005 Parallel Processing*, volume 3648 of *Lecture Notes in Computer Science*, pages 262–271. Springer Berlin / Heidelberg, 2005.

[MR06] G. Malewicz and A. L. Rosenberg. A pebble game for Internet-Based computing. In *Proceedings of Essays in Memory of Shimon Even'2006*, pages 291–312, 2006.

[MRY06] G. Malewicz, A. L. Rosenberg, and M. Yurkewych. Toward a scheduling theory for scheduling DAGs in Internet-based computing. *IEEE Transactions on Computers*, 55(6):757–768, 2006.

[MS98] M. Maheswaran and H. J. Siegal. A dynamic matching and scheduling algorithm for heterogeneous computing systems. In *Proceedings of the 7th Heterogeneous Computing Workshop*, pages 57–69, 1998.

[MSK+04] J. MacLaren, R. Sakellariou, K. T. Krishnakumar, J. Garibaldi, and D. Ouelhadj. Towards service level agreement based scheduling on the grid. In *Proceedings of the Workshop on Planning and Scheduling for Web and Grid Services*, pages 100–102, 2004.

[NB08] M. A. S. Netto and R. Buyya. Rescheduling co-allocation requests based on flexible advance reservation and processor remapping. In *Proceedings of the 2008 9th IEEE/ACM International Conference on Grid Computing*, pages 144–151, 2008.

[NBB07] M. A. S. Netto, K. Bubendorfer, and R. Buyya. SLA-based advance reservations with flexible and adaptive time QoS parameters. In *Proceedings of the 5th International Conference on Service-Oriented Computing (ICSOC)*, pages 119–131, 2007.

[NKP+00] G. R. Nudd, D. J. Kerbyson, E. Papaefstathiou, S. C. Perry, J. S. Harper, and D. V. Wilcox. PACE - a toolset for the performance prediction of parallel and distributed systems. *International Journal of High Performance Computing Applications*, 14(3):228–251, 2000.

[NWB08] D. C. Nurmi, R. Wolski, and J. Brevik. Varq: virtual advance reservations for queues. In *Proceedings of the 17th international symposium on High Performance Distributed Computing*, pages 75–86, 2008.

[NWB09] D. C. Nurmi, R. Wolski, and J. Brevik. Probabilistic reservation services for large-scale batch-scheduled systems. *IEEE Systems Journal*, 3:6–24, 2009.

[OAF+04] T. Oinn, M. Addis, J. Ferris, D. Marvin, M. Senger, M. Greenwood, T. Carver, K. Glover, M. R. Pocock, A. Wipat, and P. Li. Taverna: a tool for the composition and enactment of bioinformatics workflows. *Bioinformatics*, 17(20):3045–3054, 2004.

[OH96] H. Oh and S. Ha. A static heuristic for heterogeneous processors. In *Proceedings of European Conference On Parallel Processing (Euro-Par'96)*, volume 2, pages 573–377, 1996.

[OSG] Planet: `http://www.opensciencegrid.org/`.

[PE] Pegasus: `http://pegasus.isi.edu/`.

[PHP+03] P. Padala, C. Harrison, N. Pelfort, E. Jansen, M. P. Frank, and C. Chokkareddy. OCEAN: The open computation exchange and arbitration network, a market approach to meta computing. In *Proceedings of the 2nd International Symposium on Parallel and Distributed Computing (ISPDC 2003)*, pages 185–192, 2003.

[PLR+95] M. A. Palis, J. C. Liou, S. Rajasekaran, S. Shende, and D. S. L. Wei. Online scheduling of dynamic trees. *Parallel Processing Letters*, 5(4):635–646, 1995.

[QAY08] D. M. Quan, J. ALtmann, and L. T. Yang. Mapping light communication SLA-based workflows onto grid resources with parallel processing technology. In *Proceedings of IEEE International Symposium on Parallel and Distributed Processing*, pages 191–198, 2008.

[QK05] D. M. Quan and O. Kao. Mapping grid job flows to grid resources within SLA context. In *Proceedings of the European Grid Conference (EGC2005)*, pages 1107–1116, 2005.

[qos] IBM Research — Autonomic Computing — Glossary: `http://www.research.ibm.com/autonomic/glossary.html`.

[Qua06a] D. M. Quan. *A Framework for SLA-aware execution of Grid-based workflow*. PhD thesis, University of Paderborn, 2006.

[Qua06b] D. M. Quan. Mapping heavy communication workflows onto grid resource within SLA context. In *Proceedings of the International Conference of High Performance Computing and Communication (HPCC06)*, pages 727–736, 2006.

[RA00a] S. Ranaweera and D. P. Agrawal. A scalable task duplication based scheduling algorithm for heterogeneous systems. In *Proceedings of the International Conference on Parallel Processing*, pages 383–390, 2000.

[RA00b] S. Ranaweera and D. P. Agrawal. A task duplication based scheduling algorithm for heterogeneous systems. In *Proceedings of the 14th International Parallel and Distributed Processing Symposium*, pages 445–450, 2000.

[RBL+09] B. Rochwerger, D. Breitgand, E. Levy, A. Galis, K. Nagin, I. M. Llorente, R. Montero, Y. Wolfsthal, E. Elmroth, J. Caceres, M. Ben-Yehuda, W. Emmerich, and F. Galan. The reservoir model and architecture for open federated cloud computing. *IBM Journal of Research and Development*, 4(53):1–11, 2009.

[Ros04] A. L. Rosenberg. On scheduling mesh-structured computations for Internet-based computing. *IEEE Transactions on Computers*, 53:1176–1186, 2004.

[RSR06] T. Röblitz, F. Schintke, and A. Reinefeld. Resource reservations with fuzzy requests. *Journal on Concurrency and Computation: Practice and Experience*, 18(13):1681–1703, 2006.

[RSW04] T. Röblitz, F. Schintke, and J. Wendler. Elastic grid reservations with user-defined optimization policies. In *Proceedings of the Workshop on Adaptive Grid Middleware (AGridM'04)*, pages 1–9, 2004.

[RvG00] A. Răduescu and A. J. C. van Gemund. Fast and effective task scheduling in heterogeneous systems. In *Proceedings of the 9th Heterogeneous Computing Workshop*, pages 229–238, 2000.

[RY05] A. L. Rosenberg and M. Yurkewych. Guidelines for scheduling some common computation-dags for Internet-based computing. *IEEE Transactions on Computers*, 54:428–438, 2005.

[SB99] J. M. Schopf and F. Berman. Stochastic scheduling. In *Proceedings of 1999 ACM/IEEE Conference on Supercomputing*, pages 48–68, 1999.

[SB08a] T. Szepieniec and M. Bubak. Evaluation of eligible jobs maximization algorithm for DAG scheduling in grids. In *Proceedings of 8th International Conference on Computational Science*, pages 254–263, 2008.

[SB08b] T. Szepieniec and M. Bubak. Investigation of the DAG eligible jobs maximization algorithm in a grid. In *9th IEEE/ACM International Conference on Grid Computing*, pages 340–345, 2008.

[Sch03] J. M. Schopf. Ten actions when grid scheduling: The user as grid scheduler. *Grid Resource Management: State of the Art and Future Trends, ch.2*, 2003.

[SCJ+05] D. P. Spooner, J. Cao, S. A. Jarvis, L. He, and G. R. Nudd. Performance-aware workflow management for grid computing. *The Computer Journal*, 3(48):347–357, 2005.

[SCR08] M. Sims, G. Cordasco, and A. L. Rosenberg. On clustering tasks in IC-optimal DAGs. In *Proceedings of the International Conference on Parallel Processing*, pages 381–388, 2008.

[SCT+02] D. P. Spooner, J. Cao, J. D. Turner, H. N. Lin Choi Keung, S. A. Jarvis, and G. R. Nudd. Localised workload management using performance prediction and qos constracts. In *Proceedings of 18th Annual UK Performance Engineering Workshop*, pages 69–80, 2002.

[SCZL96] J. Skovira, W. Chan, H. Zhou, and D. Lifka. The EASY - LoadLeveler API Project. In *Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing*, pages 41–47, 1996.

[SF05] E. Shmueli and D. G. Feitelson. Backfilling with lookahead to optimize the packing of parallel jobs. *Journal of Parallel and Distributed Computing*, 65(9):1090–1107, 2005.

[SFT00] W. Smith, I. Foster, and V. Taylor. Scheduling with advanced reservations. In *Proceedings of IEEE International Symposium on Parallel and Distributed Processing*, pages 127–132, 2000.

[Sim78] B. Simons. A fast algorithm for single processor scheduling. In *19th Annual Symposium on Foundations of Computer Science*, pages 246–252, 1978.

[SJC+03] D. P. Spooner, S. A. Jarvis, J. Cao, S. Saini, and G. R. Nudd. Local grid scheduling techniques using performance prediction. *IEE Proceedings of Computers and Digital Techniques*, 150:87–96, 2003.

[SKD05] G. Singh, C. Kesselman, and E. Deelman. Performance impact of resource provisioning on workflows. Technical report, University of Southern California, 2005.

[SKD07] G. Singh, C. Kesselman, and E. Deelman. A provisioning model and its comparison with best-effort for performance-cost optimiazation in grids. In *Proceedings of the 16th International Symposium on High Performance Distributed Computing*, pages 117–126, 2007.

[SKSS02] S. Srinivasan, R. Kettimuthu, V. Subramani, and P. Sadayappan. Characterization of backfilling strategies for parallel job scheduling. In *Proceedings of the International Conference on Parallel Processing*, pages 514–519, 2002.

[SL93] G. C. Sih and E. A. Lee. A compile-time scheduling heuristic for interconnection-constrained heterogeneous processor architectures. *IEEE Transactions on Parallel and Distributed Systems*, 2(4):175–187, February 1993.

[SLAO06] T. Sandholm, K. Lai, J. Andrade, and J. Odeberg. Market-based resource allocation using price prediction in a high performance computing grid for scientific applications. In *Proceedings of the 15th IEEE International Symposium on High Performance Distributed Computing*, pages 132–143, 2006.

[ST04] M. Shields and I. Taylor. Programming scientific and distributed workflow with Triana services. In *Proceedings of the Workflow in Grid Systems Workshop in GGF10*. GGF:Berlin, 2004.

[SVF06] M. Siddiqui, A. Villazon, and T. Fahringer. Grid capacity planning with negotiation-based advance reservation for optimized QoS. In *Proceedings of the 2006 IEEE/ACM Conference on Supercomputing*, pages 103–118, 2006.

[SW89] J. Simposon and E. Weiner. *The Oxford English Dictionary*. Oxfod University Press, 1989.

[SY08] R. Sakellariou and V. Yarmolenko. Job scheduling on the grid: Towards SLA-based scheduling. *High Performance Computing and Grids in Action*, 16:207–222, 2008.

[SZ04a] R. Sakellariou and H. Zhao. A hybrid heuristic for DAG scheduling on heterogeneous systems. In *Proceedings of the 13th Heterogeneous Computing Workshop*, pages 111–124, 2004.

[SZ04b] R. Sakellariou and H. Zhao. A low-cost rescheduling policy for efficient mapping of workflows on grid systems. *Scientific Programming*, 4(12):253–262, December 2004.

[SZTD05] R. Sakellariou, H. Zhao, E. Tsiakkouri, and M. D. Dikaiakos. Scheduling workflows with budget constraints. In *the CoreGRID Workshop on Integrated Research in Grid Computing*, pages 347–357, 2005.

[TF99] D. Talby and D. G. Feitelson. Supporting priorities and improving utilization of the IBM SP scheduler using slack-based backfilling. In *13th International Parallel Processing Symposium*, pages 513–517, 1999.

[TG] NSF Teragrid: `http://www.teragrid.org/`.

[THW99] H. Topcuoglu, S. Hariri, and M. Wu. Task scheduling algorithms for heterogeneous processors. In *Proceedings of the 8th Heterogeneous Computing Workshop*, pages 3–14, April 1999.

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

[TKB07] A. K. M. Talukder, M. Kirley, and R. Buyya. Multi-objective differential evolution for workflow execution on grids. In *MGC '07: Proceedings of the 5th international workshop on Middleware for grid computing*, pages 1–6. ACM, 2007.

[TPS+02] I. Taylor, R. Philp, M. Shields, O. Rana, and B. Schutz. The Consumer Grid. In *Global Grid Forum (2002)*, February 2002.

[TSW03] I. Taylor, M. Shields, and I. Wang. Resource management of Triana P2P services. In *Grid Resource Management*, Kluwer, Netherlands, June 2003.

[TTL05] D. Thain, T. Tannenbaum, and M. Livny. Distributed computing in practice: the Conder experience. *Concurrency and Computation: Practice and Experience*, 17(20):323–356, 2005.

[TWML01] T. Tannenbaum, D. Wright, K. Miller, and M. Livny. Condor – a distributed job scheduler. *Beowulf Cluster Computing with Linux*, pages 307–350, 2001.

[Uet01] M. Uetz. *Algorithms for Deterministic and Stochastic Scheduling*. PhD thesis, Technischen Universität Berlin, 2001.

[WG90] M.-Y. Wu and D. D. Gajski. Hypertool: a programming aid for message-passing systems. *IEEE Transactions on Parallel and Distributed Systems*, 3(1):330–343, July 1990.

[WHP08] M. Wieczorek, A. Hoheisel, and R. Prodan. Taxonomies of the multi-criteria grid workflow scheduling problem. In *Grid Middleware and Services*, pages 237–264. Springer US, 2008.

[WI] WIEN2K: `http://www.wien2k.org/`.

[Win92] P. H. Winston. *Artificial Intelligence*. Addison-Wesley, 1992.

[WPBB01] R. Wolski, J. S. Plank, J. Brevik, and T. Bryan. Analyzing market-based resource allocation strategies for the computational grid. *International Journal of High Performance Computing Applications*, 15(3):258–281, 2001.

[WPF05] M. Wieczorek, P. Prodan, and T. Fahringer. Scheduing of scientific workflows in the ASKALON grid environment. *SIGMOD Record*, 3(34):56–62, 2005.

[WPF06] M. Wieczorek, R. Prodan, and T. Fahringer. Comparison of workflow scheduling strategies on the grid. In *Proceedings of the Second Grid Resource Management Workshop (GRMW'2005)*, pages 792–800, 2006.

[WPPF08] M. Wieczorek, S. Podlipnig, R. Prodan, and T. Fahringer. Bi-criteria scheduling of scientific workflows for the grid. In *Proceedings of the 8th IEEE International Symposium on Cluster Computing and the Grid*, pages 9–16, 2008.

[WSA] GRAAP-WG: `https://forge.gridforum.org/sf/projects/graap-wg`.

[WSRM97] L. Wang, H. J. Siegel, V. P. Roychowdhury, and A. A. Maciejewski. Task matching and scheduling in heterogeneous computing environments using a genetic-algorithm-based approach. *Journal of Parallel and Distributed Computing*, 1(47):8–22, November 1997.

[WSV+06] M. Wieczorek, M. Siddiqui, A. Villazon, R. Prodan, and T. Fahringer. Applying advance reservation to increase predictability of workflow execution. In *Proceedings of the 2nd IEEE International Conference on e-Science and Grid Computing*, pages 82–89, 2006.

[WY02] N. Woo and H. Y. Yeom. k-depth look-ahead task scheduling in network of heterogeneous processors. *Information Networking: Wireless Communications Technologies and Network Applications*, pages 736–745, January 2002.

[YB04] J. Yu and R. Buyya. A novel architecture for realizing grid workflow using tuple spaces. In *Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing (Grid 2004)*, pages 119–128, 2004.

[YB05] J. Yu and R. Buyya. A taxonomy of workflow management systems for grid computing. *Journal of Grid Computing*, 3(3-4):171–200, 2005.

[YB06a] C. S. Yeo and R. Buyya. A taxonomy of market-based resource management systems for utility-driven cluster computing. *Software: Practice and Experience*, 13(36):1381–1419, 2006.

[YB06b] J. Yu and R. Buyya. Scheduling scientific workflow applications with deadline and budget constraints using genetic algorithms. *Scientific Programming*, 14:217–230, 2006.

[YB07] J. Yu and R. Buyya. Multi-objective planning for workflow execution on grids. In *Proceedings of the 8th IEEE/ACM International Conference on Grid Computing*, pages 10–17, 2007.

[YB09] J. Yu and R. Buyya. Gridbus workflow enactment engine. In L. Wang, W. Jie, and J. Chen, editors, *Grid Computing: Infrastructure, Service, and Applications*, pages 119–146. CRC Press, 2009.

[YBR08] J. Yu, R. Buyya, and K. Ramamohanarao. Workflow scheduling algorithms for grid computing. *Studies in Computational Intelligence*, 146:173–214, 2008.

[YS06] V. Yarmolenko and R. Sakellariou. An evaluation of heuristics for SLA based parallel job scheduling. In *the 3rd High Performance Grid Computing Workshop (in conjunction with IPDPS 2006)*, pages 1–8, 2006.

[YS07] Z. Yu and W. Shi. An adaptive rescheduling strategy for grid workflow applications. In *Proceedings of IEEE International Symposium on Parallel and Distributed Processing*, pages 1–8, March 2007.

[YSOG05] V. Yarmolenko, R. Sakellariou, D. Ouelhadj, and J. M. Garibaldi. SLA based job scheduling: A case study on policies for negotiation with resources. In *Proceedings of the UK e-Science All Hands Meeting 2005 (AMH'2005)*, pages 1–8, 2005.

[Yu09] Z. Yu. *Toward Practical Multi-Workflow Scheduling in Cluster and Grid Environments*. PhD thesis, Wayne State University, 2009.

[ZBN+04] L. Zeng, B. Benatallah, A. Ngu, M. Dumas, J. Kalagnanam, and H. Chang. QoS-aware middleware for web services composition. *IEEE Transactions on Software Engineering*, 30(5):311–327, 2004.

[Zha06] H. Zhao. *Towards Realistic Scheduling for Heterogeneous Systems*. PhD thesis, The University of Manchester, 2006.

[ZS06a] H. Zhao and R. Sakellariou. Advance reservation policies for workflows. In *Job Scheduling Strategies for Parallel Processing*, volume 4376 of *Lecture Notes in Computer Science*, pages 47–67. Springer Berlin / Heidelberg, June 2006.

[ZS06b] H. Zhao and R. Sakellariou. Scheduling multiple DAGs onto heterogeneous systems. In *Proceedings of the 20th International Parallel and Distributed Processing Symposium*, pages 1–14, 2006.

[ZWF+04] Y. Zhao, M. Wilde, I. Foster, J. Voeckler, T. Jordan, E. Quigg, and J. Dobson. Grid middleware services for virtual data discovery, composition, and integration. In *the 2nd Workshop on Middleware for Grid Computing*, pages 57–62, 2004.

Appendix A

The Limitations of ICO

Figure A.1 illustrates a DAG example where ICO fails in decomposition according to the "The Divide Phase" of ICO described in [MFR07]. To remain consistent with the description, a building block, which is built from source node s, is named as $B(s)$. Based on the DAG depicted in Figure A.1, the construction steps of $B(0)$ and $B(3)$ are presented separately in Tables A.1 and A.2, and their results are depicted in Figure A.2 and A.3.

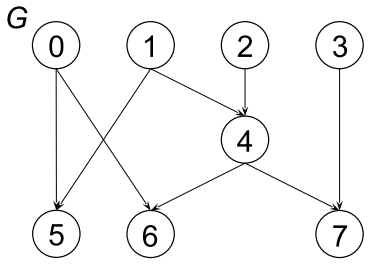


Figure A.1: A DAG example for which ICO decomposition fails

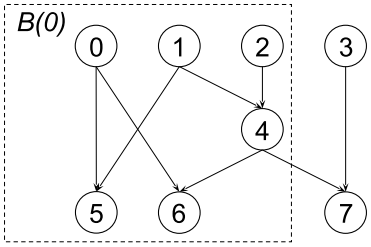


Figure A.2: A DAG example for which ICO decomposition fails

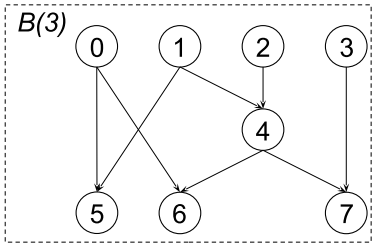


Figure A.3: Building Block $B(3)$

It is evident that $B(0)$ is not a strict sub-graph of $B(3)$, neither are they identical. In this scenario, the decomposition does not know how to handle the edge from node 4 to node 7, and therefore falls into failure. Nodes like node 4 are called **odd nodes**. They are not source nodes of G but share children with some source nodes. This bug appears when the following three conditions are met simultaneously: (I) G contains at least one odd node; (II) There is no building block which can be detached from G for the odd node to turn into a source node; (III) There is at least one building block which contains an odd node, but not all of its children.

Iteration		S	T
0		{0}	{}
1	1.1	{0}	{5,6}
	1.2	{0}	{5,6,0,1,4}
	1.3	{0,1}	{5,6,0,1,4}
2	2.1	{0,1}	{5,6,0,1,4}
	2.2	{0,1}	{5,6,0,1,4,2}
	2.3	{0,1,2}	{5,6,0,1,4,2}
3	3.1	{0,1,2}	{5,6,0,1,4,2}
	3.2	{0,1,2}	{5,6,0,1,4,2}
	3.3	{0,1,2}	{5,6,0,1,4,2}
Terminated			

Table A.1: The change of S and T when constructing building block $B(0)$

Iteration		S	T
0		{3}	{}
1	1.1	{3}	{7}
	1.2	{3}	{7,3,4}
	1.3	{3}	{7,3,4}
2	2.1	{3}	{7,3,4}
	2.2	{3}	{7,3,4,1,2}
	2.3	{3,1,2}	{7,3,4,1,2}
3	3.1	{3,1,2}	{7,3,4,1,2,5}
	3.2	{3,1,2}	{7,3,4,1,2,5,0}
	3.3	{3,1,2,0}	{7,3,4,1,2,5,0}
4	4.1	{3,1,2,0}	{7,3,4,1,2,5,0,6}
	4.2	{3,1,2,0}	{7,3,4,1,2,5,0,6}
	4.3	{3,1,2,0}	{7,3,4,1,2,5,0,6}
5	5.1	{3,1,2,0}	{7,3,4,1,2,5,0,6}
	5.2	{3,1,2,0}	{7,3,4,1,2,5,0,6}
	5.3	{3,1,2,0}	{7,3,4,1,2,5,0,6}
Terminated			

Table A.2: The change of S and T when constructing building block $B(3)$

Appendix B

Generation of Random DAGs

The randomly generated DAGs were constructed by means of the following steps:

1. Specify the number of nodes;
2. Specify the number of levels;
3. Randomly allocate the number of nodes at each level;
4. For each node except the exit, randomly appoint nodes as child nodes (at least one) in its lower neighbour level;
5. For each isolated node (non-entry node without parent), randomly appoint nodes as parent nodes in its upper neighbour level.

Figure B.1 demonstrates the procedure used to construct a DAG with 10 nodes and 5 levels, in which the light dark nodes in step 4 are isolated nodes. Obviously, no edge spanning non-neighbour levels in the DAGs will be generated by this method. Therefore, the odd nodes leading to the failure of ICO cannot appear in the generated DAGs. This guarantees the successful execution of ICO.

In the actual generation of random DAGs, the followed two parameters are considered:

1. Number of nodes
2. *HWR* (Height-Weight Ratio)

When the number of nodes is fixed, a DAG with relatively more levels usually looks longer and narrower while one with fewer levels shorter and wider.

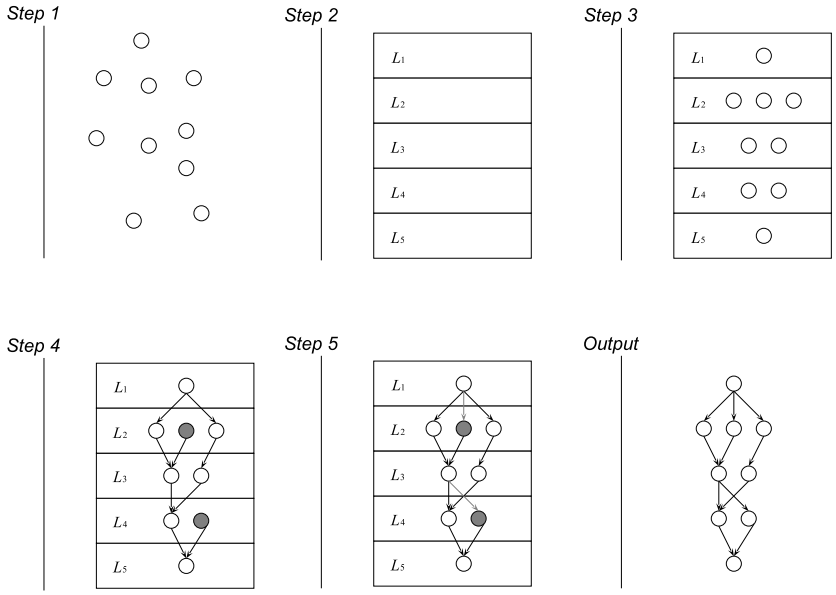


Figure B.1: Five steps to generate random DAG

The *HWR* (Height-Width Ratio) is used to identify the relationship between the number of nodes and the number of levels: $HWR = \frac{L}{\sqrt{N-2}}$, where N is the number of nodes except for entry node and exit node, and L is the number of levels.