國立交通大學

資訊科學與工程研究所

碩士論文

雲端計算環境中雙階層計畫與工作排程演算法

Two-tier Project and Job Scheduling for Cloud Computing Environments

研究生　：Thái Minh Tuấn

指導教授　：林盈達　教授

中華民國 一百零二年 六月

雲端計算環境中雙階層計畫與工作排程演算法

Two-tier Project and Job Scheduling for Cloud Computing Environments

研 究 生：蔡明俊　　**Student：Thai Minh Tuan**

指導教授：林盈達　　**Advisor：Dr. Ying-Dar Lin**

國 立 交 通 大 學

資訊科學與工程研究所

碩 士 論 文

A Thesis

Submitted to Institute of Computer Science and Engineering

College of Computer Science

National Chiao Tung University

in Partial Fulfillment of the Requirements

for the Degree of

Master

In

Computer Science

June 2013

Hsinchu, Taiwan

中 華 民 國　一 百零二年 六月

雲端計算環境中雙階層計畫與工作排程演算法

學生: 蔡明俊　　指導教授: 林盈達

國立交通大學資訊科學與工程研究所

摘要

本研究旨在解決當今雲端計算環境下的雙層排班(two-tier scheduling)問題。在此問題之中，一個計畫(project)就代表著一位雲端使用者的請求，是由多項工作(job)所組合而成，而每項工作的處理需要數種資源。研究目標是使用適合的排程演算法來縮短計畫的回復時間(turn-around time) 以及支援優先等級排程(priority scheduling) 。由於這種雙層排班問題一直以來缺少有效率之演算法，我們在此提出一組雙層回填(Two-tier Backfilling)演算法，而這組演算法乃根據著名的保守回填(Conservative Backfilling)演算法並結合計畫的寬鬆係數與優先權等概念所擴展而來。雙層嚴格回填(Two-tier Strict Backfilling, 2TSB) 演算法不允許在工作或計劃等待佇列內搶佔 (preemption) 。另一方面，可允許搶佔的雙層彈性回填(Two-tier Flexible Backfilling)演算法則有兩種版本：2TFB 和 2TFB-SF。在 2TFB 演算法中，新抵達的工作可以搶佔正在等待的工作，可是新抵達的計畫不能搶占正在等待的計畫；相比之下，2TFB-SF 演算法允許在工作和計畫佇列內搶佔。雙層優先等級回填(Two-tier Priority Backfilling, 2TPB)演算法把優先權納入考量，因此只有某些高優先權計劃能夠搶佔低優先權計畫。實驗結果指出 2TFB-SF 演算法可以縮短工作平均回復時間約 15%，而和 2TSB 相比，2TPB 演算法可以縮短高優先權計畫的平均回復時間約 25%。

關鍵字: 雲端排程、雙層、回填法、寬鬆係數、優先等級排程法

Two-tier Project and Job Scheduling for Cloud Computing Environments

Student: Thai Minh Tuan Advisor: Dr. Ying-Dar Lin

Institute of Computer Science and Engineering

National Chiao Tung University

Abstract

This study addresses a two-tier scheduling problem in a cloud computing environment. In this problem, a *project* represents a cloud user's request consisting of multiple *jobs*, and each job requires several *resources* for its processing. The goals are to reduce the *project* turn-around time and to support *priority* scheduling by employing suitable scheduling algorithms. Due to the lack of efficient algorithms for such a *two-tier* scheduling problem, here we propose a set of two-tier backfilling algorithms which extend the well-known conservative backfilling algorithm with project's *slack* and *priority* concepts. Among the proposed algorithms, Two-Tier Strict Backfilling (2TSB) does not allow preemption in job and project waiting queues. On the other hand, preemption is considered by Two-tier Flexible Backfilling (2TFB) which has two versions: 2TFB and 2TFB-SF (slack factor). In 2TFB, a new incoming project can preempt waiting *jobs* but not waiting *projects*; while 2TFB-SF permits preemption in *both* job and project waiting queues. Two-Tier Priority Backfilling (2TPB) algorithm takes priority into account such that only high-priority projects can preempt the low-priority ones. The experimental results indicate that, compared to 2TSB, 2TFB-SF could reduce the mean job turn-around time by 15% and 2TPB could reduce the mean turn-around time of high-priority projects by 25%.

Keywords: cloud scheduling, two-tier, backfilling, slack factor, priority scheduling.

Acknowledgments

I would like to express my deep thanks to my advisor, Professor Ying-Dar Lin, for his continued support throughout my study for master's degree. Sharing his life experience and research knowledge with me, Prof. Lin has enriched my graduate experience immensely. I also want to thank Professor Chih-Chang Wang. He helped me to explore interesting problems and guided me to conduct meaningful research.

I would like to thank all my graduate colleagues at High Speed Network Lab at National Chiao Tung University, Taiwan. I really appreciate the opportunity of meeting and working with each of you.

I am deeply grateful to my family. My mother has always been supporting me throughout my whole study. My father in heaven has been watching over me all the time. Last but not least, I would like to express my special thanks to my wife Bảo Thụy. She has given me love and courage during my successful study for master's degree.

Thái Minh Tuấn

Table of Contents

List of Figures

List of Tables

Chapter 1: Introduction

Cloud computing is an Internet-based computing paradigm whereby computational resources are delivered to users on demand over the Internet, in the same manner as public utilities [1]. In recent years, there have been more cloud-based service providers. Google Drive [2] is a good example of a cloud-based service in which all the infrastructure, software, and storage are hosted remotely and users only need a Web browser to access the service. One of the advantages in moving a service to a cloud is that it allows resources being shared among organizations and users in order to serve an even larger number of user's requests. With a proper job scheduling and resource allocation strategies, a cloud system can manage its resources to serve user's requests in the most efficient way. As a result, the cloud can improve resource utilization as well as reducing its service turn-around time.

Two-tier Scheduling in Cloud Environments

Scheduling strategies in a cloud environment vary, depending on the deployment model of the cloud. This work focuses on a *two-tier scheduling* problem within a cloud environment described as follows. In our studied system, a *project* represents a service request submitted by the cloud's users. Once the cloud accepts a project, it is obligated to complete a set of *jobs* belonging to the project. Each job has its own estimated *service time* which is determined during the pre-processing stage. Then, to start its processing, a job must be allocated a specific set of *resources* of certain types such as server, application, tool, storage, and network. In addition, the resource requirement of jobs may involve more than one resource type. At the moment when a project is submitted to a cloud, the project's characteristics become *available* to the cloud; the cloud must make scheduling decisions *immediately* and then inform users of when the project will be finished.

Since projects and their jobs arrive to a cloud one by one over time, the cloud must always make scheduling decisions and resource allocation decisions without knowledge of any *future* projects arrivals. This concept is called *on-line scheduling* in literature [3]. In addition, our two-tier scheduling problem differs from the traditional *one-tier* scheduling problem since a project consists of multiple jobs each requiring several resources for its processing. Although the cloud allocates resources directly to jobs, the purpose is to improve system performance with respect to *projects* instead of *jobs*.

The motivation of this study is to provide an efficient cloud service by employing suitable scheduling algorithms and resource allocation strategies for projects and their jobs. Put it in another way, we want to increase user satisfaction by reducing the project *turn-around time* – the time period from the moment a project arrives at a cloud provider to its departure. Moreover, achieving high resource utilization, as a general expectation for cloud computing, is also our objective.

Our Solution Set

The concept of two-tier scheduling has been addressed by several studies [4, 5, 6, 7] in recent years. Unfortunately, the solutions of these studies cannot be applied here since there are some differences between our scheduling model and those already existing. Therefore, to deal with such a complicated scheduling problem, we have proposed a set of algorithms based on the well-known *conservative backfilling* algorithm [8, 9] for the one-tier scheduling problems. The spirit behind the algorithm is that a job could be moved to the head of the waiting queue as long as it will not delay the execution of any *reserved* jobs. This helps increase resource utilization while decreasing the mean job waiting time and hence the mean project turnaround time in our problem. Another key advantage of adopting conservative backfilling is its *predictability* since it allows every waiting job to establish resource reservation. By doing so, each project is granted a guaranteed departure time when it is submitted to the cloud. This feature satisfies the two-tier scheduling model, and is also very useful for users to plan their work ahead of time.

We extend the conservative backfilling with the concept of *slack factor*, by which the actual departure time of reserved projects can be *relaxed* up to a certain slack, in order to make the algorithm more flexible to support *priority* scheduling where some projects have higher priority than others. The idea of slack factor is not new in the area of scheduling research, but up to now, it has not been applied to the two-tier scheduling problem like our study. So far, we have developed three core scheduling policies including *non-preemptive* waiting queues for jobs and projects, *non-preemptive* waiting queue for *projects* and *preemptive* waiting queue for *jobs*, and *preemptive* waiting queues for jobs and projects. Our method calculates the slack time of a reserved project by *multiplying* its turn-around time with the system's slack factor. Furthermore, the project's priority is also taken into account such that only high-priority projects can preempt the low-priority ones. In order to evaluate the

performance of our solution set, we have implemented a discrete-event simulator based on CSIM 20 [10].

The rest of this work is organized as follows. In Chapter 2, we first discuss the background related to our scheduling problem, including on-line scheduling problem, existing two-tier scheduling models, various backfilling algorithms, and data structures for advanced resource reservation. In Chapter 3, we introduce the formal description of our scheduling problem and its model. In Chapter 4, we describe the details of the proposed algorithms and their implementation. Simulation study and experiment results are presented in Chapter 5 to verify the performance of the algorithms. Finally, Chapter 6 concludes this work with a brief discussion of future study.

Chapter 2: Background and Related Work

This chapter first gives a brief overview about on-line scheduling problems and its difficulties. Then, some existing two-tier scheduling models are surveyed. After that, a variety of backfilling algorithms which have been widely studied in literature are discussed in depth. Finally, some existing advanced data structures for resource reservation are introduced.

2.1. On-line Scheduling

Most classical scheduling problems are concerned with *off-line* algorithms which are given *complete* information about the scheduling problem at hand and are required to output a solution, i.e., schedule, to the problem. In contrast to the off-line algorithms, an on-line scheduling algorithm is intended to address a common realistic scenario where the scheduler does not have the access to the whole *input instances* [3]. In other words, on-line scheduling decisions must always be made without knowledge of any *future* job arrivals since jobs arrive at the scheduler one by one over time. Additionally, some information about jobs, i.e. their *service time*, is unknown to the scheduler initially and during the run-time. They become known only when jobs have actually been finished. At the moment when a job is presented to the scheduler, the scheduling decision for the job has to be made before the next job arrives. Furthermore, the decision is *irreversible* once it is made, even if we find other obviously better schedules afterward.

In order to evaluate the performance of an on-line algorithm, Sleator and Tarjan [11] suggested using *competitive analysis*. In a competitive analysis, the output of an on-line algorithm is compared to an *optimal* value which might be obtained *if* the entire job inputs were known in advance as the off-line version. An on-line algorithm is ρ-*competitive* if its objective value is no more than ρ times in comparison to the optimal off-line value for any sequences of job inputs.

Since the assumptions of pure on-line scheduling make it impossible to find optimal solutions, some concepts have been introduced in order to handle the variants of this class of scheduling problems. For example, *semi-online* scheduling [12], which assumes that *partial* information about the scheduling problem are available before constructing a schedule, has attracted researchers in recent years. The authors in [13, 14] presented the concept of *delaying* the scheduling time of a job for a period of time and then applying scheduling rules to *accumulated* current jobs.

2.2. Two-tier Scheduling

There are several variants of two-tier scheduling model which have been already studied last few years. *Bag-of-Tasks (BoT)* application [4, 5] model, which is an application whose tasks are *identical* and *independent*, is often considered as a suitable model for heterogeneous clusters and desktop grid environments. In order to minimize the maximum *stretch*, i.e., the maximum ratio between the actual time a BoT application has spent in the system and the time this application would have spent if executed alone, the authors in [4] introduced two algorithms: an optimal off-line and a heuristic on-line for the model. In [5], a set of task selection polices is proposed in order to minimize the turn-around time of BoT applications.

Gopalan and Chiueh [6] designed and implemented a scheduler for *periodic soft real-time* applications with the goal of maximizing the number of applications admitted into the system. A periodic soft real-time application consists of a sequence of tasks whose execution *repeats itself* over the lifetime of the application, and there is a *precedence constraint* among the tasks in an application. Furthermore, the execution of a sequence of tasks requires time-bound completion.

The two-tier scheduling model which is introduced in [7] allows *preemption in resources*. In other words, the processing of a job, which represents the ownership of resources, may be preempted by another job before its completion. An algorithm which aims to prevent *deadlock* is also proposed in this work.

2.3. Backfilling Algorithm

The backfilling algorithm, a way to balance between the goals of utilizing system resources and maintaining the FCFS (first come, first served) order of job execution [15], was first introduced by Lifka [16]. The implication behind the algorithm is that it allows small jobs from the *back* of the waiting queue to be processed *before* previously submitted jobs that are delayed due to the *insufficiency* of available resources. This principle helps exploit *idle* resources by backfilling with suitable jobs, thereby increasing system utilization and throughput. Figure 1 illustrates the difference between the traditional FCFS scheduling and backfilling scheduling. In Figure 1, backfilling scheduling allows Job B to be processed ahead of Job A; therefore, the resultant job waiting time and resource idle time are reduced significantly in comparison with those of FCFS. Backfilling scheduling might lead to *"starvation"*, a phenomenon where some jobs never occupy sufficient resources and

hence never start processing because they are constantly delayed by new job arrivals that are granted the use of resources ahead of those already waiting in the queue. In order to prevent starvation from happening, a backfilling algorithm needs to make resource *reservation* for some of the waiting jobs for future time *in advance*. One should notice here that backfilling algorithms require job *service time* to be known in advance, which in practice is often specified by an *upper-bound*.

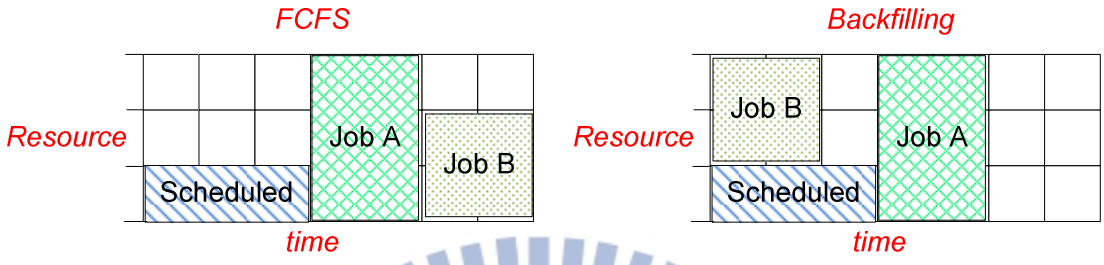


Figure 1: Illustration of how backfilling can reduce job waiting time and idle time of resources.

Aggressive vs. Conservative

In contrast to aggressive backfilling, *conservative backfilling* [8, 9] makes reservation for *every* queued job which cannot be executed at a given moment. It means that a job can be backfilled on the condition that it does not *delay* any previous jobs in the queue. Clearly, this reduces the number of jobs that can utilize idle resources. As a result, its performance tends to be inferior to that of aggressive backfilling. For the performance comparison between these two approaches, Mu'alem and Feitelson [8] showed that the performance of aggressive backfilling algorithm is better than that of conservative backfilling in most cases. However, conservative backfilling can remove the above-mentioned weakness of aggressive backfilling because of its ability to *guarantee* job starting time by establishing resource reservation for *every* waiting job.

There are several variants of backfilling algorithms. The most popular one is *aggressive backfilling* [15, 16], in which only the first job in the queue can receive a resource reservation. To put it another way, if an arrived job is the *first* job in the queue and *cannot* be processed immediately, the algorithm calculates the *earliest possible* starting time for this job using its resource requirement and service time; then, the scheduler makes a *reservation* for this job at this pre-calculated time. Other jobs are allowed to backfill *only if* they do not violate this reservation. The core problem of aggressive backfilling is its unpredictability since waiting jobs, *except* the first one, do

not get reservations. Therefore, the algorithm cannot give every job in the queue a guaranteed starting time.

Some variants of backfilling algorithms between aggressive and conservative backfilling, for example making reservation for the *first few* jobs in the waiting queue, have also been introduced [17, 18]. The ideas of using an *adaptive* number of reservations were presented by the authors of [17]. In this strategy, jobs are not necessarily given reservations *until* their expected turn-around time exceeds some *threshold*, whereupon they get a reservation. Chiang et al. [18] suggested that *four* is a good number of reservations for compromise between aggressive and conservative backfilling.

Slack-based Backfilling

In the original backfilling algorithm, a newly arriving job can be backfilled as long as it does not delay any existing reservations. In order to make backfilling scheduling more flexible and increase resource utilization, slack-based backfilling algorithms [19, 20, 21, 22] have introduced the concept of *slack factor*, by which the actual starting time of reserved jobs can be *relaxed* up to a certain slack. In other words, a newly submitted job can move to the head of the waiting queue on the condition that it will not delay already existing reservations by more than a specific slack factor. In those algorithms, the system's slack factor is used to control for how long jobs will have to wait before the start of execution.

The idea of slack factor has already been introduced to real time scheduling, parallel scheduling, and grid scheduling environments, and has been confirmed to be effective [15]. *Dynamic backfilling* allows the scheduler to *overrule* a previous reservation by a slight delay if doing so can improve system utilization considerably [19]. In order to enhance backfilling and support priority scheduling, Talby and Feitelson [20] combined three parameters – the target job's individual priority, tunable system slack factor, and the average job waiting time – to assign each waiting job a *slack value*. The authors also provided several heuristics to reduce the search space of finding the least costly schedule profile from all possible candidates. The cost of a schedule is the *sum* of costs of all its jobs, and the cost of a job is calculated based on its delay and resource requirements. In [21], Ward et al. suggested the use of a relaxed backfilling strategy in which a backfill candidate is selected from the job waiting queue by considering its waiting time, estimated service time and resource requirement together. Bo Li et al. [22] introduced an approach different from previous

algorithms such that the slack factor is calculated based on each job's service time and slack-based backfilling with more than one reservation is supported.

Other Variants of Backfilling

Lawson and Smirni [23] introduced *multiple-queue backfilling* which divides the system resources into multiple *disjoint partitions*. Each partition is associated with an individual queue, and a submitted job is assigned to a partition and hence the associated queue based on its estimated service time. The approach aims at reducing fragmentation of system resources reducing the likelihood that a short job is queued behind a long job. *Backfilling with lookahead* [24] algorithms make scheduling decisions by considering a set of jobs at once. The algorithms look ahead into the job queue and try to find a packing of jobs which maximizes the scheduler's objective using a *dynamic programming* technique.

2.4. Data Structure for Advanced Resource Reservation

Advanced resource reservation is a process of requesting resources for use at specific *future* times [25]. In the cloud environment, it is challenging for the scheduler to manage available resources and to allocate them to jobs efficiently because of the *large* number of resources and submitted jobs. Choosing a suitable data structure for advanced resource reservation could significantly affect quality of service of a cloud. The most common operations for such a data structure are searching available resources, adding new reservations and deleting existing ones. In general, data structures for advanced resource reservation can be classified into two types: *discrete* [25, 26, 27, 28] and *continuous* [29]. In the discrete data structure, the reservation time is divided into time slots each of which represents a computation time unit. On the other hand, each request defines its own time scale in the continuous case.

Many data structures for advanced reservation have been proposed and widely studied in literature. A *tree-based* data structure is commonly used for admission control in network bandwidth reservation [26, 27], where each tree node represents a time *interval* and the amount of reserved bandwidth in its sub-trees. Brown et al. proposed *Calendar Queue* [28] as a priority queue for future event set problems in discrete event simulation. In the Calendar Queue, events are stored in *buckets* which represent a fixed small time interval. Then, events which are scheduled at the same time interval are stored in a *sorted linked list*. In [25], Sulistio et al. proposed GarQ (Grid Advance Reservation Queue), which combines Calendar Queue and Segment

Tree, for administering advanced reservation in the grid environments. Qing Xiong et al. [29] introduced a linked-list data structure for advanced reservation admission control. Among afore-mentioned data structures, the linked list is the simplest and most flexible at all since accepted reservations can be inserted into the list based on their starting time. Operations can be easily performed on the linked-list data structure by iterating through the list from the head node. However, if there are *many* reservations for *small* time intervals, the linked-list data structure can become very inefficient for running these operations since it needs to *traverse* through the linked list to find the correct position for each reservation.

Chapter 3: Problem Description

This chapter first describes the system model and scheduling models of our work. Then, the problem statement is introduced. Figure 2 provides a schematic description of our scheduling model, where the notations for this model are shown in Table 1.

3.1. Two-tier System and Scheduling Models

System Model

In our system model, the cloud has N types of resources. Each type of resources has a limited capacity, and capacity M_i denotes the maximum number of type-i resources that are available for use simultaneously. A resource in the cloud can be allocated to only one job at any time; i.e., a resource cannot be shared among multiple jobs concurrently. Let $R = \{M_i \mid 1 \leq i \leq N\}$ denote a set of N types of resources of the system.

A project with multiple jobs represents a request submitted by the cloud's users. Let $P = \{p_u \mid 1 \leq u \leq |P|\}$ denote a set of projects of the cloud where $|P|$ is the number of projects. A project p_u arrives to the cloud at time ta_u. It is also the earliest time when the cloud can start processing the jobs that belong to p_u. Let $J_u = \{j_{u,v} \mid 1 \leq v \leq |J_u|\}$ denote a set of $|J_u|$ jobs that are to be processed for project p_u.

The processing of job $j_{u,v}$ requires a *service time* $te_{u,v}$. The moment when the processing of job $j_{u,v}$ begins is referred to as its *starting time* $ts_{u,v}$, and the moment when the processing of job $j_{u,v}$ is completed is referred to as its *finish time* $tf_{u,v}$. Let $lts_{u,v}$ denote the *latest starting time* of job $j_{u,v}$. The resource requirement of job $j_{u,v}$ is given by $E_{u,v} = \{q_{u,v}^i \mid 1 \leq i \leq N\}$ where $q_{u,v}^i$ is the number of type-i resources required by $j_{u,v}$, $0 \leq q_{u,v}^i \leq M_i$.

It is assumed that job service time and resource requirement are precisely determined during the pre-processing stage. Besides, there is *non-precedence* constraint between jobs. In other words, the cloud can process a set of jobs in any order. It is further assumed that the processing of a job is *non-preemptive*. Once it is started, it cannot be stopped until its completion. Next we define important time notations for projects.

Definition 1 (Project starting time): the starting time tc_u of a project p_u, defined as

$$tc_u = Min(J_u.ts) \quad \text{where} \quad J_u.ts = \{ ts_{u,v} \mid 1 \le v \le |J_u| \}, \qquad (1)$$

is the time moment when the *first* job of p_u starts its processing.

Definition 2 (Project departure time): the departure time td_u of a project p_u, defined as

$$td_u = Max(J_u.tf) \quad \text{where} \quad J_u.tf = \{ tf_{u,v} \mid 1 \le v \le |J_u| \}, \qquad (2)$$

is the time moment when the *last* job of p_u finishes its processing.

Definition 3 (Project waiting time): the waiting time tw_u of a project p_u, defined as

$$tw_u = ta_u - tc_u, \qquad (3)$$

is the time period from its arrival time ta_u to its starting time tc_u.

Definition 4 (Project running time): the running time tr_u of a project p_u, defined as

$$tr_u = td_u - tc_u, \qquad (4)$$

is the time period from its starting time tc_u to its departure time td_u.

Definition 5 (Project turn-around time): the turn-around time tn_u of a project p_u, defined as

$$tn_u = td_u - ta_u, \qquad (5)$$

is the time period from its arrival time ta_u to its departure time td_u.

Table 1: Notations used in the two-tier system and scheduling models.

Notations	Meaning
Resource	
$R = \{M_i \mid 1 \le i \le N\}$	A set of N types of resources of the system where M_i is the capacity of type-i resources
Project	
$P = \{ p_u \mid 1 \le u \le \|P\| \}$	A set of projects of the cloud where p_u is the u-th project and $\|P\|$ is the number of projects
$J_u = \{ j_{u,v} \mid 1 \le v \le \|J_u\| \}$	The set of jobs which are required to be processed for the u-th project where $j_{u,v}$ is the v-th job of the u-th project and $\|J_u\|$ is the number of jobs
ta_u	The arrival time of the u-th project
tc_u	The starting time of the u-th project; $tc_u = Min(J_u.ts)$ where $J_u.ts = \{ ts_{u,v} \mid 1 \le v \le \|J_u\| \}$
td_u	The departure time of the u-th project; $td_u = Max(J_u.tf)$ where $J_u.tf = \{ tf_{u,v} \mid 1 \le v \le \|J_u\| \}$
tw_u	The waiting time of the u-th project; $tw_u = ta_u - tc_u$
tr_u	The running time of the u-th project; $tr_u = td_u - tc_u$
tn_u	The turn-around time of the u-th project; $tn_u = td_u - ta_u$
$\overline{PTA}$	The mean turn-around time of projects; $\overline{PTA} = \frac{1}{\|P\|} \sum_{u=1}^{\|P\|} TA_u$, where |P| is the number of projects
Job	
$te_{u,v}$	The service time of the v-th job of the u-th project
$ts_{u,v}$	The starting time of the v-th job of the u-th project
$lts_{u,v}$	The latest starting time of the v-th job of the u-th project
$tf_{u,v}$	The finish time of the v-th job of the u-th project
$E_{u,v} = \{q^i_{u,v} \mid 1 \le i \le N\}$	The resource requirement of the v-th job of the u-th project where $q^i_{u,v}$ is the number of resource type i

Scheduling Model

Figure 2 illustrates the scheduling model of this work. In the system model, projects and their jobs arrive to the cloud one by one over time. When project p_u and its jobs J_u arrive to the cloud, these jobs could be processed immediately if resources are sufficient and the processing does not delay any existing reservations over the maximum number of allowable times. Otherwise, the cloud scheduler must make the scheduling decision immediately. The scheduling decision determines the starting time $ts_{u,v}$ for each job $j_{u,v} \in J_u$ and the departure time tn_u of the project p_u. Then, the cloud scheduler calculates the slack time for the project and the latest starting time for the jobs. After that, these jobs are granted resource reservations. One should notice that here new incoming jobs can preempt a job in the queue by its latest starting time. Similarly, the departure time of a waiting project may be delayed by new incoming projects several times before the project actually departs from the cloud, but it cannot exceed the slack time of the project.

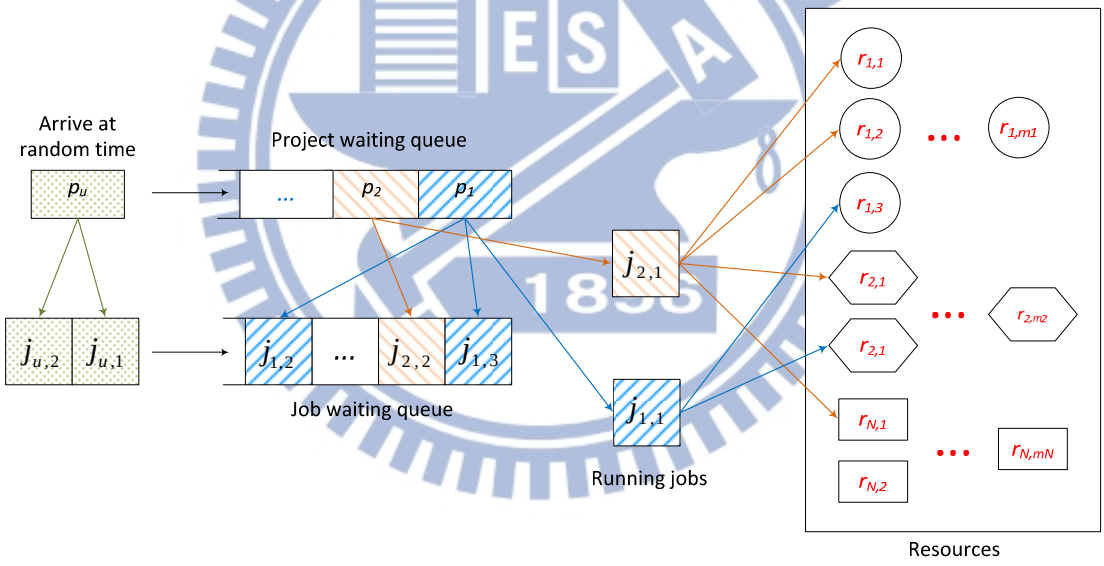


Figure 2: Two-tier project and job scheduling model.

A good practical example for our two-tier scheduling model is test cloud which provides testing services for customers across the Internet. In a test cloud, a test project represents a test request submitted by the cloud's customers. Once the cloud accepts a test project, it is obligated to complete a set of test jobs for system under test specified in the test project. The processing of a test job requires a certain amount of test resources such as test machines, test equipment, etc. Since the cloud's customers need to plan their work ahead of time, the departure time of submitted test projects

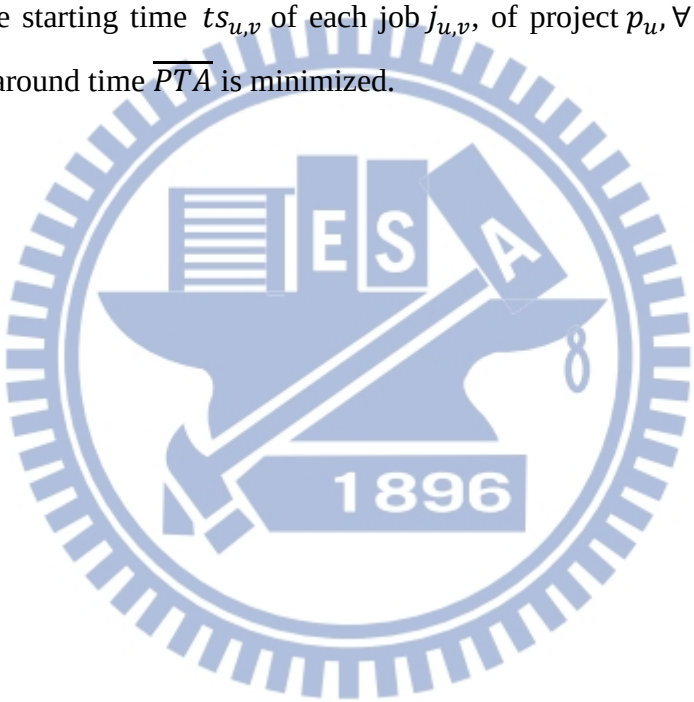
should be informed to customers at the time moment when the projects are accepted by the cloud.

3.2. Problem Statement

The problem statement of our work can be described as follows. Given a set of resources R and a set of projects P, each project $p_u \in$ P consists of a set J_u of jobs. Each job $j_{u,v} \in J_u$ requires a set of resources $E_{u,v}$ for its processing. The formula

$$\overline{PTA} = \frac{1}{|P|} \sum_{u=1}^{|P|} tn_u, \qquad (6)$$

is to calculate the mean turn-around time of projects. The objective of this work is determining the starting time $ts_{u,v}$ of each job $j_{u,v}$, of project $p_u, \forall\, p_u \in$ P such that the mean turn-around time $\overline{PTA}$ is minimized.



Chapter 4: Two-tier Project and Job Scheduling with Backfilling, Slack Factor and Priority

In this chapter, we describe our proposed scheduling policies, developed for solving the two-tier scheduling problem defined in chapter 3. In Section 4.1, we give an overview on two-tier backfilling with a slack factor of project turn-around time. The details of our approach are elaborated in Section 4.2, Section 4.3, and Section 4.4. Finally, Section 4.5 introduces a data structure for resource reservation which is used to implement our proposed scheduling policies.

4.1. Approach Overview

As mentioned in the first chapter, one of the basic requirements for our scheduling model is *predictability*. In other words, a project should be granted a guaranteed departure time at the project's arrival time. This requirement can be satisfied by conservative backfilling algorithm since it provides resource reservation for every waiting job. Besides, the system must calculate a precise estimate on job service time and resource requirement before applying the scheduling algorithm, which is satisfied by our scheduling model as well. From this point of view, the choice of conservative backfilling for our problem is straightforward.

In order to make the algorithm more flexible and support priority scheduling, we enhance conservative backfilling with the concept of slack factor, by which the departure time of reserved projects can be delayed for up to a certain slack time. The idea of slack factor has already been introduced to many scheduling problems, and has been confirmed to be effective in solving these problems [19, 20, 21, 22]. However, up to now, it has not been applied to two-tier scheduling problem like our study does.

Our method calculates the slack of each project p_u by *multiplying* its turn-around time tn_u with a system parameter slack factor (SF). The actual departure time of the project can be relaxed to a value in the time range $[td_u, td_u + tn_u \cdot SF]$. When the latest departure time of the project is determined, we can easily calculate the *latest* starting time $lts_{u,v}$ of each job $j_{u,v}$ which belongs to J_u by

$$lts_{u,v} = td_u + tn_u \cdot SF - te_{u,v}. \qquad (7)$$

Newly arriving jobs cannot delay job $j_{u,v}$ beyond its latest starting time $lts_{u,v}$ set by the cloud scheduler.

Furthermore, another system parameter PL (preemption limit) is also introduced to control the number of *preempted* projects not to exceed PL. The implication of using this parameter is that we can limit the number of projects whose departure time will be re-arranged. By doing so, the behavior of the cloud scheduler is controlled as well.

Job finish time and project departure time could be determined at the project arrival time or be relaxed later. In general, we have three core scheduling policies as shown in Table 2 where the last one has two versions: single type of projects, two types of projects (high-priority and low-priority ones). For each scheduling policy, we have developed a scheduling algorithm with particular attributes, i.e. objective, priority scheduling, etc. Hence, it is responsibility of the cloud administrator to select an appropriate scheduling policy.

Table 2: Two-tier scheduling policies

Scheduling policies		**Objective**	**Algorithm**	**Priority scheduling**	**Job finish time**	**Project departure time**
Non-preemptive waiting queues for jobs and projects		Minimize project turn-around time	Two-tier Strict Backfilling	No	Determined at project arrival time	Determined at project arrival time
Non-preemptive waiting queue for projects and preemptive waiting queue for jobs		Minimize project turn-around time	Two-tier Flexible Backfilling (SF = 0)	No	Flexible	Determined at project arrival time
Preemptive waiting queue for jobs and projects	Single type of projects	Minimize project turn-around time	Two-tier Flexible Backfilling (SF > 0.0)	No	Flexible	Flexible
	Two types of projects	Minimize high-priority project turn-around time	Two-tier Priority Backfilling	Yes	High-priority: Determined at project arrival time Low-priority: Flexible	High-priority: Determined at project arrival time Low-priority: Flexible

4.2. Non-preemptive Waiting Queues for Jobs and Projects

In this scheduling policy, job finish time $tf_{u,v}$ and project departure time td_u are determined when they arrive to the system. New jobs can be backfilled only if they do not delay any existing reservations. For this policy, we have designed Two-tier Strict Backfilling (2TSB) algorithm whose pseudo-code is showed in Figure 3.

2TSB is an algorithm similar to *first-fit* conservative backfilling since submitted jobs are scheduled at the *earliest possible* starting time. When project p_u and its jobs J_u arrive to the system, they are scheduled as follows: for each job $j_{u,v}$ of p_u, if it is feasible to allocate enough resources for $j_{u,v}$ and the processing of $j_{u,v}$ does not delay any existing reservations, $j_{u,v}$ will be backfilled to start immediately (line 4). Otherwise, the earliest possible starting time of $j_{u,v}$ is determined (line 6) by the operation $earliestStartingTime$. Since the algorithm does not allow newly arrived jobs to delay existing reservations, $ts_{u,v}$ and $lts_{u,v}$ are the same (line 7). Finally, the job is granted a resource reservation (line 8).

Algorithm 1: $\boldsymbol{Two\_Tier\_StrictBackfilling}(p_u)$

```
1   Begin
2       for each j_{u,v} ∈ J_u do
3           If ( j_{u,v} can start immediately) then
4               start j_{u,v}
5           else
                # Find the earliest starting time for job j_{u,v}
6               ts_{u,v} ← earliestStartingTime (j_{u,v})
                # Set up a resource reservation for the job j_{u,v}
7               lts_{u,v} ← ts_{u,v}
8               addReservation(j_{u,v})
9           end-if
10      end-for
11  End
```

Figure 3: Pseudo code of Two-tier Strict Backfilling algorithm

4.3. Non-preemptive Waiting Queues for Projects and Preemptive Waiting Queue for Jobs

The design philosophy behind this policy is that we try to accommodate newly submitted jobs by delaying the *job* finish time of reserved ones without changing the departure time of the waiting *projects*. Put it in another way, newly arrived jobs can delay only the starting time of any queued jobs which are not the *last* job of the waiting projects, i.e. the job with the *largest* finish time. Given slack factor $SF = 0$, the *latest starting time* $lts_{u,v}$ of job $j_{u,v}$ cannot be greater than $td_u - te_{u,v}$. Two-tier Flexible Backfilling (2TFB) algorithm ($SF = 0$) is introduced to deal with this policy. Figure 4 describes the pseudo code of the algorithm, and the steps of the algorithm are summarized as follows: the steps listed on lines 2-4 are similar to those of 2TSB since job $j_{u,v}$ can start immediately as long as there are sufficient resources for job $j_{u,v}$ and

no existing reservations are delayed over their latest starting time. Otherwise, the set of all feasible backfilling times, BT, is determined for job $j_{u,v}$ (line 6) based on the current scheduling plan by the operation $feasibleBackfillingTimes$. A feasible backfilling time $bt \in BT$ is the starting time of a gap in the job waiting queue at which $q_{u,v}^{i} \leq a_i, \forall i: 1 \leq i \leq M$, where a_i is the number of available type-i resources at the time slot bt. Then, we set up a resource reservation for job $j_{u,v}$ (line 10) at a feasible backfilling time $bt \in BT$. After that, the operation $shiftReservations$ (line 11) checks the availability of system resources and may relax some existing reservations if necessary. One should notice that if there is more than one possible reservation which could be delayed by the operation $shiftReservations$, the reservation with the *largest* latest starting time is chosen. In this scheduling policy, the number of allowable preempted projects PL is obviously *zero*. If the operation $shiftReservations$ fails and job $j_{u,v}$ cannot be backfilled at bt, then the current scheduling plan S_{old} is restored (line 13) and the *next* feasible backfilling time is considered. Note that there is always at least one feasible backfilling time at which the job can be backfilled successfully, that is the job's earliest possible starting time. After scheduling all jobs $j_{u,v} \in J_u$ and determining the turn-around time tn_u successfully, the algorithm updates the latest starting time $lts_{u,v}$ for each job (line 21-22).

4.4. Preemptive Waiting Queues for Jobs and Projects

This scheduling policy allows both job finish time and project departure time to be re-arranged after a project has been accepted only if the resultant delay does not exceed the maximum amount of delay defined by the slack factor. Furthermore, we extend the policy into two versions: single type of projects, and two types of projects. In the former, all projects have the same priority, which means that new projects can delay any existing reservations up to the maximum number of allowable times. On the other hand, the latter policy takes priority into account such that only some high-priority projects can preempt the low-priority ones.

Single Type of Project

The difference between this policy and 2TFB is slack factor $SF > 0$. As a result, both waiting projects and jobs could be preempted by the newly arrived ones. The maximum amount of allowable delay for project p_u is $tn_u \cdot SF$, and the *starting* time of job $j_{u,v}$ can be relaxed to a value in the time range $[ts_{u,v}, td_u + tn_u \cdot SF -$

$te_{u,v}$]. This, in comparison with 2TFB, could increase the number of successfully backfilled jobs. Two-tier Flexible Backfilling algorithm with the slack factor $SF > 0$ (2TFB-SF) can handle this scheduling policy. Besides, 2TFB-SF uses parameter $PL > 0$ to control the number of preempted projects.

Algorithm 2: $\boldsymbol{Two\_Tier\_FlexibleBackfilling}(p_u, SF, PL)$

```
1   Begin
2       for each j_{u,v} ∈ J_u do
3           If ( j_{u,v} can start immediately) then
4               start j_{u,v}
5           else
                # Find all the feasible backfilling times for the job j_{u,v}
6               BT ← feasibleBackfillingTimes (j_{u,v})
7               Let S_old be the current scheduling plan
8               for each bt ∈ BT do
9                   ts_{u,v} ← bt
                    # Set up a resource reservation for job j_{u,v}
10                  addReservation(j_{u,v})
                    # Check the availability of system resource and
                    # delay existing reservations in order to
                    # accommodate job j_{u,v} if necessary.
11                  succeed ← shiftReservations(j_{u,v}, PL)
12                  if (! succeed) then
13                      restore S_old
14                  else
15                      break
16                  end-if
17              end-for
18          end-if
19      end-for
20      for each j_{u,v} ∈ J_u do
            # Update the latest starting time for the resource reservation of
            # job j_{u,v}
21          lts_{u,v} ← td_d + (tn_{u,v} · SF) − te_{u,v}
22          updateLatestStartingTime(j_{u,v})
23      end for
24  End
```

Figure 4: Pseudo code of Two-tier Flexible Backfilling algorithm

Two Types of Projects

This scheduling policy considers a realistic scenario of a cloud environment where some projects are more important than the others. Therefore, they need to be severed as soon as possible. Here we just consider two types of projects in this scheduling policy, i.e., high-priority project and low-priority project. We design this

Two-tier Priority Backfilling (2TPB) in order to fulfill the requirement of the scheduling policy such that high-priority projects are scheduled by 2TFB-SF while low-priority projects are scheduled by 2TSB. Moreover, the priority of a project, $0 \leq pi_u \leq 1$, is taken into account in *recalculating* the slack factor, $SF = (1 - pi_u) \cdot SF$, for each project.

The steps of 2TPB are briefed in Figure 5. If the newly arrived project p_u is high-priority, it is scheduled by 2TFB-SF (line 4) with slack factor SF recomputed for p_u (line 3). Otherwise, it is scheduled by 2TSB (line 6). After that, we update the *latest* starting time $lts_{u,v}$ for each job $j_{u,v} \in J_u$ with recomputed slack factor SF (line 8-10).

Algorithm 3: $\boldsymbol{Two\_Tier\_PriorityBackfilling}(p_u, SF, PL)$

```
1   Begin
2       if (p_u is high_priority)
3           SF ← (1 − pi_u) * SF
4           Two_Tier_FlexibleBackfilling(p_u, SF, PL)
5       else
6           Two_Tier_StrictBackfilling(p_u)
7           for each j_{u,v} ∈ J_u do
8               SF ← (1 − pi_u) · SF
9               lts_{u,v} ← td_u + (tn_{u,v} · SF) − te_{u,v}
10              updateLatestStartingTime(j_{u,v})
11          end-for
12      end-if
13  End
```

Figure 5: Pseudo code of Two-tier Priority Backfilling algorithm

4.5. Case study: An Example Run for 2TSB and 2TFB-SF

An example run for 2TSB and 2TFB-SF is illustrated in Figure 6. Note that there are two types of resources. Thus, we plot the schedules in two blocks. Table 3 depicts the inputs of these two algorithms, which are a sequence of four submitted projects and their parameters.

In Figure 6(a), the schedule for the first 4 jobs is the same for both algorithms at time slot 2. Job $j_{3,1}$ is backfilled successfully at time slot 2 by both algorithms. The difference appears when job $j_{3,2}$ is scheduled. In 2TSB, a potential time slot for job $j_{3,2}$ to run is time slot 3, but doing so would delay the reservation of job $j_{2,1}$, which is *not* allowed by 2TSB. Therefore, job $j_{3,2}$ is scheduled at time slot 7. On the other hand, job $j_{3,2}$ could begin to execute at time slot 3 under 2TFB. Doing so would delay

job$j_{2,1}$'s execution from time slot 6 to time slot 7, which is *acceptable* because the latest starting time of job $j_{2,1}$ is 7.2 ($lts_{2,1} = 7.0 + 6.0 \cdot 0.2 - 1.0 = 7.2$).

When job $j_{4,1}$ arrives, it is scheduled by those two algorithms as shown in Figure 6(b). Time slot 6 is a potential flexible backfilling time for the job. Unfortunately, this choice will delay job $j_{2,1}$ *beyond* its latest starting time (see Figure 6(c)).

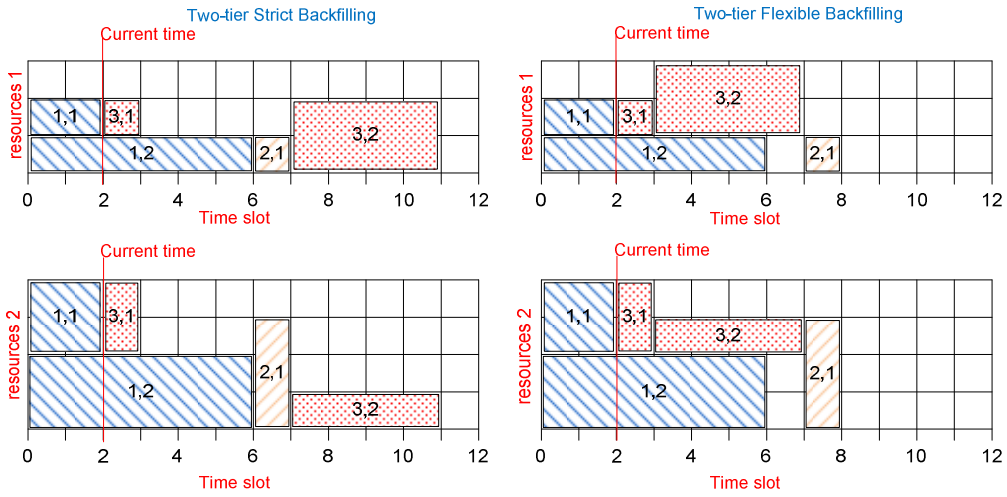
The performance of these two algorithms is compared in Table 4. As can be seen from the table, the mean project turn-around time $\overline{PTA}$ is reduced around 8% by 2TFB-SF in comparison with the 2TSB's performance.

Table 3: A sequence of four submitted projects and their parameters

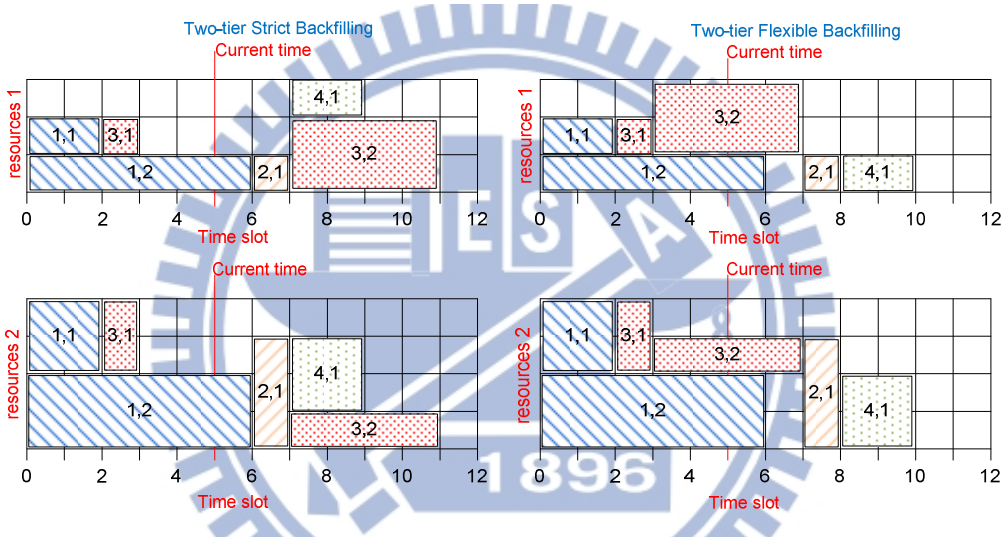
p_u	p_1		p_2	p_3		p_4
$j_{u,v}$	$j_{1,1}$	$j_{1,2}$	$j_{2,1}$	$j_{3,1}$	$j_{3,2}$	$j_{4,1}$
ta_u	0		1	2		5
$te_{u,v}$	2	6	1	1	4	2
$q^1_{u,v}$	1	1	1	1	2	1
$q^2_{u,v}$	2	2	3	2	1	2
N	2					
M_1	3					
M_2	4					
SF	0.2					
PL	∞					

Table 4: Performance of an example run for 2TSB and 2TFB-SF

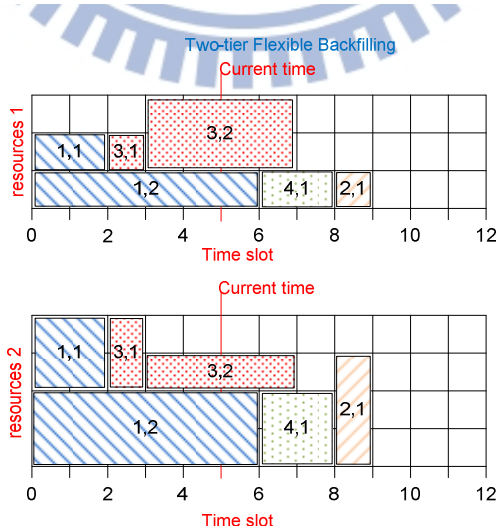
	p_u	p_1		p_2	p_3		p_4
	$j_{u,v}$	$j_{1,1}$	$j_{1,2}$	$j_{2,1}$	$j_{3,1}$	$j_{3,2}$	$j_{4,1}$
	te_u	0		1	2		5
Two-tier Strict backfilling	$tf_{u,v}$	2	6	7	3	11	9
	tn_u	6		6	9		4
	$\overline{PTA}$	6.25					
Two-tier Flexible backfilling	$tf_{u,v}$	2	6	8	3	7	10
	tn_u	6		7	5		5
	$\overline{PTA}$	5.75					



(a) Schedules for the first four jobs are the same for both the algorithms. $Job_{3,2}$ is moved to the head of the waiting queue by Two-tier Flexible Backfilling algorithm.



(b) Schedules for four projects



(c) Time slot 6 is a potential backfilling time slot for $job_{4,1}$'s execution. Unfortunately, doing so will delay $job_{2,1}$ beyond its latest starting time.

Figure 6: An example run for 2TSB and 2TFB

4.6. Two-tier Backfilling Implementation

In order to implement the proposed algorithms, it is important to organize the information of resource availability and reservations in a data structure which can provide efficient operations for searching, adding, deleting, and updating. In this section, we introduce a data structure for advanced resource reservation which is used to implement our proposed scheduling algorithms. The data structure is based on the linked-list data structure [29] because of its simplicity and flexibility.

The description of our proposed data structure is illustrated in Figure 7, while Table 5 shows the implemented operations on the data structure. The data structure is a linked-list-based data structure. Each node in the list is defined as a $node(timeslot, availability, reservation)$, where $timeslot$ denotes a time moment at which changes in reservations or resource availability occur, $availability$ denotes the number of available resources from the node to the next node, and $reservations$ is a linked-list of resource reservation records at $timeslot$. Each record is a 5-tuple information consisting of project index u, job index v, job service time $te_{u,v}$, the latest starting time of job $lts_{u,v}$ and the resource requirement $E_{u,v}$.

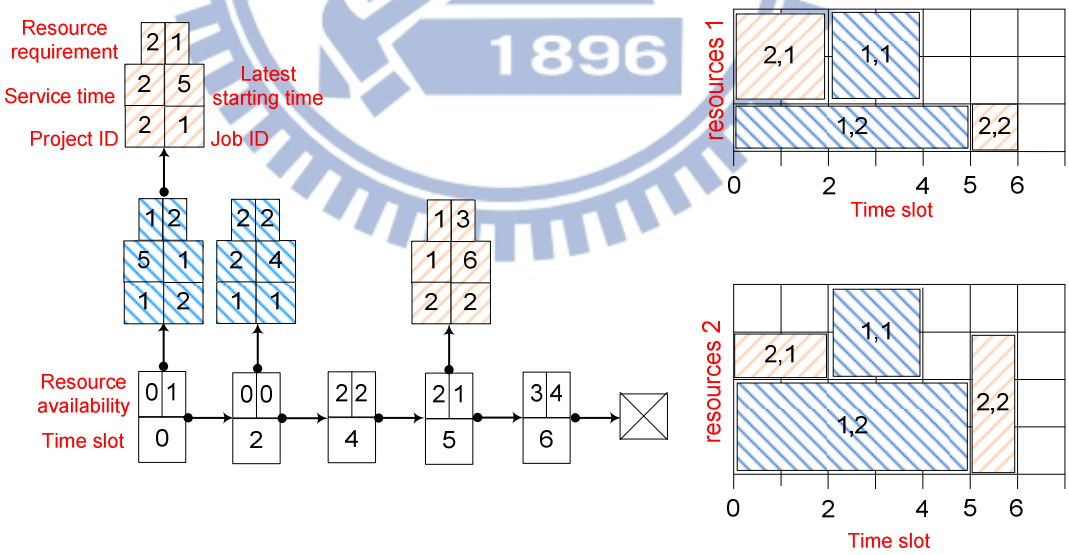


Figure 7: Data structure used for resource reservation

Table 5: Data structure operations

Operations	Explanation
$earliestStartingTime(j_{u,v})$	Search the earliest possible starting time for $j_{u,v}$
$feasibleBackfillingTimes(j_{u,v})$	Find all the feasible backfilling times for $j_{u,v}$.
$addReservation(j_{u,v})$	Add a resource reservation for $j_{u,v}$ at $ts_{u,v}$
$deleteReservation(j_{u,v})$	Delete the existing reservation of waiting job $j_{u,v}$
$shiftReservations(j_{u,v}, PL)$	Check the resource availability and delay some reservations to accommodate $j_{u,v}$ if necessary, given that the number of delayed projects $\leq PL$.

Chapter 5: Experimental Evaluation

This chapter presents the experimental evaluation where the effectiveness of proposed algorithms is verified. We first introduce the simulation methodology and then present the experimental results and the analysis.

5.1. Simulation Methodology

In order to evaluate the proposed scheduling algorithms, we have developed a simulator for our scheduling problem. The simulator is implemented based on CSIM 20 for Java [10] which is a simulation package with a process-oriented discrete-event scheduling model. CSIM 20 has been widely used to simulate complex systems in academia as well as industry.

Table 6: Simulation parameters

	Parameters		Distribution	Random function parameters
Project	Inter-arrival time	ita	Exponential	$\overline{ita}$ can be adjusted
	Number of jobs	$\lvert J_u \rvert$	Normal	$\overline{\lvert J_u \rvert} = 5.0$; $\sigma_{I_u} = 2.0$
Job	Service time	$te_{u,v}$	Exponential	$\overline{te} = 500.0$
	Resource requirement	$q^i_{u,v}$	Exponential	$\overline{q^i_{u,v}} = 2.0$
Resource	Types of resources	N	Constant	5
	Capacity of type-i resources	M_i	Uniform	$Min_{M_i} = 20$ $Max_{M_i} = 40$

Table 6 summarizes our simulation parameters which are randomly generated according to some types of distributions. One should notice that the values of two parameters, $\lvert J_u \rvert$ and $q^i_{u,v}$, are integer parts of the floating-point value generated by random functions. In our experiments, the inter-arrival time ita between any two successive project arrivals is exponentially distributed with an adjustable mean value in order to control project arrivals. By doing so, we can observe the performance of the proposed algorithms under different system loads. All the simulation results shown here are obtained by averaging the results of 5 simulation runs with different seeds for random number generation. Each simulation run is terminated upon the successful completion of 1000 projects.

The overall performance of the proposed scheduling algorithms could be evaluated by two major metrics: mean project turn-around time and average resource utilization. The former is used to measure the performance from the customer's point

of view, while the latter is the most common system-centric metric. However, it is shown that the average resource utilization does not change notably under different scheduling policies. Thus in the result analysis section, we do not present the results in terms of this metric.

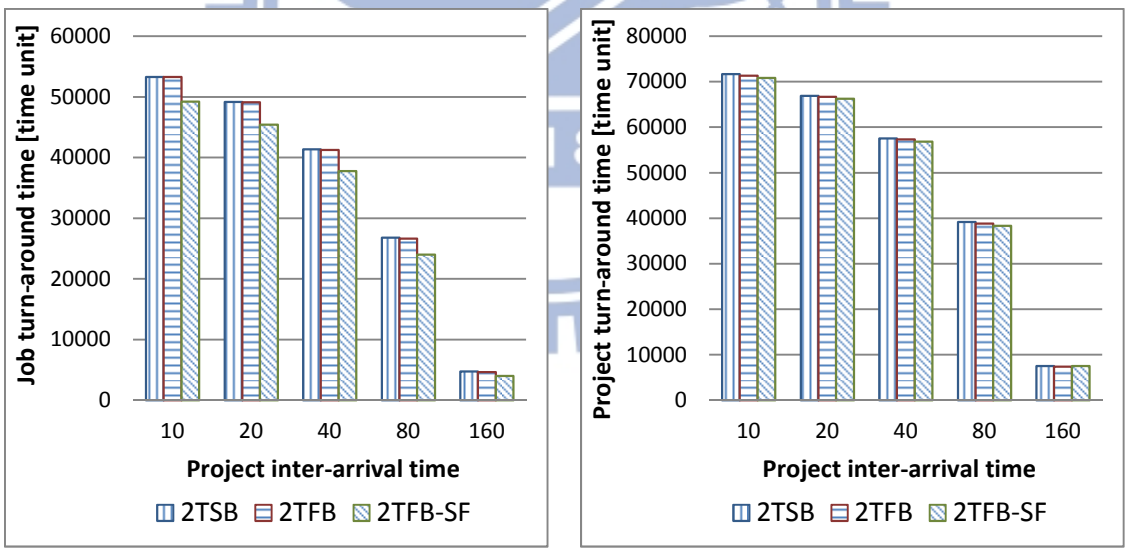
5.2. Result Analysis

Job Scheduling vs. Project Scheduling

In this experiment, we compare the performance of three algorithms: Two-tier Strict Backfilling (2TSB), Two-tier Flexible Backfilling with SF = 0 (2TFB), and Two-tier Flexible Backfilling with SF > 0 (2TFB-SF). For 2TFB-SF, we use the parameters $SF = 0.5$ and $PL = \infty$. 2TSB algorithm is used as the baseline for the performance comparison purposes. In addition to the mean project turn-around time $\overline{PTA}$, the mean job turn-around time $\overline{JTA}$, which is defined as

$$\overline{JTA} = \frac{1}{|P|}\sum_{u=1}^{|P|}\frac{1}{|J_u|}\sum_{v=1}^{|J_u|}(tf_{u,v} - ta_u) \qquad (8)$$

is another performance metric to investigate.



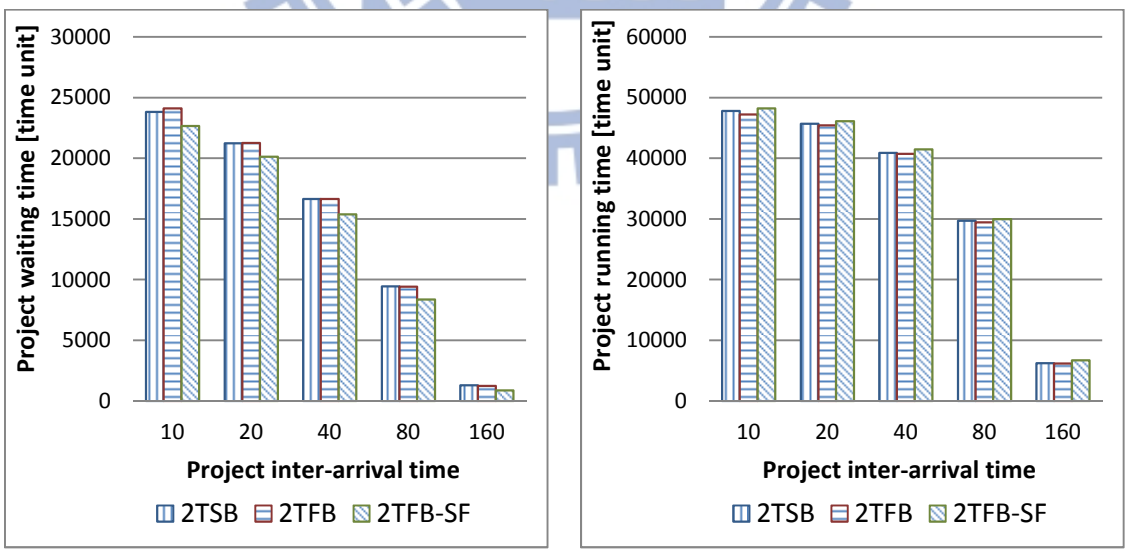
(a) **(b)**

Figure 8: Job scheduling vs. Project scheduling

As shown in Figure 8(a), it is not surprising that the mean job turn-around time is reduced considerably by 2TFB-SF algorithm because the effectiveness of the concept of slack factor has already been confirmed in several existing one-tier scheduling works [20, 21, 22]. Furthermore, the reduction in the mean job turn-around time greatly depends on the system load. For example, for the case where the mean

project inter-arrival time is set to 10, the performance difference between 2TSB and 2TFB-SF is about 4000 time units, which means a 7.5% improvement in the mean job turn-around time. On the other hand, in the case of the mean project inter-arrival time being set to 160, the difference is merely 700 time units but an improvement of 15.5%. Figure 8(a) also demonstrates that 2TSB and 2TFB have almost identical performance for all the values of the mean project inter-arrival time used in this experiment. This can be explained by the fact that the opportunities of carrying out the flexible backfilling mechanism are *rare* in 2TFB because of $SF = 0$.

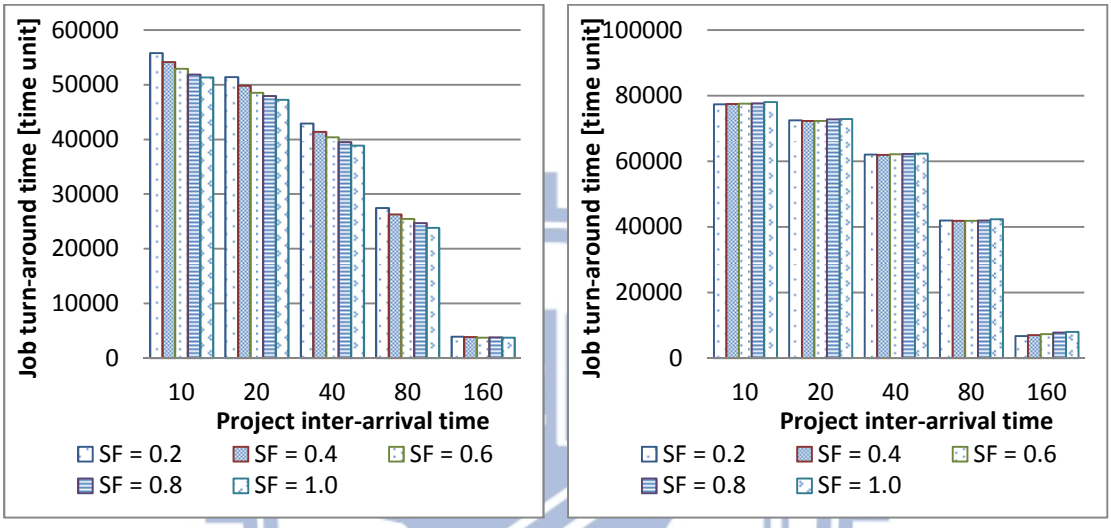
Figure 8(b) clearly indicates that the performance of 2TSB, 2TFB, and 2TFB-SF in terms of the mean project turn-around time is roughly the same. This observed phenomenon can be explained by Figure 9 where 2TFB and 2TFB-SF can reduce neither the mean project waiting time nor the mean project running time. Take 2TFB-SF for example; on the average, the mean project waiting time is decreased by 5% to 33% when the mean project inter-arrival time is changed from 10 to 160, but meanwhile, the mean project running time is increased by 0.7% to 7%. These results indicate that 2TFB-SF can decrease the waiting time of the *first* job of a project but lead to an increase in the waiting time of other *remaining* jobs of the project. Overall, adopting 2TFB-SF does not lead to a significant improvement on the mean project turn-around time.



(a) **(b)**

Figure 9: Mean project waiting time vs. Mean project running time

In order to understand more about the relationship between two metrics $\overline{PTA}$ and $\overline{JTA}$, we conduct another experiment whose results are shown in Figure 10. In this experiment, we measure and observe the performance of 2TFB-SF while varying the *SF* parameter. The experiment results show that using *larger* slack factors improves the mean job turn-around time significantly. However, this causes a modest increase in the mean project turn-around time. Based on our observation, the decrease of $\overline{JTA}$ does *not* lead to the decrease of $\overline{PTA}$.



(a) **(b)**

Figure 10: The impact of slack factor SF on 2TFB-SF

To sum up, although 2TFB and 2TFB-SF can reduce the mean job turn-around time notably in comparison with 2TSB, the improvement on the mean project turn-around time is negligible. One can suggest that 2TSB might be a good choice for two-tier cloud scheduling since it achieves the same performance as 2TFB and 2TFB-SF in terms of the mean project turn-around time, but its complexity is comparatively light weight.

The Impact of Priority Scheduling

In order to test how well Two-tier Priority Backfilling algorithm (2TPB) schedules high-priority projects, we devise the following two scenarios for the experiments. In the first scenario, all the submitted projects are scheduled by 2TSB. On the other hand, 2TPB is used to schedule projects in the second scenario. High-priority projects are given the priority value pi_u = 1.0, while the rest low-priority projects are given pi_u = 0.0. In both of these two scenarios, the probability that a submitted project is high-priority is 1/5. Since the performance of 2TPB can be

influenced by the slack factor SF and the preemption limit PL, we conduct the following two experiments to observe the effect of these parameters.

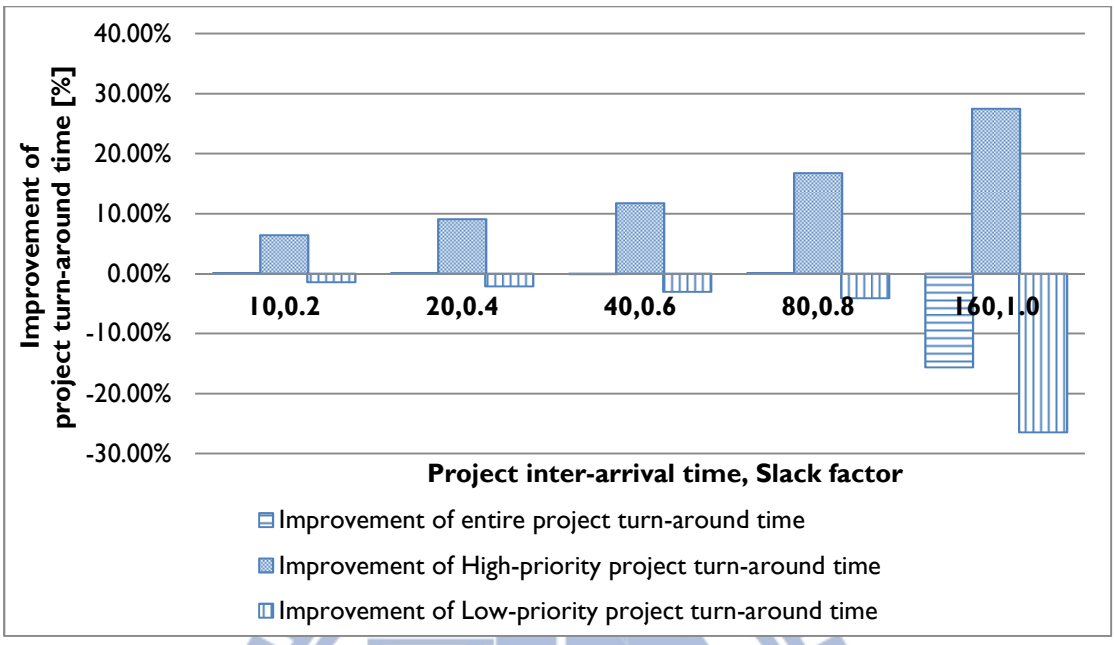


Figure 11: Improvement on the mean project turn-around time by 2TPB algorithm with differential values of slack factor SF

In the first experiment, we study the performance of 2TPB with $PL = \infty$ by observing the mean project inter-arrival time $\overline{ita}$ and slack factor SF. Figure 11 shows the improvement on the mean turn-around time of all the high-priority and low-priority projects in comparison with the 2TSB's performance. As expected, 2TPB decreases the mean turn-around time of the high-priority projects by 6% to 27% but increases the turn-around time of others by 1% to 26% when the value of $(\overline{ita}, SF)$ is increased from (10,0.2) to (160,1.0). Surprisingly, the mean turn-around time of all the projects remains almost unchanged except for the last case where the value of $(\overline{ita}, SF)$ is (160, 1.0). For the last case, the mean turn-around time is increased by 16%.

Figure 12 presents the results of the second experiment in which the performance of 2TPB with $SF = 0.5$ is measured at different values of the mean project inter-arrival time $\overline{ita}$ and preemption limit PL. The results are similar to those in the first experiment. The mean turn-around time of all the projects stays stable except for the case with the lowest loading of project arrivals. As the value of $(\overline{ita}, PL)$ increases from (10,1) to (160,∞), the mean turn-around of the high-priority projects is

reduced remarkably from 1.5% to 20%. On the other hand, this also leads to an increase around 0.01% to 10.7% in the mean turn-around time of the low-priority projects.

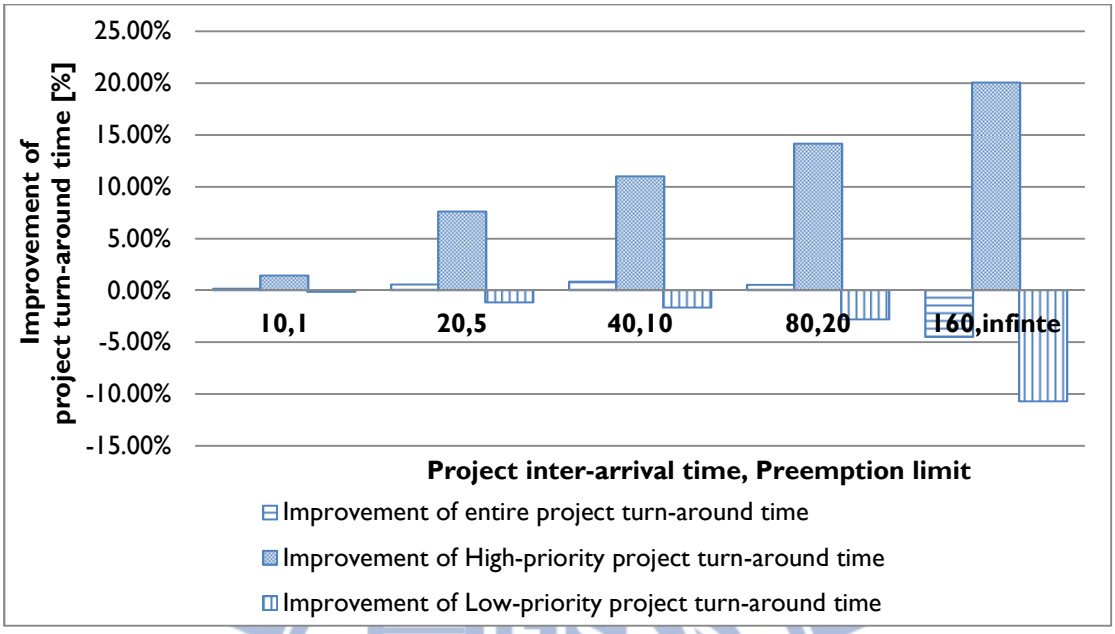


Figure 12: Improvement of project turn-around time of 2TPB with differential limits on the number of allowable delayed projects PL

The above experimental results indicate that 2TPB works well with priority scheduling where some projects are preferred over the others. Besides, 2TPB does not lead to a general degradation in system service in most cases. Since two system parameters SF and PL have a strong impact on the mean turn-around time of both types of projects, the system behavior can be controlled by adjusting these parameters.

Chapter 6: Conclusions and Future Work

In this work, we study a two-tier scheduling problem which is present in the cloud computing environments. This scheduling problem differs from the traditional one-tier scheduling problems since a submitted project consists of multiple jobs each requiring several resources for its processing. In order to reduce cloud service's turn-around time and support priority scheduling, we have developed a set of scheduling algorithms of different attributes. All the algorithms are derived from conservative backfilling algorithm, but enhanced with the concept of project's slack which is calculated by multiplying project turn-around time with a system parameter *slack factor*. Besides, another system parameter *preemption limit* is also proposed to control the behavior of the cloud scheduler.

The algorithms developed in this study have been experimentally evaluated under different system loads by computer simulation. The experimental results indicate that Two-tier Flexible Backfilling with $SF > 0$ (2TFB-SF) can reduce the job turn-around time by 7.5% to 15.5% and achieve almost the same performance in terms of the mean project turn-around time metric as Two-tier Strict Backfilling (2TSB) when the mean project inter-arrival time is changed from 10.0 to 160.0. Based on these results, we also reach an interesting conclusion that the decrease in the mean job turn-around time does not always lead to a decrease in the mean project turn-around time.

The experimental results also indicate that Two-tier Priority Backfilling (2TPB) can efficiently reduce the mean turn-around time of high-priority projects, but does not lead to an increase in the mean turn-around of all the projects in most cases. Furthermore, the behavior of the algorithm can be easily controlled by tuning two system parameters: *slack factor* SF and *preemption limit* PL, whose impact is analyzed in this work as well. More specifically, the mean turn-around time of high-priority projects is decreased from 6% to 27% when the value of ($\overline{ita}$,SF) is increased from (10,0.2) to (160,1.0). As the value of ($\overline{ita}$,PL) is relaxed from (10,1) to (160,∞), the mean turn-around time of high-priority projects is reduced from 1.5% to 20%.

Our proposed algorithms satisfy one fundamental requirement of the two-tier scheduling problem that a project should be granted a guaranteed departure time at the project's arrival time. However, doing so might decrease the jobs' backfilling opportunities since we must reserve resources for all the waiting jobs that have been

granted. Hence, the project turn-around time cannot be reduced considerably by our proposed algorithms. For future work, we plan to consider a *less* conservative backfilling approach in which only the jobs belonging to the first project in the waiting queue can receive resource reservations. Obviously, this less conservative approach will degrade the predictability of two-tier backfilling algorithms, but it may bring a more considerable reduction in cloud service's turn-around time.

Furthermore, an optimal algorithm for the off-line version of the scheduling problem, in which all projects' characteristics are known beforehand, will be studied in our future work. Other scheduling objectives, i.e. project success ratio, cloud provider revenue, are considered as well.

References

[1] L. Ai, M. Tang, C. Fidge, “Resource allocation and scheduling of multiple composite web services in cloud computing using Cooperative Coevolution,” in *International Conference on Neural Information Processing (ICONIP 2011)*, Shanghai, China, pp.13-17, Sep. 2011.

[2] “Google Drive”, available at: https://drive.google.com

[3] A.P.A. Vestjens, “On-line Machine Scheduling,” Ph.D. Thesis, Department of Mathematics and Computing Science, Technische Universiteit Eindhoven, Eindhoven, The Netherlands, 1997.

[4] A. Benoit, L. Marchal, J.-F. Pineau, Y. Robert, F. Vivien, "Scheduling Concurrent Bag-of-Tasks Applications on Heterogeneous Platforms," in *IEEE Transactions on Computers*, vol.59, no.2, pp.202–217, Feb. 2010.

[5] C. Anglano, M. Canonico, "Scheduling algorithms for multiple Bag-of-Task applications on Desktop Grids: A knowledge-free approach," in *IEEE International Symposium on Parallel and Distributed Processing* (IPDPS 2008), pp.14–18, April 2008.

[6] K. Gopalan, T. Chiueh, “Multi-resource allocation and scheduling for periodic soft real-time applications,” in *Proceedings of Multimedia Computing and Networking*, pp.34–45, Jan. 2002.

[7] M. Holenderski, R.J. Bril, J. Lukkien, “Parallel-Task Scheduling on Multiple Resources,” in *Proceedings of the 24th Euromicro Conference on Real-Time Systems* (ECRTS '12), pp.233–244. 2012.

[8] A.W. Mu'alem, D.G. Feitelson, "Utilization, predictability, workloads, and user runtime estimates in scheduling the IBM SP2 with backfilling," in *Parallel and Distributed Systems, IEEE Transactions on Parallel and Distributed Systems*, vol.12, no.6, pp.529–543, Jun 2001.

[9] D.G. Feitelson, "Experimental analysis of the root causes of performance evaluation results: a backfilling case study," in *IEEE Transactions on Parallel and Distributed Systems*, vol.16, no.2, pp.175–182, Feb 2005.

[10] “CSIM 20”, available at: http://www.mesquite.com/products/csim20.htm

[11] D.D. Sleator, R.E. Tarjan, “Amortized efficiency of list update and paging rules,” in *Communications of the ACM 28*, pp.202–208, 1985.

[12] J. Tan, "Semi-online scheduling problems on m parallel identical machines," in *Information Technology and Artificial Intelligence Conference (ITAIC), 2011 6th IEEE Joint International* , vol.1, pp.289–291, 20–22 Aug 2011.

[13] J.A. Hoogeveen, A.P.A. Vestjens, "Optimal on-line algorithms for single-machine scheduling," in *Proceedings Fifth International Conference on Integer Programming and Combinatorial Optimization*, Vancouver, British Columbia, Canada, June 3–5, 1996, vol.1084 of LNCS, pp. 404–414, Springer- Berlin, 1996.

[14] X. Lu, R.A. Sitters, L. Stougie, "A class of on-line scheduling algorithms to minimize total completion time," in *Operations Research Letters*, vol.31, no.3, pp 232–236, May 2003.

[15] D.G. Feitelson, L. Randolph, U. Schwiegelshohn, "Parallel job scheduling – a status report," in *Proceedings Tenth Workshop on Job Scheduling Strategies for Parallel Processing*, vol.3277 in LNCS, pp.1–16, 2004.

[16] D. Lifka, "The ANL/IBM SP scheduling system," in *Job Scheduling Strategies for Parallel Processing*, D. G. Feitelson and L. Rudolph (eds.), vol. 949 of LNCS, pp. 295–303, Springer-Verlag, 1995.

[17] S. Srinivasan, R. Kettimuthu, V. Subramani, P. Sadayappan, "*Selective reservation strategies for backfill job scheduling*," in *Job Scheduling Strategies for Parallel Processing*, D. G. Feitelson and L. Rudolph (eds.), vol. 2537 of LNCS, pp 55–71. Springer Verlag, 2002.

[18] S-H. Chiang, A. Arpaci-Dusseau, M. K. Vernon, "*The impact of more accurate requested runtimes on production job scheduling performance*," in *Job Scheduling Strategies for Parallel Processing*, D. G. Feitelson, L. Rudolph, and U. Schwiegelshohn (eds.), vol.2537 of LNCS, pp. 103–127, Springer-Verlag, 2002.

[19] J. P. Jones, B. Nitzberg, "Scheduling for Parallel Supercomputing: A Historical Perspective of Achievable Utilization," in *Job Scheduling Strategies for Parallel Processing*, vol.1659 of LNCS, pp. 1–16, Springer-Verlag, 1999.

[20] D. Talby, D.G. Feitelson, "Supporting priorities and improving utilization of the IBM SP scheduler using slack-based backfilling," in *13th International and 10th Symposium on Parallel and Distributed Processing*, pp.513-517, 12-16 Apr 1999.

[21] Jr. W. A. Ward, C. L. Mahood, J. E. West, "Scheduling Jobs on Parallel Systems Using a Relaxed Backfill Strategy," in *Job Scheduling Strategies for Parallel Processing*, vol. 2537 of LNCS, pp. 88-102, Springer-Verlag, 2002.

[22] B. Li, Y. Li, M. He, H. Wu, J. Yang, "Scheduling of a Relaxed Backfill Strategy with Multiple Reservations," in *International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT 2010)*, pp.311-316, 8-11 Dec 2010.

[23] B. G. Lawson, E. Smirni, "Multiple-Queue Backfilling Scheduling with Priorities and Reservations for Parallel Systems," in *Job Scheduling Strategies for Parallel Processing*, vol. 2537 of LNCS, pp. 72–87, Springer-Verlag, 2002.

[24] E. Shmueli, D.G. Feitelson, "Backfilling with lookahead to optimize the performance of parallel job scheduling," in *Job Scheduling Strategies for Parallel Processing*, D. G. Feitelson, L. Rudolph, and U. Schwiegelshohn (eds.), vol. 2862 of LNCS, pp. 228 – 251, Springer-Verlag, 2003.

[25] A. Sulistio, U. Cibej, S. Prasad, R. Buyya, "GarQ: An Efficient Scheduling Data Structure for Advance Reservations of Grid Resources," in *International Journal of Parallel, Emergent and Distributed Systems (IJPEDS)*, Taylor & Francis Publication, UK, 4 April 2008.

[26] A. Brodnik, A. Nilsson, "A static data structure for discrete advance bandwidth reservations on the internet," in *Proceedings of Swedish National Computer Networking Workshop (SNCNW)*, Stockholm, Sweden, September 2003.

[27] T. Wang, J. Chen, "Bandwidth tree – a data structure for routing in networks with advanced reservations," in *Proceedings of the 21st Intl. Performance, Computing, and Communications Conference (IPCCC)*, pp. 37–44, Phoenix, USA, 2002.

[28] R. Brown, "Calendar queues: A fast O(1) priority queue implementation for the simulation event set problem," in *Communications of the ACM 31(10)*, pp.1220–1227,1988.

[29] Q. Xiong, C. Wu, J. Xing, L. Wu, H. Zhang, "A Linked-List Data Structure for Advance Reservation Admission Control," in *ICCNMC 2005*, LNCS 3619, pp. 901 – 910, Springer-Verlag Berlin Heidelberg, 2005.