Journal of Network and Computer Applications 52 (2015) 26–36

Contents lists available at ScienceDirect

Journal of Network and Computer Applications

journal homepage: www.elsevier.com/locate/jnca

Two-tier project and job scheduling for SaaS cloud service providers

Ying-Dar Lin [a], Minh-Tuan Thai [a], Chih-Chiang Wang [b,*], Yuan-Cheng Lai [c]

[a] Department of Computer Science, National Chiao Tung University, No. 1001, Ta Hsueh Road, Hsinchu 300, Taiwan
[b] Department of Computer Science, National Kaohsiung University of Applied Sciences, 415 Chien Kung Road, Kaohsiung 807, Taiwan
[c] Department of Information Management, National Taiwan University of Science and Technology, No. 43, Keelung Road, Taipei 106, Taiwan

ARTICLE INFO

Article history:
Received 22 January 2014
Received in revised form
1 December 2014
Accepted 5 February 2015
Available online 14 March 2015

Keywords:
Two-tier scheduling
SaaS scheduling
Backfilling
Slack factor

ABSTRACT

This study addresses a two-tier job scheduling problem for SaaS cloud service providers which rely on resources leased from IaaS cloud providers to achieve elasticity to computational power. In our model, a project represents a user request which consists of multiple jobs; the SaaS is obligated to complete projects using multiple resources leased from IaaS or PaaS providers. The goals are to reduce the project turn-around time and to support priority scheduling by employing suitable scheduling algorithms. We propose a set of two-tier backfilling algorithms which extend the well-known conservative backfilling algorithm with project slack-time and priority concepts. Among the proposed algorithms, Two-Tier Strict Backfilling (2TSB) does not allow preemption in job waiting queues. On the other hand, preemption is allowable in Two-tier Flexible Backfilling 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 with 2TSB, 2TPB could reduce the mean turn-around time of high-priority projects by more than 25%.

1. Introduction

Cloud computing is an Internet-based computing paradigm whereby computational resources are delivered to users on demand over the Internet as a public utility (Ai et al., 2011). Customers can make use of the cloud as a software (SaaS), platform (PaaS), or infrastructure (IaaS) provider on a pay-per-use basis, hence avoiding maintenance costs while enjoying elasticity to the available computational power. Google cloud platform is a good example of cloud computing services in which all the infrastructure, software, and storage are hosted remotely and users only need a Web browser to access the service.

Here we are particularly interested in the SaaS cloud provider which relies on resources leased from IaaS or PaaS cloud providers to achieve elasticity to its computational power. We consider an SaaS model as described below. Each customer submits his work as a project to be executed by the system. Once the SaaS accepts a project, it is obligated to complete a set of jobs belonging to the project by using multiple computational resources leased from IaaS or PaaS providers. We assume that each job has its own estimated service time determined during the pre-processing stage. To start processing a job, the SaaS must have a specific set of resources, such as server CPU cycles, disk storage, or network bandwidth, allocated to the job for its processing. Furthermore, the resource requirement of a job may involve multiple resource types simultaneously. We assume that at the moment when a project is submitted to the SaaS, the project's workload characteristics become available to the SaaS such that the SaaS must make scheduling decisions immediately and then inform users of when the project will be finished. As projects arrive to the system one by one over time, the SaaS must always make scheduling decisions on the fly without knowledge of any future project arrivals. This practice is called on-line scheduling in the literature (Vestjens, 1997).

The objective of this work is to provide an efficient SaaS service by employing suitable scheduling algorithms and resource allocation strategies. In specifics, we want to improve the mean project turn-around time, the mean value of the overall time from submission to completion of the project, while achieving high utilization of cloud resources. Even though the concept of two-tier scheduling has been addressed by several studies (Benoit et al., 2010; Anglano and Canonico, 2008; Gopalan and Chiueh, 2002; Holenderski et al., 2012) in recent years, our solution is different from those already existing because we deal with two-tier scheduling, job priorities, and multi-type resource allocation at the same time. Our solution

* Corresponding author. Tel.: +886 9 0919 919624.
E-mail addresses: ydlin@cs.nctu.edu.tw (Y.-D. Lin), ccwang@kuas.edu.tw (C.-C. Wang), laiyc@cs.ntust.edu.tw (Y.-C. Lai).

http://dx.doi.org/10.1016/j.jnca.2015.02.008
1084-8045/

extends the conservative backfilling (Mu'alem and Feitelson, 2001; Feitelson, 2005) 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. The idea of slack factor is not new in the area of scheduling research, but up to now, we are not aware of any existing work that have applied the slack-factor concept to the two-tier scheduling problem the way our approach does. So far, we have developed a set of algorithms by taking the slack time and project priority 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.

We like to emphasize that even though fault tolerance is an important issue in cloud computing as system reliability is perceived as part of cloud service-level agreements, due to the limited scope of this work, we focus on improving the mean project turn-around time and simply assume that there will be an underlying fault-tolerance mechanism taking care of reliability-related issues. Generally speaking, high-performance computing systems usually adopt *checkpointing* (with rollback-recovery) (Agarwal et al., 2004; Gelenbe, 1979; Ozaki et al., 2004; Oliner et al., 2006; Katsaros et al., 2007) and *job replication* (Silva et al., 2003; Li and Mascagni, 2003; Hou and Shin, 1994) as fault-tolerance mechanisms. By checkpointing, resources from time to time store the states of their running jobs in a stable storage so that once the failure occurs, the scheduler can place the failed job on another resource and resume the computation from the checkpoint state on recovery. On the other hand, replication aims to raise the job success rate by replicating each job and executing job replicas on different resources in parallel.

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

2. Related work

This section first gives a brief overview about on-line scheduling and then a survey on existing two-tier scheduling models. After that, a variety of backfilling algorithms which have been widely studied in the literature are discussed in depth.

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. In contrast to off-line algorithms, an on-line scheduling algorithm is intended for a realistic scenario where the scheduler does not have the access to the whole input instances (Vestjens, 1997). In other words, at the moment when a job is presented to the scheduler, on-line scheduling decisions must always be made without knowledge of any future job arrivals. Furthermore, the scheduling decision is irreversible once it is made even if we find other obviously better schedules afterward.

In order to evaluate performance of an on-line algorithm, Sleator and Tarjan (1985) 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. An on-line algorithm is ρ-competitive if its objective value is no more than ρ times of the optimal off-line value for any arbitrary sequence of job input instances. Since the assumptions of pure on-line scheduling make it impossible to find the optimal solution, some concepts have been developed to handle variants of this scheduling problem. For example, semi-online scheduling (Tan, 2011) assumes that partial information about the scheduling problem is available to the scheduler before the scheduler constructs a schedule. The authors in Hoogeveen and Vestjens (1996) and Lu et al. (2003) presented the concept of delaying the scheduling time of a job for a period of time and later applying scheduling rules to the accumulated jobs in the queue.

2.2. Two-tier scheduling

There are several variants of two-tier scheduling models that have been proposed during the last few years. The Bag-of-Tasks (BoT) application (Benoit et al., 2010; Anglano and Canonico, 2008) model, whose tasks are identical and independent, is often considered as a suitable model for heterogeneous clusters and desktop grid environment. The objective of BoT application model is often 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 Benoit et al. (2010) introduce an optimal off-line algorithm and a heuristic on-line algorithm to minimize the maximum stretch of BoT applications. In Anglano and Canonico (2008), a set of task selection polices is proposed in order to minimize the turn-around time of BoT applications.

Gopalan and Chiueh (2002) 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 application's lifetime and is subject to a precedence constraint among the tasks. The two-tier scheduling model introduced in Holenderski et al. (2012) allows the processing of a job to be preempted by another job before its completion.

2.3. Backfilling algorithm

The backfilling algorithm, which was first introduced by Lifka (1995), aims to balance between the goals of utilizing system resources and maintaining the FCFS (First Come, First Served) order of job execution (Feitelson et al., 2004). The spirit behind the backfilling 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 insufficiency of available resources. This principle helps exploit idle resources by backfilling with suitable jobs, thereby increasing system utilization and throughput. 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 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 the waiting jobs in the queue.

There are several variants of backfilling algorithms. The most popular one is aggressive backfilling (Lifka, 1995; Feitelson et al., 2004), in which only the first job in the queue can receive a resource reservation. If an arrived job is the first job in the queue and cannot be processed immediately, the scheduler calculates the earliest possible starting time for this job using its resource requirement and service time, then makes a resource reservation for this job at its pre-calculated starting time. Other jobs are allowed to backfill only if they do not violate the first job's reservation. The core problem of

aggressive backfilling is its unpredictability since except for the first one in the queue, the waiting jobs do not have a guaranteed starting time. In contrast to aggressive backfilling, conservative backfilling (Mu'alem and Feitelson, 2001; Feitelson, 2005) makes reservation for every queued job which cannot be executed immediately. It means that a job can be backfilled on condition that it does not delay any queued job. Clearly, conservative backfilling can guarantee every job's starting time, but its performance tends to be inferior to that of aggressive backfilling. There are some variants of backfilling algorithms making reservation for the waiting jobs adaptively. In Srinivasan et al. (2002), the waiting jobs are not given reservations until their expected turn-around time exceeds a certain threshold. Chiang et al. (2002) suggested that making reservations for the first four waiting jobs in the queue is a good compromise between aggressive backfilling and conservative backfilling. Ward et al. (2002) introduced multiple-queue backfilling which divides the system resources into multiple disjoint partitions. This approach aims at reducing fragmentation of system resources and hence reducing the likelihood that a short job is queued behind a long job. Backfilling with lookahead (Li et al., 2010) algorithm makes scheduling decisions by considering a set of jobs at once. It looks ahead into the job queue and tries to find a packing of jobs which maximizes the scheduler's objective.

2.4. Slack-based backfilling

To make backfilling scheduling more flexible and also to increase resource utilization, slack-based backfilling algorithms (Lawson and Smirni, 2002; Shmueli and Feitelson, 2003; Jones and Nitzberg, 1999; Talby and Feitelson, 1999) 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 condition that it will not delay the existing reservations by more than a specific threshold. In these algorithms, the system's slack factor is used to control for how long jobs will have to wait before the start of its execution.

The idea of slack factor has already been introduced to real time scheduling and grid scheduling environments, and has been confirmed to be effective (Lifka, 1995). Dynamic backfilling allows the scheduler to overrule a previous reservation by a slight delay if doing so can improve system utilization considerably (Lawson and Smirni, 2002). In order to enhance backfilling and support priority scheduling, Shmueli and Feitelson (2003) 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 calculated based on its jobs' delay and resource requirements. Jones and Nitzberg (1999) suggested a relaxed backfilling strategy in which a backfill candidate is selected from the job waiting queue based on the job's waiting time, its estimated service time and resource requirement. Li et al. (2010) introduced a different slack-based backfilling which supports more than one reservation.

3. Model description

This section first describes the two-tier scheduling model of our work. Figure 1 provides a schematic illustration of our scheduling model, where the notations for this model are shown in Table 1. In our system model, the cloud has N types of resources. Each type of resources has limited capacity M_i which 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 $R=\{M_i \mid 1 \le i \le N\}$ denote a set of N types of cloud resources of the system.

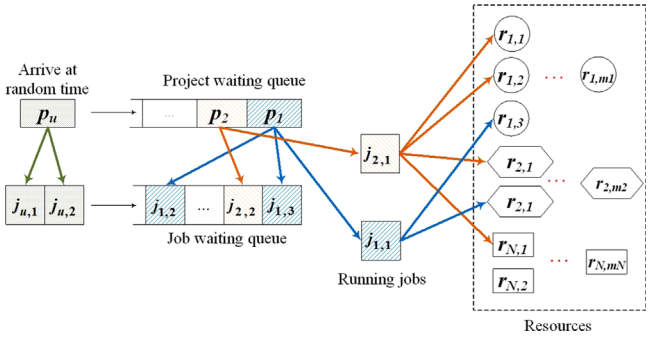

Fig. 1. Two-tier scheduling model.

A project with multiple jobs represents a request submitted by the SaaS user. Let $P=\{p_u \mid 1 \le u \le |P|\}$ denote a set of projects of the SaaS 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 $J_u=\{j_{u,v} \mid 1 \le v \le |J_u|\}$ 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 $E_{u,v}=\{q^i_{u,v} \mid 1 \le i \le N\}$ where $q^i_{u,v}$ is the number of type-i resources required by $j_{u,v}$, $0 \le q^i_{u,v} \le M_i$.

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

$$tc_u = \min(J_u \cdot ts) \quad \text{where } J_u \cdot ts = \{ts_{u,v} \mid 1 \le v \le |J_u|\} \tag{1}$$

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

$$td_u = \max(J_u \cdot tf) \quad \text{where } J_u \cdot tf = \{tf_{u,v} \mid 1 \le v \le |J_u|\} \tag{2}$$

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

$$tw_u = tc_u - ta_u \tag{3}$$

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

$$tr_u = td_u - tc_u \tag{4}$$

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

$$tn_u = td_u - ta_u \tag{5}$$

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

Table 1
Notations used in the two-tier scheduling model.

Symbol	Meaning
$R=\{M_i \mid 1\le i\le N\}$	A set of N types of cloud resources where M_i is the capacity of type-i resources
$P=\{p_u \mid 1\le u\le \lvert P\rvert\}$	A set of projects submitted to the SaaS where p_u is the uth project and $\lvert P\rvert$ is the number of projects
$J_u=\{j_{u,v} \mid 1\le v\le \lvert J_u\rvert\}$	The set of jobs which are required to be processed for the uth project where $j_{u,v}$ is the vth job of the uth project and $\lvert J_u\rvert$ is the number of jobs belonging to p_u
ta_u	The arrival time of the uth project
tc_u	The starting time of the uth project: $\mathrm{tc}_u=\min(J_u\cdot \mathrm{ts})$ where $J_u\cdot \mathrm{ts}=\{\mathrm{ts}_{u,v} \mid 1\le v\le \lvert J_u\rvert\}$
td_u	The departure time of the uth project: $\mathrm{td}_u=\max(J_u\cdot \mathrm{tf})$ where $J_u\cdot \mathrm{tf}=\{\mathrm{tf}_{u,v} \mid 1\le v\le \lvert J_u\rvert\}$
tw_u	The waiting time of the uth project: $\mathrm{tw}_u=\mathrm{tc}_u-\mathrm{ta}_u$
tr_u	The running time of the uth project: $\mathrm{tr}_u=\mathrm{td}_u-\mathrm{tc}_u$
tn_u	The turn-around time of the uth project: $\mathrm{tn}_u=\mathrm{td}_u-\mathrm{ta}_u$
$\overline{\mathrm{PTA}}$	The mean turn-around time of projects: $\overline{\mathrm{PTA}}=\frac{1}{\lvert P\rvert}\sum_{u=1}^{\lvert P\rvert}\mathrm{tn}_u$, where $\lvert P\rvert$ is the number of projects
$\mathrm{te}_{u,v}$	The service time of the vth job of the uth project
$\mathrm{ts}_{u,v}$	The starting time of the vth job of the uth project
$\mathrm{lts}_{u,v}$	The latest starting time of the vth job of the uth project
$\mathrm{tf}_{u,v}$	The finish time of the vth job of the uth project
$E_{u,v}=\{q^i_{u,v} \mid 1\le i\le N\}$	The resource requirement of the vth job of the uth project where $q^i_{u,v}$ is the number of type-i resource

A good practical example for our two-tier scheduling model is SaaS software services which use resources leased from IaaS or PaaS cloud providers to provide software services for customers across the Internet. In the SaaS system, a project represents a user request submitted by the SaaS customers. Once the SaaS accepts a project, it is obligated to complete a set of jobs specified in the project. The processing of a job requires a certain amount of cloud resources such as server CPU cycles, storage space and network bandwidth. Since the customers need to plan their work ahead of time, the departure time of a submitted project should be reported to the corresponding customer at the moment when the SaaS accepts the project.

Problem statement. Given a set of resources R and a set of projects P, each project $p_u\in P$ consists of a set of jobs J_u. Each job $j_{u,v}\in J_u$ requires a set of resources $E_{u,v}$ for its processing. The objective of this work is determining the starting time $\mathrm{ts}_{u,v}$ and the departure time td_u of each job $j_{u,v}$ of project p_u, $\forall p_u\in P$ such that the mean turn-around time $\overline{\mathrm{PTA}}$ is minimized while the service level agreement is abided. The formula

$$\overline{\mathrm{PTA}}=\frac{1}{\lvert P\rvert}\sum_{u=1}^{\lvert P\rvert}\mathrm{tn}_u \tag{6}$$

is used to calculate the mean turn-around time of projects.

Service level agreement (SLA): We assume that at the moment when a project is submitted to our proposed SaaS scheduler, the project's workload characteristics become available to the scheduler such that the scheduler must make scheduling decisions immediately and then inform the submitter of when the project is planned to be finished (i.e. the project departure time or the relaxed project departure time). As projects arrive to the system one by one over time, the scheduler must always make scheduling decisions on the fly without knowledge of any future project arrivals, and it must abide by the decisions of the (relaxed) project departure time which were reported earlier to the submitters.

4. Two-tier backfilling scheduling with slack factor and job priority

In this section, we describe our proposed scheduling policies for solving the two-tier scheduling problem defined in Section 3. We first give an overview on two-tier backfilling algorithm with a slack factor of project turn-around time. The details of our approach are then elaborated in the subsequent subsections. Finally, we introduce a data structure for resource reservation which is used to implement our proposed scheduling policies.

4.1. Overview

One of the fundamental requirements for our scheduling model is predictability. In other words, every project should be granted a guaranteed departure time at its arrival time. This requirement can be satisfied by conservative backfilling algorithm, which 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 scheduling algorithm more flexible and also to 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 (Lawson and Smirni, 2002; Shmueli and Feitelson, 2003; Jones and Nitzberg, 1999; Talby and Feitelson, 1999). 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 SF. The actual departure time of the project can be relaxed to a value in the time range $[\mathrm{td}_u, \mathrm{td}_u+\mathrm{tn}_u\cdot \mathrm{SF}]$. When the latest departure time of the project is determined, the scheduler can easily calculate the latest starting time $\mathrm{lts}_{u,v}$ of each job $j_{u,v}$ which belongs to J_u by

$$\mathrm{lts}_{u,v}=\mathrm{td}_u+\mathrm{tn}_u\cdot \mathrm{SF}-\mathrm{te}_{u,v} \tag{7}$$

Newly arrived jobs cannot delay job $j_{u,v}$ beyond its latest starting time $\mathrm{lts}_{u,v}$ which is set by the SaaS scheduler.

Furthermore, another system parameter PL is also introduced to control the number of preempted projects not to exceed PL. The implication of using parameter PL is that we can limit the number of projects whose departure time will be re-arranged. By doing so, the behavior of the SaaS 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

and priority. Hence, it is up to the SaaS administrator to select an appropriate scheduling policy.

4.2. Non-preemptive job/project scheduling

In this scheduling policy, job finish time $tf_{u,v}$ and project departure time td_u are determined when projects arrive to the system. New jobs can be backfilled only if they do not delay any existing reservation. For this policy, we have designed Two-tier Strict Backfilling (2TSB) algorithm whose pseudo-code is listed in Figure 2. 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 reservation, $j_{u,v}$ will be backfilled to start immediately (line 4). Otherwise, the earliest possible starting time of $j_{u,v}$ is determined by the operation *earliestStartingTime* (line 6). 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).

4.3. Non-preemptive project scheduling with preemptive job scheduling

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 the starting time of any queued job if and only if the job is not the last job of any waiting project (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 exceed $(td_u - te_{u,v})$. We devise Two-tier Flexible Backfilling (2TFB) algorithm to implement this policy. Figure 3 shows the pseudo-code of 2TFB algorithm, which are summarized as follows. The steps listed on lines 2–4 are similar to those of 2TSB algorithm because 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 $j_{u,v}$ based on the current scheduling plan by the operation *feasibleBackfillingTimes* (line 6). A feasible backfilling time $bt \in BT$ is the starting time of a gap in the job waiting queue at which $q^i_{u,v} \le a_i$, $\forall i : 1 \le i \le M$, 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. By 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, and that is the job's earliest possible starting time. After finishing the scheduling of all the jobs $j_{u,v} \in J_u$ and determining

Table 2
Summary of two-tier scheduling policies.

Algorithm	Policy	Objective	Job Finish Time	Project Departure Time
Two-tier Strict Backfilling	Non-preemptive: job/project scheduling	Minimize project turn-around time	Determined at project arrival time	Determined at project arrival time
Two-tier Flexible Backfilling (SF=0)	Non-preemptive: project scheduling Preemptive: job scheduling	Minimize project turn-around time	Flexible	Determined at project arrival time
Two-tier Flexible Backfilling (SF > 0.0)	Preemptive: job/project scheduling (Single type of projects)	Minimize project turn-around time	Flexible	Flexible
Two-tier Priority Backfilling	Preemptive: job/project scheduling (Two types of projects)	Minimize high-priority project turn-around time	High-priority: Determined at project arrival time Low-priority: Flexible	High-priority: Determined at project arrival time Low-priority: Flexible

Algorithm 1: *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 job j_{u,v}
7.          lts_{u,v} ← ts_{u,v}
8.          addReservation(j_{u,v})
9.        End-If
10.     End-For
11.   End
```

Fig. 2. Pseudo-code for Two-tier Strict Backfilling algorithm.

the turn-around time tn_u, 2TFB algorithm updates the latest starting time $lts_{u,v}$ for each job (lines 21–22).

4.4. Preemptive job/project scheduling

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 reservation up to the maximum number of allowable times. On the other hand, the latter po icy takes priority into account such that only some high-priority projects can preempt the low-priority ones.

4.4.1. Single type of projects

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) implements this scheduling policy. Besides, 2TFB-SF uses parameter $PL > 0$ to control the number of preempted projects.

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

The steps of 2TPB algorithm are briefed in Figure 4. If the newly arrived project p_u is high-priority, it is scheduled by 2TFB-SF (line 4) with slack factor SF recomputed with 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 (lines 8–10).

Algorithm 2: *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 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 resources 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 · SF) − te_{u,v}
22.     updateLatestStartingTime(j_{u,v})
23.   End-For
20.  End
```

Fig. 3. Pseudo-code for Two-tier Flexible Backfilling algorithm.

4.5. Implementation of two-tier backfilling

In order to implement the proposed scheduling 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 description of our proposed data structure is illustrated in Figure 5 while Table 3 shows the implemented operations on the data structure. The data structure is based on the linked-list data structure (Xiong et al., 2005) because of its simplicity and flexibility. 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 $j_{u,v}$ which is $lts_{u,v}$, and the resource requirement $E_{u,v}$.

5. Performance evaluation

This section presents the experimental evaluation where the effectiveness of proposed algorithms is verified. We first analyze the time complexity of our proposed scheduling algorithms and then present the simulation methodology, the experimental results and the analysis.

5.1. Complexity analysis

5.1.1. Algorithm 1: Two_Tier_StrictBackfilling (p_u)

The Two-tier Strict Backfilling algorithm (2TSB) is based on the key operation *earliestStartingTime*, which is basically a constrained sorting algorithm traversing through resource reservation records of N-type cloud resources. For each job $j_{u,v} \in p_u$, 2TSB has to traverse through N-type resource reservation records with sorting complexity bounded by $O(N \times \log(N))$, so 2TSB has a time complexity of $O(|J| \times N \times \log(N))$ per project, where $|J|$ denotes the mean number of jobs contained in a project.

Algorithm 3: *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 · SF) − te_{u,v}
10.       updateLatestStartingTime(j_{u,v})
11.     End-For
12.   End-If
11.  End
```

Fig. 4. Pseudo-code for Two-tier Priority Backfilling algorithm.

5.1.2. Algorithm 2: Two_Tier_FlexibleBackfilling (p_u, SF, PL)

Here we use the same analytic method from Shmueli and Feitelson (2003) to perform the time complexity analysis. For each job $j_{u,v} \in p_u$, Two-tier Flexible algorithm (2TFB) first calls the operation *feasibleBackfillingTimes* to go through all the jobs in the waiting queue and find the set of all feasible backfilling times BT. Then, 2TFB iterates through all the feasible backfilling times in BT and selects the one that complies with relaxed backfilling and also has the largest latest starting time. Apparently, 2TFB has a time complexity of $O(|J| \times N \times \log(N) \times |\mathrm{WQ}|^2)$ because the mean size of BT is bounded by the mean length of the waiting queue which is denoted by $|\mathrm{WQ}|$.

It is difficult to derive a close-form solution or a good approximation to $|\mathrm{WQ}|$ of our two-tier scheduling model because unlike traditional queuing models, our model does not have a specific number of servers. Nevertheless, if given that the averaged system capacity is μ jobs per unit time, the averaged project arrival rate leads to λ job arrivals per unit time and on average the system can process m jobs in parallel, then we can use the well-known $M/M/m$ or $M/G/m$ queuing model to find out an approximate solution to $|\mathrm{WQ}|$ (Kleinrock, 1975).

5.1.3. Algorithm 3: Two_Tier_PriorityBackfilling(p_u, SF, PL)

In this algorithm, if the newly arrived project p_u is high-priority, it is scheduled by 2TFB-SF while low-priority ones are scheduled by 2TSB. After that, the scheduler updates the latest starting time for each job. Therefore, Two-tier Priority Backfilling (STPB) algorithm has a time complexity of $O(|J| \times N \times \log(N) \times |\mathrm{WQ}|^2)$.

5.2. 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 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. The simulation time is set in unit of seconds.

Table 4 summarizes our simulation parameters which are randomly generated according to some well-known distributions. One should notice that the values of two parameters, $|J_u|$ 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

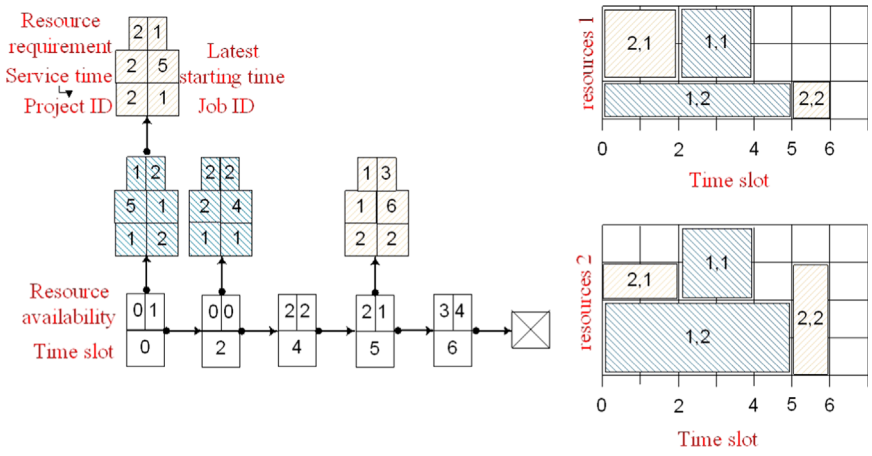

Fig. 5. Data structure used for resource reservation.

Table 3
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 $\mathrm{ts}_{u,v}$
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

Table 4
Simulation parameters.

Object type	Parameters	Distribution	Random function parameters
Project	Inter-arrival time (ita)	Exponential	$\overline{\mathrm{ita}}$ can be adjusted
Project	Number of jobs ($\lvert J_u \rvert$)	Normal	$\overline{\lvert J_u \rvert} = 5.0$; $\sigma_{J_u} = 2.0$
Job	Service time ($\mathrm{te}_{u,v}$)	Exponential	$\mathrm{te}_{u,v} = 500.0$
Job	Resource requirement ($q^i_{u,v}$)	Exponential	$q^i_{u,v} = 2.0$
Resource	Types of resources (N)	Constant	5
Resource	Capacity of type-i resources (M_i)	Uniform	$\min_{M_i} = 20.0$; $\max_{M_i} = 40.0$

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 five 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.

5.3. Result analysis

5.3.1. 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=∞. 2TSB algorithm is used as the baseline for the performance comparison purpose. In addition to the mean project turn-around time $\overline{\text{PTA}}$, the mean job turn-around time $\overline{\text{JTA}}$, which is defined as

$$\overline{\text{JTA}} = \frac{1}{|P|}\sum_{u=1}^{|P|}\frac{1}{|J_u|}\sum_{v=1}^{|J_u|}(\text{tf}_{u,v} - \text{ta}_u) \tag{8}$$

is another performance metric to investigate.

As shown in Figure 6(a), it is not surprising that the mean job turn-around time is reduced by 2TFB-SF algorithm because the effectiveness of the concept of slack factor has already been confirmed in several existing one-tier scheduling literatures (Talby and Feitelson, 1999; Ward et al., 2002; Li et al., 2010). 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 6(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 6(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 7 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 from 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 from 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.

In order to understand more about the relationship between two metrics $\overline{\text{PTA}}$ and $\overline{\text{JTA}}$, we conduct another experiment whose results are shown in Figure 8. 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 $\overline{\text{JTA}}$ does not lead to the decrease of $\overline{\text{PTA}}$.

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.

Figure 9 displays the results of five simulation runs of 2TSB, 2TFB, and 2TFB-SF experiments in terms of resource utilization rate. To test the robustness of our proposed algorithms with dynamic workload attributes, our simulation experiments generate resource requirements of the offered load randomly based on exponential distributions, meaning that the workload's demand on system resources is unevenly distributed. Hence, when some types of system resources are highly utilized or overloaded, they become the system bottleneck which stalls the job processing and causes other types of system resources under-utilized. For example, Resource Type 5 and Resource Type 3 are the bottleneck resources in Run 1 and Run 2, respectively. Figure 10 plots the simulation results in terms of average resource utilization rate. For the mean project inter-arrival time equal to 160, the average resource utilization rates of 2TSB, 2TFB, and 2TFB-SF fall at some points around 74%. For the mean project inter-arrival time equal to or above 80, the average resource utilization rates of 2TSB, 2TFB, and 2TFB-SF remain within a fixed range between 81.9% and 83.9%. In summary, all our policies contribute to a high system utilization rate.

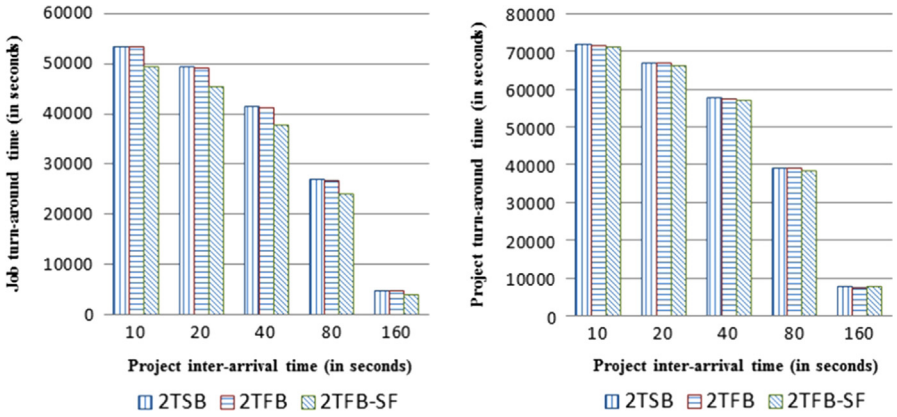

Fig. 6. Job scheduling vs. Project scheduling.

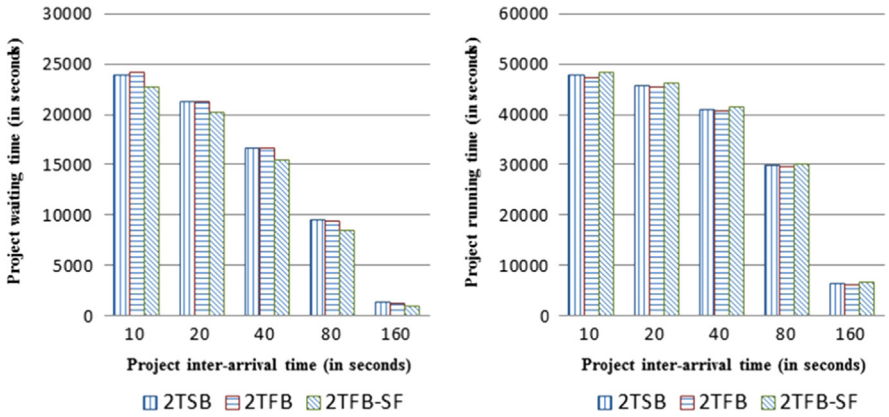

Fig. 7. Mean project waiting time vs. Mean project running time.

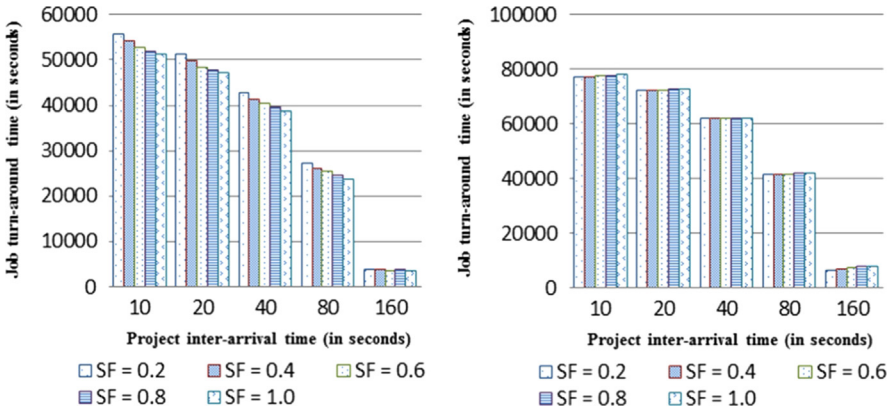

Fig. 8. The impact of slack factor SF on 2TFB-SF.

Inter-arrival Time		10			20			40			80			160		
		2TSB	2TFB	2TFB-SF	2TSB	2TFB	2TFB-SF	2TSB	2TFB	2TFB-SF	2TSB	2TFB	2TFB-SF	2TSB	2TFB	2TFB-SF
Run 1	Resource 1	0.856	0.856	0.893	0.855	0.856	0.882	0.855	0.856	0.891	0.854	0.854	0.87	0.777	0.777	0.779
	Resource 2	0.851	0.85	0.895	0.851	0.851	0.881	0.85	0.851	0.888	0.848	0.848	0.873	0.777	0.778	0.779
	Resource 3	0.552	0.553	0.582	0.553	0.552	0.575	0.553	0.553	0.573	0.551	0.551	0.564	0.503	0.504	0.507
	Resource 4	0.642	0.642	0.665	0.642	0.642	0.659	0.642	0.643	0.659	0.641	0.641	0.651	0.589	0.591	0.595
	Resource 5	0.992	0.993	0.99	0.992	0.992	0.992	0.992	0.992	0.987	0.989	0.99	0.988	0.975	0.974	0.969
Run 2	Resource 1	0.8	0.802	0.817	0.798	0.801	0.821	0.799	0.798	0.815	0.789	0.791	0.798	0.663	0.664	0.663
	Resource 2	0.834	0.837	0.867	0.834	0.836	0.865	0.832	0.838	0.862	0.825	0.826	0.85	0.708	0.709	0.709
	Resource 3	0.974	0.978	0.967	0.974	0.976	0.966	0.97	0.971	0.962	0.957	0.961	0.959	0.853	0.854	0.848
	Resource 4	0.932	0.939	0.952	0.933	0.936	0.942	0.927	0.934	0.947	0.918	0.919	0.937	0.795	0.796	0.795
	Resource 5	0.894	0.898	0.915	0.894	0.898	0.917	0.892	0.895	0.909	0.882	0.886	0.892	0.757	0.756	0.759
Run 3	Resource 1	0.985	0.987	0.983	0.984	0.987	0.981	0.982	0.984	0.981	0.976	0.979	0.979	0.955	0.956	0.951
	Resource 2	0.931	0.936	0.937	0.931	0.932	0.941	0.929	0.932	0.943	0.919	0.923	0.93	0.841	0.838	0.844
	Resource 3	0.634	0.636	0.644	0.634	0.637	0.649	0.632	0.634	0.65	0.625	0.628	0.636	0.564	0.563	0.57
	Resource 4	0.903	0.905	0.923	0.906	0.904	0.929	0.897	0.903	0.923	0.896	0.897	0.91	0.812	0.809	0.816
	Resource 5	0.874	0.876	0.897	0.875	0.877	0.903	0.871	0.874	0.891	0.866	0.869	0.881	0.785	0.784	0.794
Run 4	Resource 1	0.842	0.842	0.842	0.842	0.842	0.841	0.842	0.842	0.842	0.842	0.842	0.842	0.711	0.711	0.711
	Resource 2	0.716	0.716	0.716	0.716	0.716	0.716	0.716	0.716	0.716	0.716	0.716	0.716	0.609	0.61	0.61
	Resource 3	0.741	0.741	0.741	0.741	0.741	0.741	0.741	0.741	0.741	0.741	0.741	0.741	0.634	0.634	0.634
	Resource 4	0.681	0.681	0.68	0.681	0.681	0.681	0.681	0.681	0.681	0.681	0.681	0.681	0.579	0.579	0.579
	Resource 5	0.954	0.954	0.949	0.954	0.954	0.944	0.954	0.954	0.954	0.954	0.954	0.954	0.805	0.805	0.804
Run 5	Resource 1	0.68	0.683	0.716	0.681	0.683	0.715	0.681	0.683	0.706	0.678	0.68	0.697	0.63	0.63	0.641
	Resource 2	0.707	0.71	0.735	0.706	0.71	0.733	0.707	0.709	0.737	0.704	0.706	0.723	0.667	0.666	0.678
	Resource 3	0.963	0.966	0.976	0.963	0.968	0.974	0.96	0.967	0.977	0.957	0.96	0.971	0.911	0.912	0.913
	Resource 4	0.992	0.992	0.987	0.992	0.992	0.988	0.991	0.991	0.985	0.989	0.99	0.984	0.975	0.976	0.97
	Resource 5	0.679	0.68	0.713	0.678	0.681	0.708	0.677	0.68	0.702	0.677	0.679	0.698	0.636	0.636	0.645
	Average	0.82436	0.82612	0.83928	0.8244	0.8258	0.83776	0.82292	0.82488	0.83688	0.819	0.82048	0.829	0.74044	0.74048	0.74252

Fig. 9. Results of five simulation runs of 2TSB, 2TFB, and 2TFB-SF experiments in terms of resource utilization rate.

5.3.2. 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 0.2. Since the performance of 2TPB can be influenced by slack-factor parameter SF and the

preemption-limit parameter PL, we conduct the following two experiments to observe the effect of these parameters.

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 from 6% to 27% but increases the turn-around time of others by from 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 project arrival rate. As the value of $(\overline{ita}, PL)$ increases from $(10, 1)$ to $(160, \infty)$, the mean turn-around time of the high-priority projects is reduced remarkably from 1.5% to 20%. On the other hand, this also leads to an increase in the mean turn-around time of the low-priority projects by around 0.01% to 10.7%. 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.

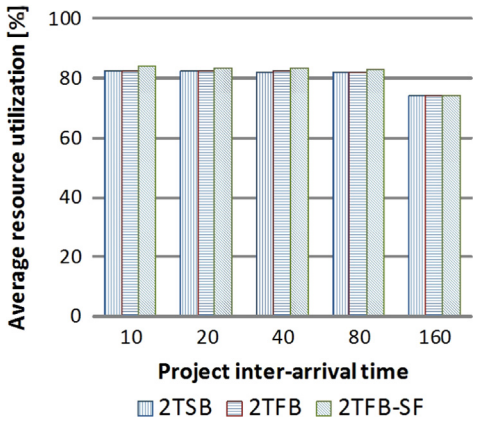

Fig. 10. Average resource utilization rates as the simulation results of 2TSB, 2TFB, and 2TFB-SF experiments.

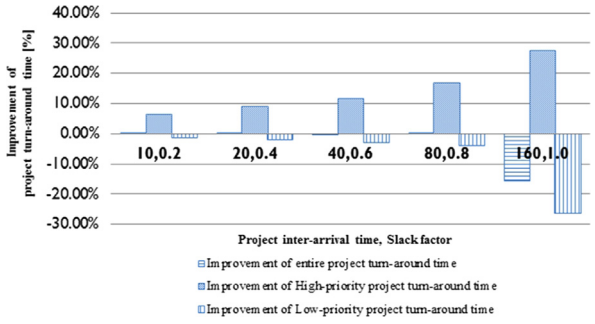

Fig. 11. Improvement on the mean project turn-around time by 2TPB algorithm with differential values of slack factor SF.

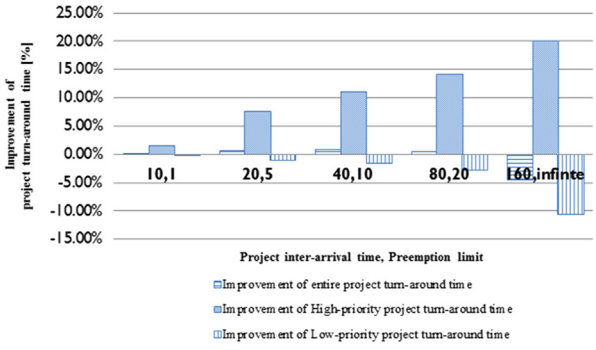

Fig. 12. Improvement of project turn-around time of 2TPB with differential limits on the number of allowable delayed projects PL.

6. Conclusion

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 factor 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 from 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 by 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 $(160, \infty)$, the mean turn-around time of high-priority projects is reduced by 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. 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. 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

Agarwal S, Garg R, Gupta M, Moreira J. Adaptive incremental checkpointing for massively parallel systems. In: Proceedings of SC; 2004. p. 277–86.

Ai L, Tang M, Fidge C. Resource allocation and scheduling of multiple composite web services in cloud computing using cooperative coevolution. In: Proceedings of international conference on neural information processing; 2011. p. 13–7.

Anglano C, Canonico M. Scheduling algorithms for multiple bag-of-task applications on desktop grids: a knowledge-free approach. In: Proceedings of IEEE international symposium on parallel and distributed processing; 2008. p. 14–8.

Benoit A, Marchal L, Pineau J-F, Robert Y, Vivien F. Scheduling concurrent bag-of-tasks applications on heterogeneous platforms. IEEE Trans Comput 2010;59(2):202–17.

Chiang S-H, Arpaci-Dusseau A, Vernon MK. The impact of more accurate requested runtimes on production job scheduling performance. In: Proceedings of the 8th workshop on job scheduling strategies for parallel processing; 2002. p. 103–27.

Csim 20. ⟨http://www.mesquite.com/products/csim20.htm⟩.

Feitelson DG. Experimental analysis of the root causes of performance evaluation results: a backfilling case study. IEEE Trans Parallel Distrib Syst 2005;16(2):175–82.

Feitelson DG, Randolph L, Schwiegelshohn U. Parallel job scheduling v a status report. In: Proceedings of the workshop on job scheduling strategies for parallel processing; 2004. p. 1–16.

Gelenbe E. On the optimum checkpoint interval. J ACM 1979;2(2):259–70.

Google cloud platform, ⟨https://cloud.google.com⟩.

Gopalan K, Chiueh T. Multi-resource allocation and scheduling for periodic soft real-time applications. In: Proceedings of multimedia computing and networking; 2002. p. 34–45.

Holenderski M, Bril R, Lukkien J. Parallel-task scheduling on multiple resources. In: Proceedings of the 24th Euromicro conference on real-time systems; 2012. p. 233–44.

Hoogeveen JA, Vestjens APA. Optimal on-line algorithms for single-machine scheduling. In: Proceedings of the 5th international conference on integer programming and combinatorial optimization; 1996. p. 404–14.

Hou C-J, Shin KG. Replication and allocation of task modules in distributed real-time systems. In: Proceedings of FTCS; 1994. p. 26–35.

Jones JP, Nitzberg B. Scheduling for parallel supercomputing: a historical perspective of achievable utilization. In: Proceedings of the 5th workshop on job scheduling strategies for parallel processing; 1999. p. 1–16.

Katsaros P, Angelis L, Lazos C. Performance and effectiveness trade-off for checkpointing in fault-tolerant distributed systems. Concurr Comput: Pract Exp 2007;19(21):37–63.

Kleinrock L. Queueing systems, vol. I: theory. New York: Wiley Interscience; 1975.

Lawson BG, Smirni E. Multiple-queue backfilling scheduling with priorities and reservations for parallel systems. In: Proceedings of the 8th workshop on job scheduling strategies for parallel processing; 2002. p. 72–87.

Li Y, Mascagni M. Improving performance via computational replication on a large-scale computational grid. In: Proceedings of CCGRID; 2003. p. 442–8.

Li B, Li Y, He M, Wu H, Yang J. Scheduling of a relaxed backfill strategy with multiple reservations. In: Proceedings of international conference on parallel and distributed computing. Applications and technologies; 2010. p. 311–6.

Lifka D. The anl/ibm sp scheduling system. In: Proceedings of the workshop on job scheduling strategies for parallel processing; 1995. p. 295–303.

Lu X, Sitters RA, Stougie L. A class of on-line scheduling algorithms to minimize total completion time. Oper Res Lett 2003;31:232–6.

Mu'alem A, Feitelson D. Utilization, predictability, workloads, and user runtime estimates in scheduling the ibm sp2 with backfilling. IEEE Trans Parallel Distrib Syst 2001;12(6):529–43.

Oliner A, Rudolph L, Sahoo R. Cooperative checkpointing: a robust approach to large-scale systems reliability. In: Proceedings of SC; 2006. p. 14–23.

Ozaki T, Dohi T, Okamura H, Kaio N. Min-max checkpoint placement under incomplete failure information. In: Proceedings of DSN; 2004. p. 721–30.

Shmueli E, Feitelson DG. Backfilling with lookahead to optimize the performance of parallel job scheduling. In: Proceedings of the 9th workshop on job scheduling strategies for parallel processing; 2003. p. 228–51.

Silva D, Cirne W, Brasileiro F. Trading cycles for information: using replication to schedule bag-of-tasks applications on computational grids. In: Proceedings of Euro-par; 2003. p. 169–80.

Sleator DD, Tarjan RE. Amortized efficiency of list update and paging rules. ACM Commun 1985;28(2):202–8.

Srinivasan S, Kettimuthu R, Subramani V, Sadayappan P. Selective reservation strategies for backfill job scheduling. In: Proceedings of the 8th job scheduling strategies for parallel processing; 2002. p. 55–71.

Talby D, Feitelson DG. Supporting priorities and improving utilization of the ibm sp scheduler using slack-based backfilling. In: Proceedings of the 13th international parallel processing symposium and 10th symposium on parallel and distributed processing; 1999. p. 513–7.

Tan J. Semi-online scheduling problems on m parallel identical machines. In: Proceedings of the 6th IEEE joint international information technology and artificial intelligence conference; 2011. p. 289–91.

Vestjens A. On-line machine scheduling [Ph.D. thesis], Department of mathematics and computing science, Technische Universiteit Eindhoven, Eindhoven, Netherlands; 1997. p. 13–7.

Ward, JWA, Mahood CL, West JE. Scheduling jobs on parallel systems using a relaxed backfill strategy. In: Proceedings of the 8th workshop on job scheduling strategies for parallel processing; 2002. p. 88–102.

Xiong Q, Wu C, Xing J, Wu L, Zhang H. A linked-list data structure for advance reservation admission control. In: Proceedings of ICCNMC; 2005. p. 901–10.