Parallel task scheduling under multi-Clouds

Yongsheng Hao[1,2*], Mandan Xia[3], Na Wen[4], Rongtao Hou[5], Hua Deng[4], Lina Wang[6], Qin Wang[5]
[1] Information management department, Nanjing University of Information Science & Technology, Nanjing, 210044, China
[e-mail: yongshenghao@yahoo.com]
[2] State International S&T Cooperation Base of Networked Supporting Software, Jiangxi Normal University, 330022, China
[3] School of languages and cultures, Nanjing University of Information Science & Technology, Nanjing, 210044, China
[4]College of Atmospherics Science, Nanjing University of Information Science & Technology, Nanjing, 210044, China
[5]School of computer and software, Nanjing University of Information Science & Technology, Nanjing, 210044, China
[6] School of electronic & information engineering, Nanjing University of Information Science & Technology, Nanjing, 210044, China
*Corresponding author: Y. Hao

Received August 15, 2016; revised October 31, 2016; revised December 2, 2016; accepted December 4, 2016; published January 31, 2017

Abstract

In the Cloud, for the scheduling of parallel jobs, there are many tasks in a job and those tasks are executed concurrently on different VMs (Visual machines), where each task of the job will be executed synchronously. The goal of scheduling is to reduce the execution time and to keep the fairness between jobs to prevent some jobs from waiting more time than others. We propose a Cloud model which has multiple Clouds, and under this model, jobs are in different lists according to the waiting time of the jobs and every job has different parallelism. At the same time, a new method-ZOMT (the scheduling parallel tasks based on ZERO-ONE scheduling with multiple targets) is proposed to solve the problem of scheduling parallel jobs in the Cloud. Simulations of ZOMT, AFCFS (Adapted First Come First Served), LJFS (Largest Job First Served) and Fair are executed to test the performance of those methods. Metrics about the waiting time, and response time are used to test the performance of ZOMT. The simulation results have shown that ZOMT not only reduces waiting time and response time, but also provides fairness to jobs.

Keywords: parallel tasks, 0-1 integer programming, Virtual machines, multi-Clouds

1. Introduction

The Cloud is a type of parallel and distributed system consisting of a collection of inter-connected and virtualized computers that are dynamically provisioned and presented as one or more unified computing resource(s) based on service-level agreements established through negotiation between the service providers and consumers [1]. In recent years, more and more attention has been given to Cloud market/SLA(service level agreement) management, Green Clouds/resource efficiency, Cloud resource management, Cloud programming models and so on [2].

More and more resources and systems are migrated to Cloud environment. The importance of scheduling methods is apparent in Cloud computing. It not only decides the performance of the Cloud system, but also influences the satisfaction of the Cloud user. Some scheduling methods have been proposed to solve the problem.

Z. Fan et al. extend the OO (ordinal optimization) scheme to meet the special demands from cloud platforms that apply to virtual clusters of servers from multiple data centers [3]. A new RCO (Ranking Chaos Optimization) is proposed to schedule large-scale Cloud resources [4]. With the consideration of large-scale irregular solution spaces, a new adaptive chaos operator is designed to traverse wide spaces within a short time. Besides, dynamic heuristics and ranking selection are introduced to control the chaos evolution in the proposed algorithm.

L.Wenjuan et al. add trust management to check workflows QoS and propose a novel customizable cloud workflow scheduling model [5]. It divides workflow scheduling into two stages: the macro multi-workflow scheduling as the unit of cloud user and the micro single workflow scheduling. Trust mechanism is used to enhance the system performance in different aspects. Some methods also pay attention to energy consumption in the Cloud. F. Satoh et al. develop a Cloud energy management system with sensor management functions, with an optimized VM allocation tool to minimize energy consumption at multiple data centers [6]. A. Alnowiser et al. propose an energy efficient job scheduling approach based on a modified version of weighted round robin scheduler that incorporates VMs reuse and live VM migration without compromising the SLA (Service Level Agreement) [7]. Martin et al. investigate three possible distributed solutions for resource scheduling methods in Cloud [8]: approaches inspired by honeybee foraging behavior [9], biased random sampling and active Clustering [9]. N. Cordeschi et al. [15] propose a scheduling method to minimum-energy which takes account of those aspects at the same time: task sizes, computing rates, communication rates and communication powers. Furthermore, considering the hard per-job delay-constraints, N. Cordeschi et al. [16] propose a scheduling method to minimize the overall computing-plus-communication energy consumption in VNetDC (virtualized networked data center).

The scheduling of parallel jobs is more difficult in comparison with the scheduling of non-parallel jobs. The scheduling goal is to maximize the performance of the system and to minimize unnecessary delays; at the same time, it provides fairness for resource users. The task of the same job can be allocated to different VMs at the same time and those tasks will be executed synchronously. In this way, we avoid that a task waits other tasks of the same job. And in this paper, we suppose every job has some parallel tasks. The scheduling of parallel jobs has been widely studied in the past in distributed system [11, 19], cluster system [10, 13, 20], parallel system [14], Grid and Cloud Computing [14, 21, 22, 23, 24].

Because parallel tasks of a job need to get data from other tasks or provides data for others' tasks frequently [10], we need to find a way to assign those tasks that belong to the same job as soon as possible, and to reduce the execution time. We also need to dynamically initial VMs

(or hosts) to satisfy the system requirement. The main contributions of the paper include: (1) we give a framework for the scheduling of parallel tasks under multi-Clouds; and then (2) we propose our scheduling method based on our analysis of the goals and the requirements; lastly, (3) simulations are executed to test our method under different conditions.

In the simulation, we give a comparison between our method and AFCFS (Adapted First Come First Served) and LJFS (Largest Job First Served), because most scheduling methods of parallel jobs in different areas [10~14] are AFCFS and LJFS, or an extension of them. Fair Scheduler [27] has been widely used in the Cloud, so we give a comparison between Fair Scheduler and our method. Fair Scheduler ensures the job shares more resources if there are some resources having no jobs. The simulations are executed in a simulation environment and on a real log.

This paper focuses on the scheduling of parallel jobs in Cloud computing, and proposes a scheduling method-ZOMT. The structure of this paper is as follows. Section 2 is the related work of scheduling methods of parallel jobs. Section 3 introduces details of workload models. Section 4 proposes our scheduling method based on the analysis of the goals and the requirements. Section 5 presents the parameters of the simulation and discusses the result in the simulation. Finally, in Section 6, we provide some concluding remarks and suggestions for our future work.

2. Related Work

There are many traditional scheduling methods for parallel jobs in different systems. Gang scheduling is a special case of parallel scheduling and many methods of Gang scheduling have been evaluated by researchers. They are adapted first come first served (AFCFS) [12, 13, 14], largest job first served (LJFS) [12, 13, 14] (or LGFS, in fact, A Gang is a big job with some tasks), queue insertions dependent largest job first served (QLJFS) [12], periodic largest job first served (PLJFS) [12]. AFCFS schedules the job if the number of empty VMs is greater than the parallelism of the job. When the number of empty VMs is less than the parallelism of the job in the waiting queue, then the AFCFS policy gives up the job and considers the next job until it finds a job that can be executed. With LJFS, according to the increasing order of the parallelism of the job, jobs are placed in the queue (jobs with large parallel jobs are placed ahead in the queue). All jobs are searched one by one, and the first job starts execution if the number of VMs is more than the parallelism of the job. PLJFS policy rearranges the scheduling order of jobs only at the end of predefined time period *t*. At the end of a period, the scheduler center recalculates the priorities of all jobs in the queue and sorts them in the decreasing order of the parallelism of the job. QLJFS rearranged the scheduling order according to the LJFS method when every job was inserted in the waiting queue. In Cloud, considering the migrations of VM, adapted first come first served with migrations (AFCFSwM) [15] and largest Gang first served with migrations (LGFSwM) [13] are proposed. AFCFSwM algorithm uses AFCFS as the local scheduling method in each system (Cloud, Grid and so on), and at the same time, it also implements local and server migrations. The same as AFCFSwM, LGFSwM is the migration version of the LGFS. At the same time, Gang scheduling also has been widely studied in many areas including the Clusters [10], the distributed system [11, 17], multi-core system[13], parallel system [12, 18] Grid and Cloud Computing [14]. **Table 1** gives a summary of the traditional methods (or based on those traditional methods) that have been discussed in different papers.

Table 1. The scheduling method in different papers in different environments

	Distributed system	Multi-core system	Parallel system	Cluster	Grid Cloud
AFCFS	[11]	[13]	[12]	[10]	[14]
LJFS	[11]	[13]	[12]	[10]	[14]
AFCFSwM	-	-	-	-	[13,15]
LGFSwM	-	-	-	-	[13,15]

In fact, considering more aspects of the scheduling of parallel jobs, more methods have been given by researchers. Z. Fan et al. extend the OO (ordinal optimization) scheme to meet the special demands from cloud platforms that apply to virtual clusters of servers from multiple data centers [3]. In [19], the scheduling goal is to maximize the number of finished jobs that have been completed before their deadlines. Considering the scheduling of non-malleable parallel tasks and malleable tasks specially, they propose two polynomial-time approximation algorithms for different tasks. EDF (earliest deadline first) is used to schedule non-malleable tasks and an extension of EDF is used to schedule malleable tasks.

G. B. Jorge et al. try to solve the problem of minimizing the makespan of a batch of jobs with different system loads [20]. They schedule the resource dynamically. When a new job coming that may result in changing the assigned resources to other jobs during its execution. The scheduling method includes two steps: a scheduling strategy and a scheduling algorithm. An adaptation of the HEFT (Heterogeneous Earliest-Finish-Time) algorithm is proposed to solve the problem of scheduling of parallel tasks in heterogeneous environment.

The AsQ (Adaptive Scheduling with QoS Satisfaction algorithm) [21] tries to find a resource scheduling scheme for different aspects: the utilization rate of the private cloud, the renting expense, and the task finish time are optimized and so on. They take the scheduling of the parallel tasks as a variation of the multi-dimension multi-choice knapsack problem and they develop a fast scheduling strategy which ensures the cost and the deadline constraints.

H. Ting et al. consider the problem of scheduling of parallel tasks of different priorities [22]. Low-priority tasks may be assigned to underutilized computation resources which have been assigned to some high-priority tasks. But this may result in some tasks waiting more time and some servers migrating dynamically. The goal of scheduling is to allocate the low-priority tasks to low load server resources while minimizing the combined cost of waiting and migration. They solve the scheduling problem as restless MAB (Multi-Armed Bandits).

Scheduling parallel scientific workflows in the cloud is a very complex task since the scheduling may have different goals and criterions. In [23], the authors introduce an adaptive scheduling heuristic for parallel execution of scientific workflows in the cloud that is based on three criterions: total execution time (makespan), reliability and financial cost. They propose a weighted cost function and an adaptive scheduling heuristic to explore the dynamicity of clouds while scheduling Cloud activities to several VMs in a virtual cluster. They not only implement the cost model but also evaluate the scheduling algorithms in SciCumulus.

In the paper, we only pay attention to AFCFS and LJFS in the simulation, because most of works of parallel scheduling are based on them and the past works also show that they (or one of them) have better performance (especially AFCFS). For the scheduling of parallel tasks, migrations may bring many problems for the scheduling, because one job has many tasks which are executed at the same time, and one migration of a VM not only influences the task, but also influences the tasks which belong to the same job, so we do not take account of the migrations of the parallel tasks in the paper. Though we do not take account of migrations of VMs, simulations evaluate the performance of our method when the VM can shut off or

migrate under some assumptions.

3. System model

Tasks of a job are required to send and receive data very frequently in the same jobs. Thus, the number of VMs required by the job and its parallelism have the same values. In the model under our study, parallelisms of those jobs are random numbers following the discrete uniform distribution and the maximum of the parallelism is *maxplism*. The mean time between jobs' arriving is exponentially distributed with a mean of 1/λ. There exists no correlation between service time and the parallelism of jobs. For example, a job with large parallelism does not mean to must have a long service time. A large job (We call it LPJ in the following paper) refers to the job with big value of parallelism. It needs more processors at the same time, and this does not mean that it needs more time than others.

According to the parallelism of the job, there are two kinds of jobs: small parallel jobs (SPJ) and large parallel jobs (LPJ) [12]. The parallelism of SPJ has a scope of [1, *midv*] and its probability is *q*. *midv* is the maximum value of the parallelism of SPJ. LPJ has the parallelism in the range of [*midv*+1, *midl*] and its probability is 1-*q*. *midl* is the maximum value of the parallelism of LPJ. The two kinds have the same scopes. In other words, *midl* is double of *midv*. So, we can get the average parallelism (*AP*) of every job as

$$AP=q*(1+midv)/2+(1-q)*(1+midv+midl)/2 \quad (1)$$

There are two kinds of jobs according to the parallelism of jobs: SPJ or LPJ (Fig. 1). They arrive at the average rate of λ. RS (Resource Schedule) is in charge of the resource scheduling. As in [3], RS controls the application-to-VM level and it assigns tasks of a given job to different VMs at the same time. The same model is also taken by [14].

4. Scheduling method for parallel tasks in multi-Clouds

4.1 job routing

In this paper, our routing method fills the same time empty queue (FSTEQ, see Algorithm 1). Every data center registers the relative data with the details of resources to the *RS*. *RS* manages the resource and monitors those states of resources. For the job j_{temp}, when it comes, first of all, it is inserted into the waiting list *wlist* (lines 1-2, Algorithm 1, same in the following paper). RS (Resource scheduler) is in charge of resource scheduling which takes FSTEQ algorithm for the system. FSTEQ checks the waiting time (wt_{temp}) of every job, and if the value is more than *time1*, the job is inserted into the scheduling list *slist*. *time1* ensures that every job does not wait too much time. Then, if there are more places for more jobs, FSTEQ checks every job in the waiting list *wlist* and inserts some of them that can be inserted into *slist*. Tasks of the same job must be assigned to different queues in the same positions since the scheduling requires that there exists a one-to-one mapping of tasks to VMs (**Fig. 1**). See job(*i*), T(1), T(*pos*) and T(*j*) are the tasks of job(*i*). RS checks the resource utilization rate and the finished rate of jobs periodically and decides to initiate (or stop) a new VM or a new Cloud server.

Algorithm 1 gives the details of the FSTEQ. Lines 1~5 check whether there are places for the new coming parallel jobs. "checkempty(*slist*)" returns "true" when there are enough resources for the job (lines 1~2, Algorithm 1); otherwise, it returns "false" (lines 3~5, Algorithm 1). If "checkempty(*wlist*)" returns "true", the job will be placed in *wlist* (lines 4,

Algorithm 1).The function of " Insertlist($j_{temp}, wlist$) " is to insert the job j_{temp} to *wlist* and returns the new *wlist*. Then, we will check the waiting time if the jobs in the *wlist*, and if the waiting time is more than *time1*, the job will be inserted to *slist* (lines 6~12). Furthermore, if there are more places for the jobs in *wlist*, we will insert more jobs to *slist* as the AFCFS (Adaptive First comes first service) policy. At last, the system checks the system load condition to decide whether it needs to initiate (or stop) a new VM or a new Host (line 20).

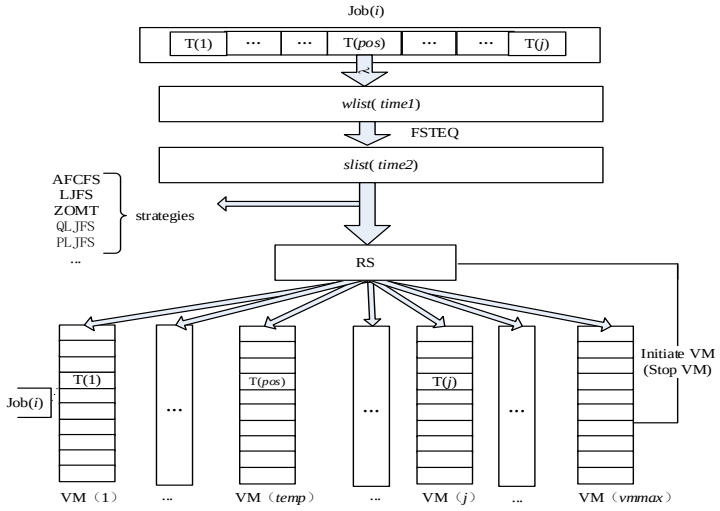


Fig. 1. FSTEQ system model

Algorithm 1: FSTEQ ( )

1. **If** checkempty(*wlist*)
   //Checks there are places in *wlist*, and if there are places, j_{temp} is inserted into *wlist*
2. $slist = \text{Insertlist}(j_{temp}, wlist)$
3. **Else**
4. $wlist = \text{Insertlist}(j_{temp}, wlist)$
   // otherwise, the job is inserted into *wlist*
5. **Endif**
6. **For** $j_{temp} \in wlist$
7. **If** wt_{temp} >*time1*
8. **If** Checkempty(*slist*)
9. $slist = \text{Insertlist}(j_{temp}, slist)$
10. **Endif**
11. **Endif**
12. **Endfor**
13. **If** Checkempty(*slist*)
14. **For** $j_{temp} \in wj$
15. **If** Checkempty(*slist*)
16. $slist = \text{Insertlist}(j_{temp}, slist)$ //Insert jb_{temp} into the scheduling list *slist*;

17. **Endif**
18. **Endfor**
19. **Endif**
20. Check the system load condition to decide whether it needs to initiate (or stop) a new VM or a new host.

To enhance the utilization ratio of resources, we can stop some VMs when we find the resources are enough and some resources always cannot get tasks, in others words, when the supply of resources is more than the demand of resources. On the contrary, if there are many jobs cannot be finished, we can initiate some VMs for the job (Line 16). Suppose that the average arrival rate of jobs is λ and the average parallelism is *AP*, then the number of VMs that we need is *needvm*:

$$needvm = \lambda * AP$$

First of all, we initiate all the 120 VMs. We only initiate 120 VMs because the maximum of the leasing VMs of a host of a Cloud center is 120 [28]. Then, according to the value of the jobs that have been finished, we get the value of λ and *AP*. To ensure the system performance, we initiate more VMs than we need. The number of initiated VMs is *inivm* ($1 > \alpha > 0$):

$$inivm = \lambda * AP * (1 + \alpha)$$

Too many VMs in the same host reduce the performance of every VM in the host. A host (or server) has a maximum number of VMs. In Fig. 1, if the online server has more ability to initiate a new VM, we will use the server until it has no ability to support a new VM; otherwise, we will initiate a new host for the initialization of the new VM. A simple process for the server and VM initial is as follows:

(1) For the new VM *cv*, computer checks every host with the relative attributes whether they can satisfy *cv*. Those attributes include: virtualization software, CPUs type and computational capabilities, shared hardware, the usage of shared storage and so on. We sort those hosts according to the ascending order of the leaving attributes (CPU, memory, and bandwidth).

(2) If we can get hosts in step 1, we initial *cv* in the first host, otherwise, we go to step (4);

(3) If a host has not been assigned any tasks for long time, we will shut the host to reduce energy consumption;

(4) If we cannot get host in step 2, we initiate a host;

(5) Repeat steps 1-5 for the new request of VM.

4.2 job scheduling

In section 2, we have introduced other scheduling methods. And in this section, we will propose our scheduling method-ZOMT.

Let *vmmax* be the maximum number of VMs. The parallelism of job_i is pl_i. The scheduling result of job_i is sc_i, if sc_i equals to 1, the job has been scheduled; otherwise, the job has not been scheduled yet. The waiting time of job_i is wt_i.

If some jobs wait more time than others, it is unfair to those jobs. So we set *wte* to denote the longest time that the job waits in the queue (See Fig.1, it is *time2*). If the waiting time is more than *wte*, the job is scheduled immediately. Suppose those jobs whose waiting time is more than *wte* are listed in the list of *olist* and others are listed in the list of *wlist*:

$$olist=\{job_1, job_2, job_3,\ldots,job_{onum}\}$$

$$wlist=\{job_1, job_2, job_3,\ldots,job_{wnum}\}$$

The number of VMs that is requested by the jobs in *olist* is

$$sumo=\sum_{temp=1}^{onum} pl_{temp} \tag{2}$$

After this scheduling of *olist*, the number of leaving VMs is

$$lnum=vmmax-sumo$$

Then the problem is how to schedule the job in *wlist* with *lnum* VMs. The scheduling must meet the condition:

$$\sum_{temp=1}^{wnum} (pl_{temp} \times sc_{temp}) \le lnum \tag{3}$$

At the same time, there are two goals in the scheduling: (1) to maximize the total value of parallelisms of all scheduled jobs, most resources have been used by jobs (2) to maximize the waiting time of all scheduled jobs, we can reduce the jobs' waiting time:

$$\begin{cases} (1)\max(\sum_{temp=1}^{wnum} (pl_{temp} \times sc_{temp})) \\ (2)\max(\sum_{temp=1}^{wnum} (pl_{temp} \times wt_{temp} \times sc_{temp})) \end{cases}$$

The coefficients are listed as:

$$oarray=[pl_1.*wt_1., \ldots, pl_{temp}.*wt_{temp}, \ldots, pl_{wnum}*wt_{wnum}]$$

$$plarray=[pl_1, \ldots, pl_{temp}, \ldots, pl_{wnum}]$$

$$scresult=[sc_1, \ldots, sc_{temp}, \ldots, sc_{wnum}]$$

oarray, *plarray*, and *scresult* are the sets of the weighted waiting time of those jobs, the parallelisms and the scheduling result, respectively.

We suppose the two goals have the same weight to the system, so the goal becomes:

$$tempmax=(0.5*\sum_{temp=1}^{wnum} (pl_{temp} \times sc_{temp} + 0.5*\sum_{temp=1}^{wnum} (pl_{temp} \times wt_{temp} \times sc_{temp})) \tag{4}$$

An implied condition is that we should schedule as many as possible of the number of VMs to enhance the resource utilization efficiency. Let us suppose that at least *rratio* resources (ratio) have been assigned to different jobs. sc_{temp} represents that whether the job has been assigned. If sc_{temp} equals 1, it has been assigned to a resource, otherwise, it waits more time.

There must exist:

$$\sum_{temp=1}^{wnum} (pl_{temp} \times sc_{temp}) \geq rratio \times lnum \tag{5}$$

i.e.

$$\sum_{temp=1}^{wnum} (-pl_{temp} \times sc_{temp}) \leq -rratio \times lnum \tag{6}$$

rratio has a range of [0, 1].

From equations (4), (5), (6), the scheduling becomes a 0-1 integer programming with many conditions and one goal, which can be solved with enumeration (some other methods also can be used to solve the problem). Tools including MATLAB provide the method to solve the problem and the method is *bintprog* (http://www.hackchina.com/en/r/153791/bintprog.m__html). *oarray* is the coefficient of the objective function. *plarray* and *-plarray* are the coefficient of ***getmax***. Formula (3) and (6) are the constraints, and (4) is the objective function.

The algorithm of our proposed scheduling method is given as follows:

Algorithm 2: ZERO-ONE scheduling in One Cloud center

Input: $olist=\{job_1, job_2, job_3,\ldots,job_{onum}\}$
$wlist=\{job_1, job_2, job_3,\ldots,job_{wnum}\}$
$tlvm=vmmax$;

Output: $scresult=\{value_1,value_2, value_3,\ldots,value_{temp},\ldots, value_{wnum}\}$

1. **For** *i*=1:*onum*
2. **If** *tlvm >= job*$_i$*.pl*
3. Scheduling(*job*$_i$) ;
4. *tlvm = tlvm -job*$_i$*.pl*;
5. **Endif**
6. **Endfor**
7. **[***scresult***,** *fval* **]=*getmax*(** *tempmax***, [** *plarray***; -** *plarray* **], [***tlvm***,** *rratio* tlvm***])**
8. **For**(*temppos*=1*:wnum)*
9. **If** (*scresult* (*temppos*)==1)
10. Scheduling(*job*$_{temppos}$) ;
11. *tlvm = tlvm - job*$_{temppos}$*.pl*;
12. **Endif**
13. **Endfor**
14. **If** sum(*scresult*)==0
15. Scheduling all jobs as AFCFS
16. **Endif**
17. **For** *i*=1:*wnum*
18. **If** *scresult >= job*$_i$*.pl && scresult*(*temppos*)==0 ;
19. Scheduling (*job*$_i$) ;
20. *tlvm = tlvm -job*$_i$*.pl*;
21. **Endif**
22. **Endfor**

Lines 1-6 are the scheduling of the jobs that have waited for a long time, i.e., the jobs in "*Scheduling list*"(Fig. 1). We use AFCFS policy for those jobs. After scheduling, all jobs in *slist* are assigned to the resources. Line 7 is ZONE-ONE integer programming with two requirements and two goals. ***getmax*** returns the goal function (Formula 4) with two conditions (Formula (3, 6)). The target is formula (4). The two conditions requirements are formula (3) and (6). *scresult* is an array of 1**wnum*. If *scresult*[*temp*] is equal to 1, the job_{temp} is scheduled by the Cloud; otherwise, the job cannot be scheduled and waits for the next time scheduling (lines 8-13). *fval* is the minimum of formula (4) and it is the value of the target function, which is the maximum of the multiple target functions. If 0-1 integer programming has no answer to the problem, the summation of *sresult* equals 0, we schedule the system as AFCFS. Sometimes, after the 0-1 integer programming, some resources may not be assigned to any jobs and we can schedule more jobs again as AFCFS (lines 17-22). We schedule resources in this way to enhance resource utilization efficiency.

The maximum of the VMs of the host of a Cloud center is *vmmax*=120 [28]. So, we have no efficiency problem to solve the 0-1 integer problem in the scheduling in one host. *bintprog* is used to solve the problem of ZONE-ONE integer programming in matlab and it can be used in our solution to take place of "***getmax***" function. We test "*bintprog*" in our simulation environment, the maximal value of executing time is less than 10 seconds, which is a very low value to the execution time of the job in the Cloud. The job in the Cloud always needs many minutes even many hours. So, for a Cloud center, we don't need to take account of the execution time of Algorithm 2.

But for a real Cloud system, there are many servers and every server has many VMs, Algorithm 2 has an efficiency problem in the real system. To overcome this shortcoming, we should find a method with higher efficiency for the scheduling in the Cloud center. Algorithm 3 is the new method.

Algorithm 3: ZOMT()

Input:*olist*={job_1, job_2, job_3,…,job_{onum}}
wlist={job_1, job_2, job_3,…,job_{wnum}};
Output: *jscresult*={$value_1$,$value_2$, $value_3$,…,$value_{temp}$,…, $value_{wnum}$};

1. Sort(*olist*);
   //Sort jobs according to the descending order of parallelisms;
2. For *i*=1:*onum*
3. **If** *tlvm* >= $job_i.pl$
4. Scheduling(job_i) ;
5. *tlvm* = *tlvm* -$job_i.pl$;
6. **Endif**
7. **If** *tlvm*<120
8. initiate(*tlvm*);
9. **Endif**
10. **Endfor**
11. **For** *temp*=1:*onum*
12. **For** $j_t \in olist$
13. **If** *tlvm* >= $j_t.pl$
14. Scheduling(j_t) ;
15. *tlvm* = *tlvm* - $j_t.pl$;
16. **Endif**
17. **If** *tlvm*<120

```
18.          initiate(tlvm);
19.        Endif
20.      Endfor
21.    Endfor
22. While Empty(olist)
23. Do
24.      For temp=1:wnum
25.        For j_t ∈ wlist
26.          If tlvm >= j_t.pl
27.            Scheduling(j_t) ;
28.            tlvm = tlvm - j_t.pl;
29.          Endif
30.          If tlvm<120
31.            initiate(tlvm);
32.          Endif
33.      Endfor
34.    Endfor
35.  While Empty(wlist)
36. Algorithm 2()
```

onum and *wnum* are the numbers of jobs in *olist* and *wlist*. *olist* and *wlist* are the sets of the jobs whose waiting time is more than the expected time and the jobs in the waiting list. $job_i.pl$ is the parallelism of the job job_i. In Algorithm 3, lines 1-7 are scheduling the jobs whose waiting time is more than a fixed value, we take the AFCFS policy for those jobs, which is the same as Algorithm 2. We schedule jobs in *olist* first. *olist* is a set of jobs that wait for being scheduled, but those jobs can also wait yet. *tlvm* records the total number of VMs. First of all, we sort jobs in *olist* according to the parallelisms of the jobs (Line 1, algorithm 3, same in the following paper) and then we schedule those jobs according to the AFCFS policy. If the selected data center has no enough resources for the selected job,we will find a new data center which has the largest bandwidth between the selected data center and the new one. We selected the largest bandwidth because the bandwidth also influences the synchronization of the parallel tasks. "initiate(*tlvm*)" tries to find the new data center and it adds the resource fragment to the new data center (line 8; lines 19 and 33 have the same meaning). Then we schedule the jobs in *wlist* (lines 11~38). First of all, we also take AFCFS policy for the scheduling of the jobs in *wlist*. When there are resource fragments in the data center, we also add those resource fragments to a new data center (lines 24~37). We also select the new data center which has the largest bandwidth to the current selected Cloud data center. During the period, we check the requirement of leaving jobs to the number of VMs to see whether it is less than 120 (line 30), and if the value is less than 120, we will use algorithm 2 to schedule the rest of jobs and VMs. The reason why we select 120 as a checkpoint: (1) A host (or server) of a Cloud center can initial the maximum number of VMs is *mavmmax*=120 [28]; (2) there are only a few seconds to get the result of 0-1 integer programming, and we have no efficiency problem under *mavmmax*=120 (Algorithm 2).

4.3 Complexity analysis

Suppose there are *onum* jobs in *olist* and *wnum* jobs in *wlist*. Our scheduling framework has three algirithms and the complexity is decided by the three algorithms:

Algirithm 1 has a complexity of O(*onum*+*wnum*).

The complexity of algorithm 2 is decided by the 0-1 integer programming. But in fact, the maximum of the number of VMs in a data center is a constant (such as 120) and the rest jobs in *wlist* is also less than the constant. So, the complexity of algorithm 2 is O(1)

The complexity of algorithm 3 is decided by the main three steps: allocation of the jobs in *olist*, allocation of the jobs in *wlist* and Algorithm 2:

For the allocation of jobs in *olist*, the sort of the jobs in *olist* is O(*onum*log(*onum*)), lines 2-10 are the AFCFS policy and they have a complexity of O(*onum*). So, the complexity of the first step is:

$$O(onum\log(onum)) + O(onum) = O(onum\log(onum))$$

For the second step, lines 11~21 are also the AFCFS for the jobs in *wlist*, and the complexity of the second step is $O(wnum)$.

For the third step, the complexity is decided by Algorithm 2.

So, the complexity of algorithm 3 is:

$O(onum\log(onum)) + O(wnum) + O(1) = O(onum\log(onum)) + O(wnum)$.

So the complexity of our method is:

$$O(onum + wnum) + O(1) + O(onum\log(onum)) + O(wnum) = O(onum + wnum) + O(onum\log(onum))$$

Suppose there are *n* jobs in *wlist* and *olist* ($n = onum + wnum$), the complexity of our method is $O(n\log(n))$.

5. Simulations and Analysis

All the parameters that we try to evaluate are given in Table 2. *ART* (Average Response Time) and *AWRT* (Average Weighted Response Time) are the average response time and average weighted response time of the finished jobs. *AWT* (Average Waiting Time) and *AWWT* are the average waiting time and the average weighted waiting time of the finished jobs. *SDRT* (Standard deviation of waiting time) is sandard deviation of waiting time of the finished jobs and *SDWWT* (Standard deviation of weighted waiting time) is the standard deviation of weighted waiting time of the finished jobs which takes account of the parallelism of jobs. *SDWT* and *SDWWT* explain the fairness among jobs. Some jobs may wait more time than others. Formulas (7), (8) and (9) give the calculation methods of *SDWT* and *SDWWT*.

$$sumtemp = \sum_{temp=1}^{N} pl_{temp} \times (wt_{temp}/pl_{temp} - AWWT)^2 \quad (7)$$

$$SDWT = \sqrt{\sum_{temp=1}^{N}(wt_{temp} - AWT)^2/N} \quad (8)$$

$$SDWWT = \sqrt{sumtemp/\sum_{temp=1}^{N} pl_{temp}} \quad (9)$$

We set *midl*=16 and *vmmax*=120 in formula (1). The maximum of VMs is *vmmax*, so

$$\lambda < vmmax/AP \quad (10)$$

We will evaluate the system when it has different ratios in different kinds of jobs, so we evaluate the system when the values of q are 0.25, 0.50, and 0.75 respectively. According to (1) and (10):

-When q=0.25, most of jobs belong to SPJ, and the average parallelism of jobs is 10.1250, λ<18.82;

Table 2. Metrics of the simulations

Symbol	Meaning
N	The total number of jobs
joblist	$\{job_1, job_2, job_3, \ldots, job_N\}$
pl_{emp}	The parallelism of job_{temp}
wt_{temp}	The waiting time of job_{temp}
q	The percent of jobs that belong to SPJ
1-q	The percent of jobs that belong to LPJ
λ	Mean arrival rate of jobs
$1/\lambda$	Mean inter-arrival rate of jobs
Totaltime	The time from the first job coming to the last job be finished
ART	Average response time
AWRT	Average weighted response time
AWT	Average waiting time
AWWT	Average weighted waiting time
SDWT	Standard deviation of waiting time
SDWWT	Standard deviation of weighted waiting time

-When q=0.50, there are the same numbers for the SPJ and LPJ, and the average parallelism of jobs is 8.2500, λ<14.54;

-When q=0.75, most of jobs belong to LPJ, and the average parallelism of jobs is 6.3750, λ<11.85.

From the above analysis, we set λ between 5 and 8.5 and the step is 0.5. There are 100000 jobs totally. We run the system 50 times and get the average value of every parameter. We set $\alpha = 0.5$, because all the algorithms have a better performance at this time, and at the same time, the utilization efficiency of the system is kept at a higher level. If the weighted waiting time of a job is doubled to the average weighted waiting time, the jobs will be scheduled first. Suppose the execution time of a task is a random between 1 and 5 on a standard VM.

We will compare our method with AFCFS, LJFS and Fair. We have introduced AFCFS and LJFS in section 1. For Fair, we suppose the algorithm can be paralleled under *N (N* is an integer) times of the parallelism.

We utilized Mathlab (2014a) programming language to implement these algorithms and ran them on an Intel Pentium (R) D 3.4 Ghz, 4 GB RAM desktop PC with 100 MB/s Ethernet card, Window 8.

5.1 *ART* and *AWRT*

Figs. 2~7 depict the comparison of the average response time and average weighted response time of AFCFS, LJFS, Fair and ZOMT when q=0.25, q=0.50 and q=0.75, respectively. As showed in those figures, there is not a big difference in the performance of AFCFS, LJFS, ZOMT when the arrival rate λ is low, especially when λ<6.5.

The values (*ART* and *AWRT*) of the three methods remain stable from 5 to 6.5. With the increase of λ, especially when λ>6.5, in general, the values of the three algorithms rise more

quickly than before. ZOMT always has the lowest value of *ART* and *AWRT* of the four methods. ZOMT keeps the average response time and average weighted response time increasing more slowly than the others.

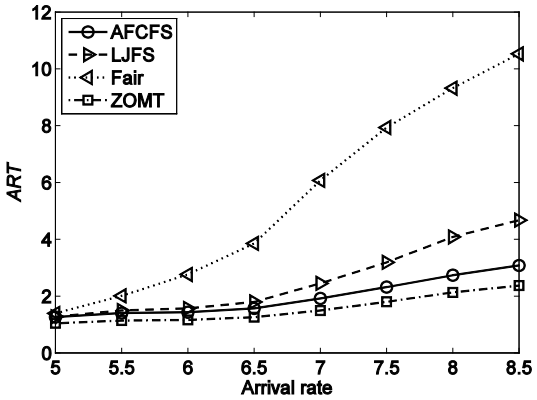


Fig. 2. *ART* vs. λ, *q*=0.25

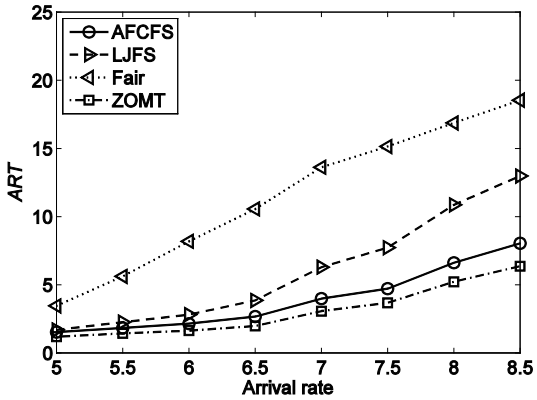


Fig. 5. *ART* vs. λ, *q*=0.50

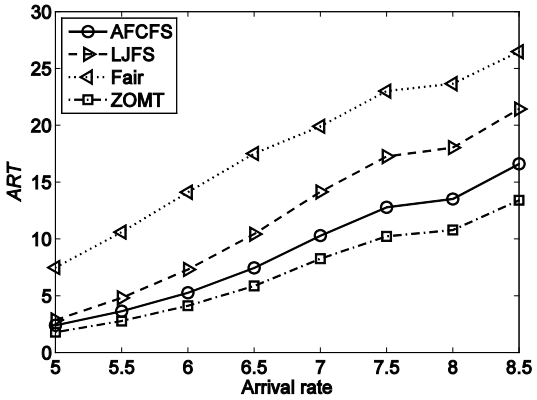


Fig. 4. *ART* vs. λ, *q*=0.75

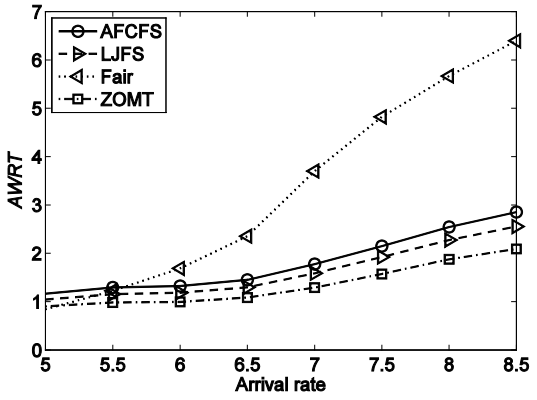


Fig. 5. *AWRT* vs. λ, *q*=0.25

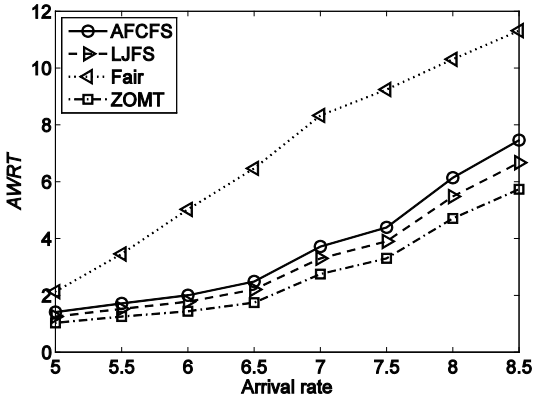


Fig. 6. *AWRT* vs. λ, *q*=0.50

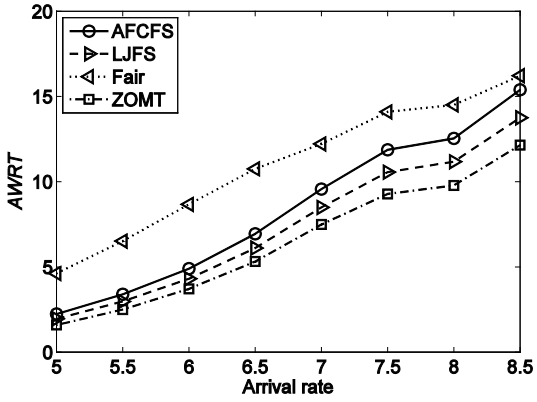


Fig. 7. *AWRT* vs. λ, *q*=0.75

From **Figs. 2~7**, we find that *ART* and *AWRT* of the four methods have the same tendency when the arrival rate is the same. Fair is the large one, followed by AFCFS, LJFS and ZOMT, respectively. With the growing of arrival rate, distinct differences of the two parameters are being postponed. Significant difference of the two parameters emerges from λ=6.5. Fair always has the largest values in *ART* and *AWRT* in the four methods.

For all the cases, to the value of *ART* of AFCFS, LJFS and Fair, ZOMT average reduces by

11.37%, 22.48% and 40.21%; to the value of *AWRT* of AFCFS, LJFS and Fair, ZOMT average reduces by 12.42%, 6.21% and 33.49%.

ZOMT has better performance because it not only makes the jobs which have more waiting time scheduled firstly, but also uses the method to enhance the resource utilization at any time. ZOMT has the lowest value in *ART* and *AWRT* because: (1) if the waiting time of a job is more than a certain value, it will be scheduled immediately; (2) ZOMT ensures most of resources can be used by the job. The two aspects reduce the waiting time and enhance the resource utilization ratio.

5.2 *AWT* and *AWWT*

Figs. 8~13 show the comparison of the average waiting time and average weighted response time given by AFCFS, LJFS, Fair and ZOMT for q=0.25, q=0.50 and q=0.75, respectively.

It is shown from those figures, the average waiting times of all methods grow more and more quickly with the arrival rate increasing. When λ<6.5 (especially for q=0.50 and q=0.75), AFCFS, LJFS and ZOMT tend to offer the same waiting time and weighted waiting time basically.

The values (*AWT* and *AWWT*) of ZOMT are lower than others. ZOMT has higher resource utilization. ZOMT tries its best scheduling as much as possible, so it reduces the value of *AWT* and *AWWT*. For all the simulation result, to the value of *AWT* of AFCFS, LJFS and Fair, ZOMT average reduces by 7.41%, 12.31% and 29.49%; to the value of *AWWT* of AFCFS, LJFS and Fair, ZOMT average reduces by 8.39%, 6.79% and 45.59%.

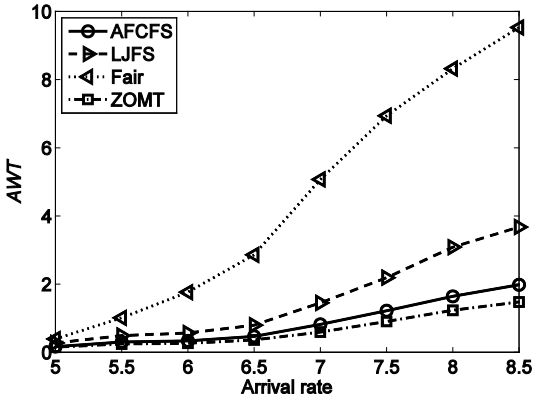


Fig. 8. *AWT* vs. λ, q=0.25

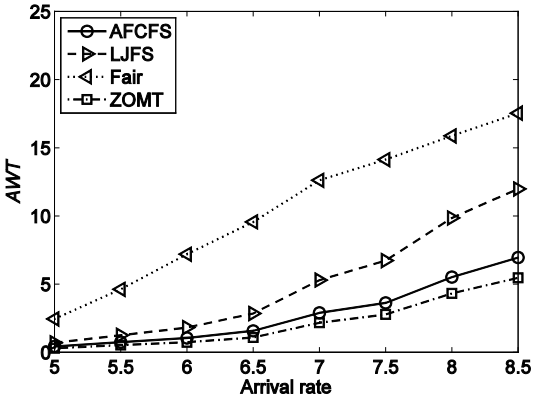


Fig. 9. *AWT* vs. λ, q=0.50

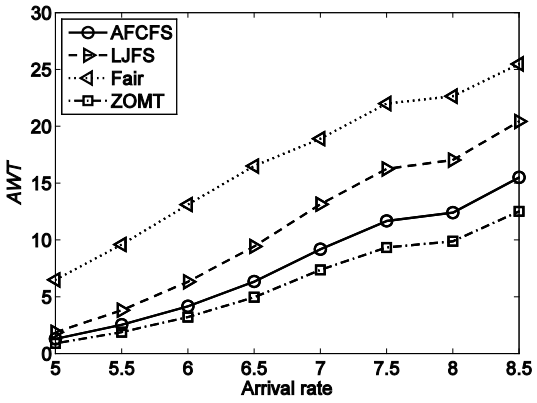


Fig. 10. *AWT* vs. λ, q=0.75

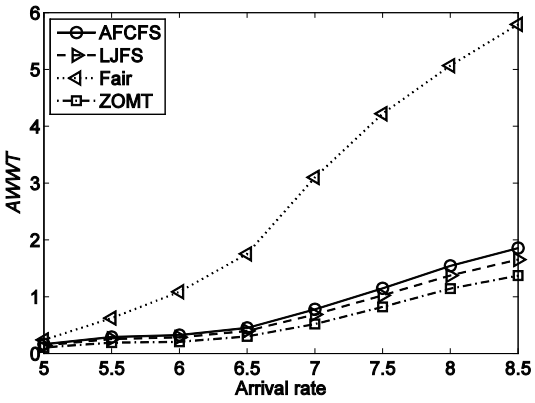


Fig. 11. *AWWT* vs. λ, q=0.25

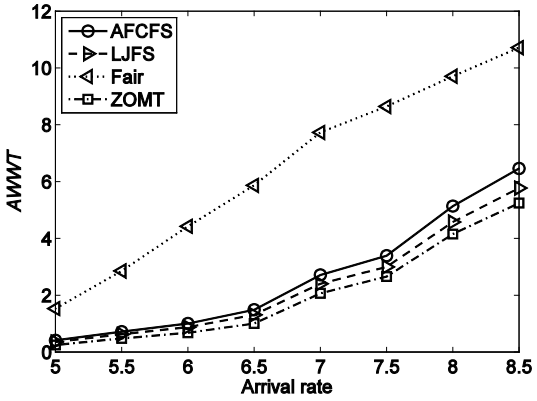


Fig. 12. *AWWT* vs. λ, q=0.50

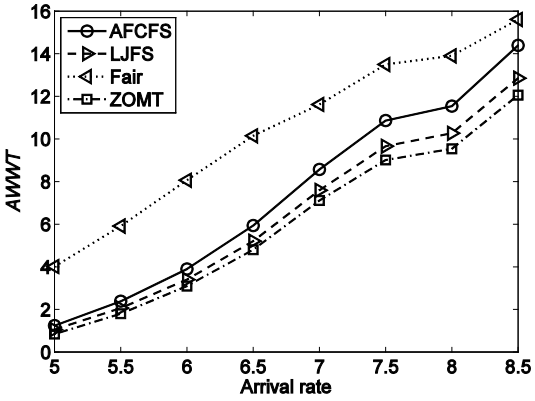


Fig. 13. *AWWT* vs. λ, q=0.75

5.3 *SDWT* and *SDWWT*

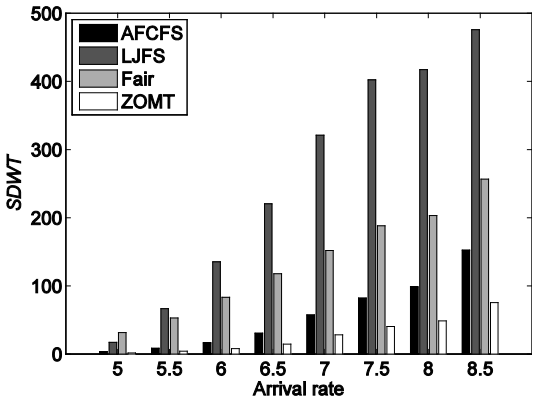


Fig. 14. Average *SDWT* vs. λ under q=0.75

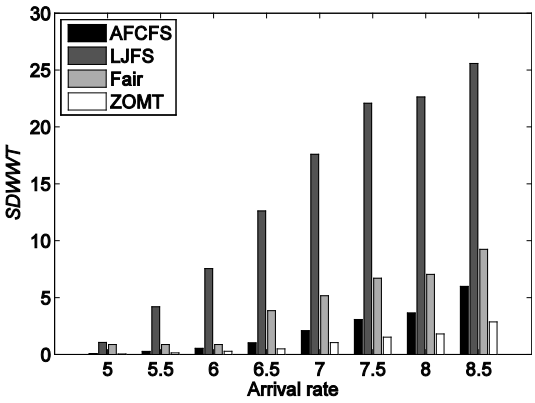


Fig. 15. Average *SDWWT* vs. λ under q=0.75

Figs. 14~15 show the comparison of average standard deviation of waiting time and average standard deviation of weighted waiting time given by AFCFS, LJFS, Fair and ZOMT when q equals 0.75. Those two values illustrate the fairness among jobs. If the value is too high, this means that some jobs wait more time than others.

As shown in those figures, besides Fair and ZOMT, standard deviation of waiting time and standard deviation of weighted waiting time of the other two algorithms grow when the arrival rate becomes higher. ZOMT and AFCFS give the same performance basically and they have a smaller value to LJFS and Fair. The value of *SDWT* and *SDWWT* of ZOMT is always a little less than the values of AFCFS. We also find that the standard deviations of *ART* and *AWRT* have the same patterns with *SDWT* and *SDWWT*. So, we do not introduce them here.

From the simulations, we find that Fair does not give a good performance in the scheduling of parallel jobs with different parallelisms especially when the arrival rate is a high value. The reason is a job in the scheduling needs more VMs. If the Fair scheduler is used, one job takes more VMs, and it may influence other jobs.

5.4 Simulation for the imgration of VMs

In the simulation, suppose there are about 1~5% percent for the migration of VMs in every hour. The time for the migration is about 2~10 minutes. Because the parallel tasks of the same job need to be kept in step in computing, and if a task is stoped, all tasks that belong to the

same job must be stoped and wait until the new VM is initialed. We also suppose the system supports "slot" [33], so we can begin execution from the most recent slots. And in the simulation, we suppose the slot is a time unit. We suppose q equals 0.5 in the simulation.

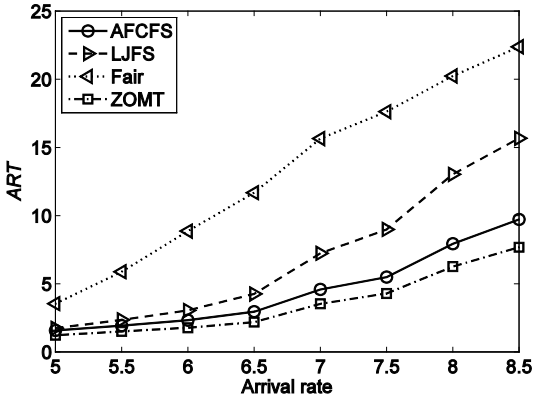


Fig. 16. *ART* vs. λ (q=0.50) with migration

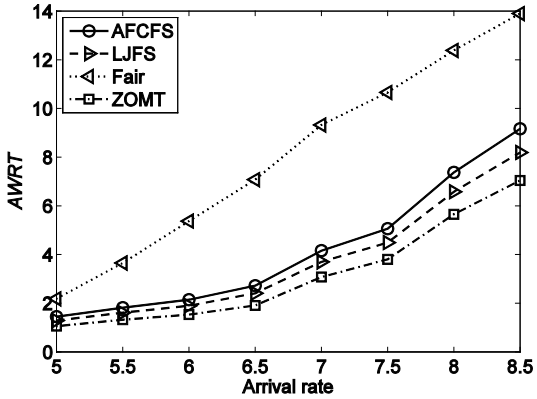


Fig. 17. *AWRT* vs. λ (q=0.50) with migration

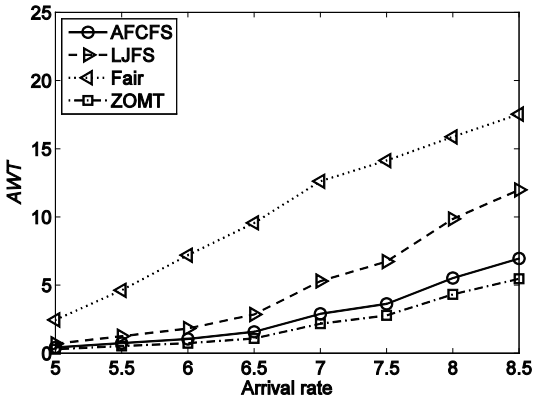


Fig. 18. *AWT* vs. λ (q=0.50) with migration

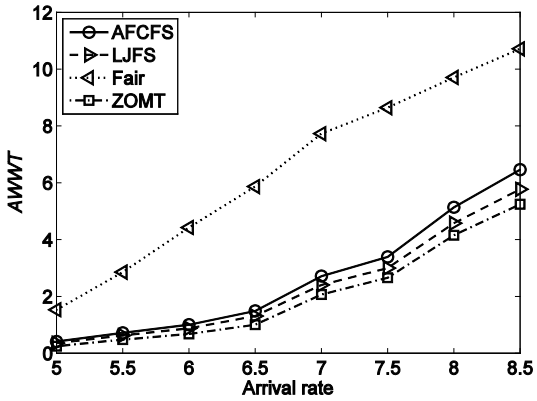


Fig. 19. *AWWT* vs. λ (q=0.50)with migration

From **Figs. 16~19**, we can find the four methods have the same trends to the result when q equals 0.5 in section 5.3. ZOMT always has the lowest value in *ART*, *AWRT*, *AWT* and *AWWT*. So, even though there are some VMs that need to be migrated in the scheduling, ZOMT also has the best performance in those four parameters. Compared to the result in section 5.3, those four methods both have a increase in those parameters when there are some VMs need that to be migrated. With the same method, it has an increase in those four parameters when there are migrations of VMs.

5.5 Simulations on a real log

This simulation uses the data from the log [29] (http://www.cs.huji.ac.il/labs/parallel/workload/l_ricc/index.html). This log contains several months accounting records from RIKEN Integrated Cluster of Clusters. It has 4 clusters, and begins running since August 2009. The system has 1024 nodes, each with 12 GB of memory and two 4-core CPUs, for a total of 12 TB memory and 8192 cores. We delete those jobs that are not executed in the log in the simulation. The simulation results are shown in Table 3. The column 5 of the log is the parallelism of the jobs. We find that ZOMT always has the smallest value in *ART*, *AWT* and *SDWT*. We also find that Fair also has a relatively good performance

in the log. We give a comparison from the above section 5.1~5.3. We find the result in **Table 3** has the same pattern when the arrival rate is 5 and q equals to 0.5. The reason is that the log comes from a Cluster which is not very busy.

Table 3. Comparisons of different parameters

	AFCFS (e+5)	LJFS (e+5)	Fair (e+5)	ZOMT (e+5)
ART	1.3418	1.3416	1.3080	1.3012
AWT	1.2297	1.2295	1.1999	1.1816
SDWT	1.0214	1.0148	1.0205	0.8145

6. Conclusion

In this paper, we propose a new scheduling method called ZOMT to schedule the job with different parallelism. We check the performance of our proposed ZOMT in *AWT*, *AWWT*, *ART*, *AWRT*, *SDWT* and *SDWWT*. Simulation resulting from different aspects proves that ZOMT outperforms AFCFS, LJFS and Fair for the parameters: *AWT*, *AWWT*, *ART* and *AWRT*. Even though there are some VMs that need to be imgrated, ZOMT still has a good performance.

In the Cloud, the VM can initiate and stop at any time. Sometimes, the VM even can migrate from different places, and this may bring new challenges to the scheduling of parallel jobs. We hope that we can extend our method to the condition of the migrations of VMs in the Cloud. Considering the deadline of different parallel tasks, we also hope we can evaluate some heuristics [30] in the future. We also hope we can evaluate our method on some simulation platforms, such as Gridsim[31] or Cloudsim[32]. Energy consumption is also a hot topic in the scheduling. N. Cordeschi et al. [15, 16] have proposed some methods to save energy consumption in networked data centers. Though this paper does not take account of the energy consumption, and we only take account of the execution time and the fairness, we also hope that we can evaluate the energy consumption in the future. Because of the non-linear speedup of parallel tasks and the dynamic of the Cloud resources, saving energy consumption is more difficult. We hope we can propose and evaluate more methods to minimize the energy consumption in some special parallel tasks such as meteorological parallel tasks [33].

Acknowledgments

The work was partly supported by the National Natural Science Foundation of China (NSF) under grant (NO. 41475089, NO. 71673145), and Open Fund Project (No. NSS1403) of State International S&T Cooperation Base of Networked Supporting Software, Jiangxi Normal University.

References

[1] Brandic, I. and R. Buyya, "Special section: Recent advances in utility and cloud computing," *Future Generation Computer Systems*, 28(1): 36-38, 2012. Article (CrossRef Link).

[2] X. Liu, Y. Zha, Q. Yin, Y. Peng, L. Qin, "Scheduling parallel jobs with tentative runs and consolidation in the cloud," *Journal of Systems and Software*, Volume 104, Pages 141-151, ISSN 0164-1212, June 2015. Article (CrossRef Link).

[3] F. Zhang, J. Cao, K. Li, S. U. Khan, K. Hwang, "Multi-objective scheduling of many tasks in cloud platforms," *Future Generation Computer Systems*, Volume 37, Pages 309-320, ISSN 0167-739X, , July 2014. Article (CrossRef Link).

[4] Y. Laili, F. Tao, L. Zhang, Y. Cheng, Y. Luo, B. R. Sarker, “A Ranking Chaos Algorithm for dual scheduling of cloud service and computing resource in private cloud,” *Computers in Industry*, Volume 64, Issue 4, Pages 448-463, ISSN 0166-3615, May 2013. Article (CrossRef Link).
[5] W. Li, J. Wu, Q. Zhang, K. Hu, J. Li, “Trust-driven and QoS demand clustering analysis based cloud workflow scheduling strategies,” *Cluster Computing*, Volume 17, Issue 3, 1013-1030, 2014. Article (CrossRef Link).
[6] F. Satoh, H. Yanagisawa, H. Takahashi, T. Kushida, "Total Energy Management System for Cloud Computing," in *Proc. of 2013 IEEE International Conference on Cloud Engineering* (IC2E), pp.233, 240, 25-27 March 2013. Article (CrossRef Link).
[7] A. Alnowiser, E. Aldhahri, A. Alahmadi, M. M. Zhu, "Enhanced Weighted Round Robin (EWRR) with DVFS Technology in Cloud Energy-Aware," in *Proc. of* 2014 *International Conference on Computational Science and Computational Intelligence (CSCI)*, pp.320,326, 10-13 March 2014. Article (CrossRef Link).
[8] R. Martin, L. David, A. Taleb-Bendiab, "A Comparative Study into Distributed Load Balancing Algorithms for Cloud Computing,” in *Proc. of waina, 2010 IEEE 24th International Conference on Advanced Information Networking and Applications Workshops*, 551-556, 2010. Article (CrossRef Link).
[9] S. Nakrani, C. Tovey, “On Honey Bees and Dynamic Server Allocation in Internet Hosting Centers,” *Adaptive Behavior*, 12, pp: 223-240, 2004. Article (CrossRef Link).
[10] Z. Papazachos, H. Karatza, “The impact of task service time variability on Gang scheduling performance in a two-cluster system,” *Simulation Modelling Practice and Theory*, 17:1276–1289. Article (CrossRef Link).
[11] H. Karatza, “Performance of Gang scheduling policies in the presence of critical sporadic jobs in distributed systems,” in *Proc. of symp perform evaluation of comp telecommun syst 2007*, San Diego, CA, pp. 547–554, 2007. Article (CrossRef Link).
[12] H. Karatza, “Performance of Gang scheduling methods in a parallel system,” *Simulation Modelling Practice and Theory*, 17:430–441. Article (CrossRef Link).
[13] Z. C. Papazachos, H. Karatza, "Gang scheduling in multi-core clusters implementing migrations," *Future Generation Computer Systems*, 27(8): 1153-1165. Article (CrossRef Link).
[14] I. Moschakis, H. Karatza, "Evaluation of gang scheduling performance and cost in a cloud computing system," *The Journal of Supercomputing*, 59(2): 975-992. Article (CrossRef Link).
[15] N. Cordeschi, M. Shojafar, E. Baccarelli, "Energy-saving self-configuring networked data centers," *Computer Networks*, 57.17, 3479-3491, 2013. Article (CrossRef Link).
[16] N. Cordeschi, M. Shojafar, D. Amendola, E. Baccarelli, "Energy-efficient adaptive networked datacenters for the QoS support of real-time applications," *The Journal of Supercomputing*, 71.2, 448-478, 2015. Article (CrossRef Link).
[17] L. Liu, G. Xie, L. Yang, R. Li, “Schedule Dynamic Multiple Parallel Jobs with Precedence-Constrained Tasks on Heterogeneous Distributed Computing Systems,” in *Proc. of* 2015 14*th International Symposium on Parallel and Distributed Computing* (*ISPDC*), pp.130 – 137, June 29 2015-July 2 2015. Article (CrossRef Link).
[18] S. Prabhakaran, M. Neumann, S. Rinke, F. Wolf, “A Batch System with Efficient Adaptive Scheduling for Malleable and Evolving Applications,” in *Proc. of 2015 IEEE International Parallel and Distributed Processing Symposium* (*IPDPS*), 25-29, Page(s):429 - 438, May 2015. Article (CrossRef Link).
[19] K. Oh-Heum, C. Kyung-Yong, “Scheduling parallel tasks with individual deadlines,” *Theoretical Computer Science*, Volume 215, Issues 1–2, Pages 209-223, ISSN 0304-3975, 28 February 1999. Article (CrossRef Link).
[20] J. G. Barbosa, B. Moreira, “Dynamic scheduling of a batch of parallel task jobs on heterogeneous clusters, *Parallel Computing*,” Volume 37, Issue 8, Pages 428-438, ISSN 0167-8191, August 2011. Article (CrossRef Link).
[21] W. Wang,Y. Chang,W. Lo,Y. Lee, “Adaptive scheduling for parallel tasks with QoS satisfaction for hybrid cloud environments,” *The Journal of Supercomputing*, Volume 66, Issue 2, pp 783-811, 2013. Article (CrossRef Link).

[22] T. He, S. Chen, H. Kim, L. Tong, K. Lee, “Scheduling Parallel Tasks onto Opportunistically Available Cloud Resources,” in *Proc. of 2012 IEEE 5th International Conference on Cloud Computing* (*CLOUD*), Page(s): 180 – 187, 2012. Article (CrossRef Link).
[23] D. Oliveira, K. A. C. S. Ocaña, F. Baião, M. Mattoso, “A Provenance-based Adaptive Scheduling Heuristic for Parallel Scientific Workflows in Clouds,” *Journal of Grid Computing*, Volume 10, Issue 3, pp 521-552, 2012. Article (CrossRef Link).
[24] Y. Hao, “Enhanced resource scheduling in Grid considering overload of different attributes,” *KSII Transactions on Internet and Information Systems*, Vol.10 No.3, 1071-1090, 2016. Article (CrossRef Link).
[25] M. S. Squillante, F. Wang, M. Papaefthymiou, "Stochastic analysis of gang scheduling in parallel and distributed systems," *Performance Evaluation*, 27–28(0): 273-296, 1996. Article (CrossRef Link).
[26] C. L. Morefield, "Application of 0-1 integer programming to multi-goal tracking problems," *IEEE Transactions on Automatic Control*, 22(3): 302-312, 1977. Article (CrossRef Link).
[27] M. Zaharia, D. Borthakur, J. Sen Sarma, K. Elmeleegy, S. Shenker, I. Stoica, “Delay Scheduling: A Simple Technique for Achieving Locality and Fairness in Cluster Scheduling,” *In EuroSys 10*, 2010. Article (CrossRef Link).
[28] Amazon Web Services LLC, Amazon elastic compute cloud (EC2), 2009. Article (CrossRef Link).
[29] Y. Hao, G. Liu, R. Hou, Yongsheng Zhu, Junwen Lu. “Performance Analysis of Gang Scheduling in a Grid,” *Journal of the Network and Systems Management*, Volume 23, Issue 3, pp 650-672, July 2015. Article (CrossRef Link).
[30] Y. Hao, G. Liu, “An Evaluation of Nine Heuristic Algorithms with Data-intensive Jobs and Computing-intensive Jobs in a Dynamic Environment,” *IET software*, Value 9, No. 1, Page 7-16, 2015. Article (CrossRef Link)
[31] Y. Hao, G. Liu, N. Wen, “An enhanced load balancing mechanism based on deadline control on GridSim,” *Future Generation Computer Systems*, Volume 28, Issue 4, Pages 657-665, ISSN 0167-739X, April 2012. Article (CrossRef Link).
[32] F. Ramezani, J. Lu, J. Taheri, F. K. Hussain, “Evolutionary algorithm-based multi-objective task scheduling optimization model in cloud environments,” *World Wide Web-internet & Web Information Systems*, 18(6), 1737-1757, 2015. Article (CrossRef Link).
[33] Y. Hao, L. Wang, M. Zheng, “An adaptive algorithm for scheduling parallel jobs in meteorological Cloud,” *Knowledge-Based Systems*, Volume 98, Pages 226-240, ISSN 0950-7051, 15 April 2016. Article (CrossRef Link).

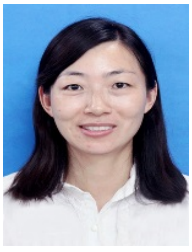
Yongsheng Hao received his MS Degree of Engineering from Qingdao University in 2008. Now, he is an engineer of Information management department, Nanjing University of Information Science & Technology. His current research interests include distributed and parallel computing, mobile computing, Grid computing, web Service, particle swarm optimization algorithm and genetic algorithm. He has published 16 papers in international conferences and journals.

Mendan Xia received her MS Degree of Arts from Nanjing Normal University in 2006. Now, she is an English teacher in School of Languages and Cultures, Nanjing University of Information Science & Technology. Her current research interests include English Language Teaching and Cross-cultural Communication.

Na Wen receives her Ph.D. degree and MS degree from Ocean University of China, P.R. China in 2009 and in 2006 respectively. Now, she is a researcher of college ofAtmospheric Science, Nanjing University of Information Science & Technology. His current research focuses on meteorology research.

Rongtao Hou received his PHD Degree of Computer science from Northeastern University in 2001. Now, he is a professor of School of computer and software, Nanjing University of Information Science & Technology. His current research interests include distributed and parallel computing, mobile computing, weather forecast model and so on.

Hua Deng received his MS Degree of Engineering from NUIST in 2005. Now, he is an engineer of College of Atmospherics Science, Nanjing University of Information Science & Technology. His current research interests include distributed and parallel computing, weather model, supercomputing.



Lina Wang received the B.S. degree in computer application from Nanjing Institute of Meteorology, Jiangsu, China in 2001 and the M.S. degree in system analysis and integration from Nanjing University of Information Science and Technology, Jiangsu, China in 2004. She received the Ph.D in computer application from Nanjing University of Aeronautics and Astronautics, Jiangsu, China in 2016. Currently, her main research interests are data mining and computer-network-technology.

Qin Wang received his MS Degree of Engineering from Nanjing normal university in 2005. Now, he is an engineer of Information management department, Nanjing University of Information Science & Technology. His current research interests mainly focus on the resource scheduling on different platforms.