Enhanced resource scheduling in Grid considering overload of different attributes

Yongsheng Hao[1,2]
[1] Network Center, Nanjing University of Information Science & Technology, Nanjing, 210044, China
[2] State International S&T Cooperation Base of Networked Supporting Software, Jiangxi Normal University, 330022, China
[e-mail: yongshenghao@yahoo.com]
*Corresponding author: Yongsheng Hao

Received November 4, 2015; accepted January 17, 2016; published March 31, 2016

Abstract

Most of scheduling methods in the Grid only consider one special attribute of the resource or one aspect of QoS (Quality of Service) of the job. In this paper, we focus on the problem that how to consider two aspects simultaneously. Based on the requirements of the jobs and the attributes of the resources, jobs are categorized into three kinds: CPU-overload, memory-overload, and bandwidth-overload jobs. One job may belong to different kinds according to different attributes. We schedule the jobs in different categories in different orders, and then propose a scheduling method-MTS (multiple attributes scheduling method) to schedule Grid resources. Based on the comparisons between our method, Min-min, ASJS (Adaptive Scoring Job Scheduling), and MRS (Multi-dimensional Scheduling) show: (1) MTS reduces the execution time more than 15% to other methods, (2) MTS improves the number of the finished jobs before the deadlines of the jobs, and (3) MTS enhances the file size of transmitted files (input files and output files) and improves the number of the instructions of the finished jobs.

Keywords: multiple attributes, resource scheduling, requirements to QoS, overload

1. Introduction

In the recent years, a lot of studies on resource scheduling in the Grid have been conducted in the past years [1-9]. The attributes of a resource include: the processing ability of the resources [1-3], memory capacity [8], network bandwidth [9] and so on. Most of researches are based on one special attribute of the resource.

Traditional Grid resource scheduling algorithms mainly focus on the CPU speed, such as min-min [1] and max-min algorithm [1]. More examples can be found in [2-6]. Some other methods also pay attention to the special attributes of the resource like the memory management [8], network bandwidth [9], I/O management [10] and so on. Only some of them consider the scheduling methods from the view of multiple attributes of the Grid resource [11-13]. At the same time, some researchers also propose new methods based on the condition of the entire Grid [12]. But, to the best of our knowledge, only a few methods consider the attributes of resources and the requirements of the jobs at the same time. There are two kinds of resources in the Grid: time-shared [14] and space-shared. Our paper pays attention to the space-shared resource. The Grid has many resources which can be provided to some jobs in the quantity and quality, but for other jobs, the Grid only has a few resources that can ensure those jobs be executed as their requests. Developing an efficient scheduling method becomes a challenge problem when we consider the two aspects simultaneously. This is the problem that we aim to solve in the paper.

The main contributions of this paper include: (1) we analyze the requirement of all jobs and the condition of the Grid, and then (2) we give a new classification of jobs: CPU-overload, memory-overload, and bandwidth-overload; (3) we propose a scheduling method which considers the attributes of the resource and the requirement of the job. One job may belong to different kinds of jobs according to the requirement of the job. Jobs belonging to the three kinds, two of the three kinds, and one of the three kinds, are assigned to different sets: *ThreeOL*, *TwoOL*, and *OneOL*, respectively. The leaving jobs are set in *OtherL*. More and more resources can satisfy the requirement of the jobs in different sets according to the order: *ThreeOL*, *TwoOL*, *OneOL* and *OtherL*. We schedule the job as the sequence of *ThreeOL*, *TwoOL*, *OneOL* and *OtherL*. After the *ThreeOL* is empty, we compute the four sets again and schedule non-empty sets again until the *ThreeOL* is empty. The same method is used for *TwoOL*, *OneOL* and *OtherL*. For the job belonging to the same set, we propose a new method-MTS to schedule them. Simulations are executed to check the performance of different methods. Simulation results prove that our method has good performance not only in improving the number of finished jobs, but also in the reducing the execution time of the finished jobs.

This paper is structured as follows. In section 2, we introduce related work. Section 3 provides an analysis of the attributes of the job and the Grid resource. Last, we propose our algorithms based on the classifications of the jobs. Simulations are executed in section 4. Section 5 is the conclusion and future work.

2. Related Work

Most of the existing resources scheduling methods in the Grid focus on the processing ability of the resource. Others are offered from the views of other attributes of the resource and other requirements of the job, such as network [15]. Only some of them pay attention to multiple

attributes of the Grid resource and to multiple requirements of the job.

Most of traditional scheduling methods in the Grid aim to solve the problem only from the processing ability of the Grid resource. The Min-min scheduling algorithm [1, 14] sets the jobs that can be completed earliest with the highest priority. Each job always is assigned to the resource that can complete the job earlier. Similar to Min-min algorithm, Max-min algorithm [1, 14] sets the highest priority to the job that has the earliest completion time. Max-min can be used in cases where there are numerous shorter jobs than there are longer jobs. For example, if there is only one long task, Min-min will first execute many jobs with short execution time, and then execute the long task. Max-min will execute short jobs concurrently with the long job. Others scheduling methods pay attention to the processing ability including: best-fit [2-4, 7], AFCFS (Adaptive First Come First Service)[16], fastest-fit [4, 5, 7, 15], adaptive intelligent [15], and threshold-based adaptive intelligent algorithm.

Based on other attributes of Grid resources or the requirements of jobs, some researchers also have been conducted. Some researchers give different methods that consider the network of the Grid. Chinnaiah et al. [17] propose a Grid network monitoring architecture which is modeled by the Grid scheduler. Gao et al. conduct a systematic review of communication/networking technologies in Smart Grid [9], including communication/networking architecture, different communication technologies that would be employed into this architecture, quality of service (QoS), optimizing utilization of assets, control and management, etc. Sulistio et al. [15] use the GridSim simulation toolkit [23] (http://www.buyya.com/gridsim/) to explore the effect of network in Grid computing. Xhafa et al. [18] present useful computational models which focus on the design of efficient Grid schedulers using heuristic and meta-heuristic methods.

Multiple attributes of the Grid resource also have been valued in the scheduling. Schnauzer et al. [11] propose an auction mechanism for allocating and scheduling resources such as processors or storage devices which have multiple attributes. RCT (Resource Category Tree) [19] organizes resources based on their characteristics represented by primary attributes (PA). RCT adopts a structure of the distributed AVL tree, with each node being a specific range of PA values. Although it adopts a hierarchical structure, it does not require nodes in higher levels maintain more information than those in lower levels, which make RCT highly scalable. RCT has features such as self-organization, load-aware self-adaptation and fault tolerance. RCT not only supports range queries but also supports multi-attribute queries, which is very import in the scheduling of resources. An attribute based access control model in Grid environment is developed [20], which satisfies the dynamic and heterogeneity characters of the Grid environment and makes the system flexible and scalable for the use of XACML access control model. Some researchers emphasize the QoS management in Grid including the capacity, capability and properties of every resource [13]. Albodour et al. introduce Business Grid Quality of Service (BGQoS) model to solve the problem. They give the definition of the Grid resource in the shape of the number of CPU cores, memory in RAM, storage resource. Qureshi et al. study CPU, memory, and I/O-intensive job scheduling, and load balancing techniques [12], which is known as Mixed Task Load Balancing (MTLB) for Cluster of Workstation (CW) systems. Their proposed MTLB strategy, pre-jobs are assigned to each worker by the master to eliminate the worker's idle time. It employs Three Resources Consideration (TRC, including CPU, memory, and I/O) for load balancing. The authors give the definition of CPU, memory, and I/O-intensive jobs. Those definitions also can be utilized in the Grid. But they are only from the view of the jobs, and they don't give any attention to the attributes of the resource. An example is a system with high memory capacity; even memory-intensive job has no problems in the scheduling. ASJS (Adaptive Scoring Job

Scheduling) [21] gives different scores to the bandwidth and the CPU, and the different scores also have different weights depending on the ratio between the data-intensive jobs and the computing-intensive jobs. MRS (Multi-dimensional Scheduling) [22] uses virtual maps to efficiently determine a best fit of resources for jobs. It tries to use Resource Potential to identify inter-relations between resources such as bandwidth and data. The Resource Potential helps to find the resources that have the least execution overheads with respect to a job. Dissimilar to prior works, in this paper, we not only pay attention to the attributes of jobs (including CPU, memory, hard disk, deadline), but also pay attention to the resource supply and resource demand. We consider more attributes, so that our algorithm is closer to the true world scheduling of Grid resource. At the same time, we consider the supplement of the resources and the demand of the Grid users from different QoSs.

In this paper, we also try to study the case that every job has a deadline. To deal with the deadlines of jobs, many methods have been proposed. Hao et al. propose a method on GridSim, which tries to consider the urgency of the job and the resource fragment at the same time [23]. They schedule the jobs which have a short time to deadline first. And if a resource only has a small resource fragment, the resource will give all its calculating ability to the job. Malawski et al. try to maximize the number of user-prioritized workflows that can be completed under budget and deadline constraints on IaaS (Infrastructure as a Service) clouds [24]. They proposed two dynamic algorithms and one static algorithm and evaluated the performance of them. Zhao et al. present an algorithm on integrating mechanisms for deadline assurance into an optimized implementation of Actors [25]. They achieve this by using deadline-driven adaptive scheduling which takes a dynamically balances in their system. In the paper, we also try to study the scheduling method when every job has a deadline.

Researchers also propose the method of scheduling Grid resources from other aspects, such as workload scheduling [26], heuristic algorithm[27], resource reservation [28] and trust management [29]. Those methods are not directly related to our work, so we do not introduce them here.

3. Scheduling resource with multiple attributes in the Grid

In this section, first of all, we introduce the framework of the Grid that used in the paper [15, 23, 30]. Then, we give the classification of jobs based on our analysis. Last, we propose our algorithm of the resource scheduling.

3.1 Grid framework and scheduling method

Every layer of the Grid is explained in details in **Fig. 1**. Grid Broker schedules the Grid resources according to the scheduling methods. Each resource contains several machines and each machine contains several PEs (Processing Elements).

In **Fig. 1**, a Grid resource has some machines. For example, Resource 1 has t machines (Machine(1,1)~ Machine(1, t)), Machine(1,1) has n PEs (PE(1,1,1)~ PE(1,1, n)). The job (Gridlet in Gridsim, see [15, 23, 30]), Grid resource and Machine are defined as follows:

Job(*JobID*, *JobLength*, *JobFileSize*, *JobOutputSize*, *MaxMemory*, *t*, *sed*, *rid*);

Machine(*Machineid*, *numPE*, *ratingPE*);

Where

-*JobID* is the unique identifier of the job;

-*JobLength* is the number of instructions;

-*JobFileSize* is input file size of the job;

-*JobOutputSize* is output file size of the job;

-*MaxMemory* is the maximum of the memory the job needs;
-*t* is the deadline of the job;
-*sed* indicates the job whether has been finished;
-*rid* is the number of the resource that has been assigned to the job;
-*numPE* is the number of PE (Processing Entity) of the machine;
-*ratingPE* is the processing ability rating of the PE.

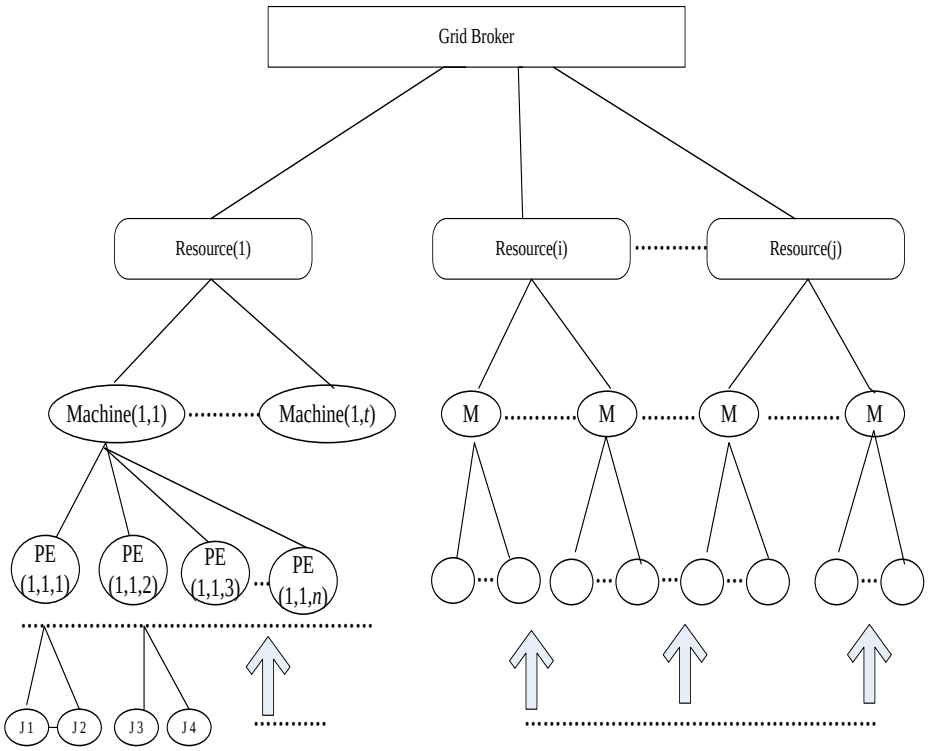

Fig. 1. The framework of three levels Grid

In our paper, based on our previous work [23] and other studies in [15, 30], we use the job, Machine, Grid resource to illustrate the entities that would be used in the scheduling. If the value of *sed* is equal to 1, the job has been finished; otherwise, it has not been finished yet.

All the jobs (j_{jtemp}) are listed in the job list *Jlist*.

$$Jlist = \{j_1, j_2, \ldots, j_{jtemp,\ldots}, j_{jend}\} \tag{1}$$

All the machines of the resource *r* are listed in the machine list *Ml*:

$$Ml_r = \{m_1^r, m_2^r, \ldots, m_{mtemp,\ldots}^r, m_{mend}^r\} \tag{2}$$

The speed of the resource *r* is defined as:

$$r.pa = \sum_{i=1}^{mend} (m_i^r.numPE \times m_i^r.ratPE) \tag{3}$$

$m_i^r.numPE$ is the number of PEs of the machine m_i^r and $m_i^r.ratPE$ is t he rating of the PE of m_i^r. It can be expressed in MIPS (Million Instructions per Second) or standard machine. All Grid resources (r_{rtemp}) are listed in the set *Rlist*:

$$Rlist = \{r_1, r_2, \ldots, r_{rtemp}, \ \ldots, r_{rend}\} \tag{4}$$

In this paper, we suppose that all the resources (for machines) are space-shared. In other words, a resource (machine) only has one job at a time.

When a job is allocated to a Grid resource, there are many methods to calculate the load of the resource. For example, Kobe et al. [31] develop a model for single-step-ahead CPU load prediction that can be utilized to predict the future CPU load in a dynamic environment. The prediction model is based on the control of multiple Local Adaptive Network-based Fuzzy Inference Systems Predictors (LAPs). Therefore, we can forecast the load of the Grid resource when a job is assigned to it. *CPUload* is the value of load for the CPU when the job *j* is assigned to the Grid resource *r*. If the value is more than ∂, the value of *j.CPUol* increases by 1. Algorithm 1 is the details of calculating the *j.CPUol* of every job.

Algorithm1: CountCPUOverloaded (Joblist *Jlist*, Resourcelist *Rlist*)

```
1. j.CPUol=0;
2. For every job j ∈ Jlist
3.    For every Resource r ∈ Rlist
4.       CPUload = l(j,r);
5.       If ( CPUload ≥ ∂ )
6.          j.CPUol=j.CPUol+1;
7.       Endif
8.    Endfor
9. Endfor
```

Same to *j.CPUol*, *j.Memol* and *j.Bwol* are the overload value of the memory and bandwidth. They have the same framework as the CPU used in Algorithm 1. The parameters are used in the calculations of *j.CPUol*, *j.Memol* and *j.Bwol* including:

CPUload, *Memload* and *Bwload* are the load of the resource *r* of the CPU, memory and bandwidth when the job *j* is assigned to the resource *r* respectively;

∂, β and δ are the upper bound number of resources from the CPU, memory, and bandwidth (related attributes) attributes which make the resource "over loaded";

l(*j*, *r*), *m*(*j*, *r*) and *n*(*j*, *r*) are the load of the CPU, memory, and bandwidth when the job *j* is assigned to the resource *r*, respectively.

Researchers have well explored the CPU, memory and I/O-intensive job scheduling, and load balancing techniques. But one of the main obstacles of the scheduling technique leads to the ignorance of jobs having a mixed requirement to resources. It is necessary that the scheduling methods developed for one kind of job are also effective for the other kinds of job. According to [12], Kalim et al. give the definition of CPU-intensive jobs, Memory-intensive jobs, I/O-intensive jobs and mixed jobs. In their model, a task is classified as a CPU intensive task, if the job contains arithmetic operators like addition, subtraction, multiplication, and division more than 40%. In the case of memory intensive jobs, assignment operators like equal and comparison operators are determined. If these operators occur more than 40% in the

forthcoming task, then the job is classified as a memory intensive task. Furthermore, files read and write operations are counted. If the job contains more than 40% file read and write operations, then the job is classified as an I/O-intensive task. The job is classified as a mixed task, if it does not belong to any of the above mentioned categories. Those definitions are not related to the resource. An example is a system with a high processing ability of every resource; even CPU-intensive job has no problems in scheduling. In our paper, taking account of the resources that would be used by the job, we classify jobs into different kinds according to the requirements of jobs and the attributes of the Grid resource.

(1) According to the requirement of the processing ability of the job

From Algorithm 1, we can get the number of resources which cannot satisfy the job *j*, and it is denoted by *j.CPUol*. If the value of *j.CPUol* is close to the number of the Grid resources, there are only a few Grid resources that can satisfy *j*. We set a boundary parameter λ to express the upper bound of the number of resources that which can ensure finishing the job before the deadline. If *j.CPUol* is more than $\lambda \times GRend$, we add *j* into *CPUul*, otherwise, we add *j* into *UnCPUul*.

$$CPUul = \{ j_t \mid j_t \in Jlist \wedge j_t.CPUol > \lambda \times rend \} \tag{5}$$

$$UnCPUul = \{ j_t \mid j_t \in Jlist \wedge j_t.CPUol <= \lambda \times rend \} \tag{6}$$

There are more resources that can ensure the jobs in *UnCPUul* can be completed than the resources that can ensure the jobs be completed in *CPUul*. In the same way, we can category jobs into different sets according to the requirement of the memory and the bandwidth. χ , η are the upper bound from the views of the memory and the bandwidth.

(2) According to the requirement of the memory of the job

$$Memul = \{ j_t \mid j_t \in Jlist \wedge j_t.Memol > \chi \times rend \} \tag{7}$$

$$UnMemul = \{ j_t \mid j_t \in Jlist \wedge j_t.Memol <= \chi \times rend \} \tag{8}$$

(3) According to the requirement of the bandwidth of the job

$$IOul = \{ j_t \mid j_t \in Jlist \wedge j_t.Bwol > \eta \times rend \} \tag{9}$$

$$UnIOul = \{ j_t \mid j_t \in Jlist \wedge j_t.Bwol <= \eta \times rend \} \tag{10}$$

From the three kinds of categories, we list the jobs belonging to *CPUul*, *Memul* and *IOul* in *ThreeOL*, list the jobs belonging to two of them in *TwoOL*, and the jobs belonging to one of them in *OneOL*. Others jobs are listed in *OtherL*.

$$ThreeOL = CPUul \cap Memul \cap IOul \tag{11}$$

$$\begin{aligned} TwoOL = {} & (CPUul \cap Memul \cap UnIOul) \cup (CPUul \cap UnMemul \cap IOul) \\ & \cup (UnCPUul \cap Memul \cap IOul) \end{aligned} \tag{12}$$

$$\begin{aligned} OneOL = {} & (CPUul \cap UnMemul \cap UnIOul) \cup (UnCPUul \cap UnMemul \cap IOul) \\ & \cup (UnCPUul \cap Memul \cap UnIOul) \end{aligned} \tag{13}$$

$$OtherL = Jlist - ThreeOL - TwoOL - OneOL \tag{14}$$

The scheduling method faces two below aspects:

(1) selecting which jobs for the first assignment, and (2) selecting which Grid resource to the job.

From above analysis, we know that only a few Grid resources that can meet the requirements of the jobs in *ThreeOL*. So, first of all, we schedule the jobs in *ThreeOL*, and then select the jobs in *TwoOL*, *OneOL*, and *OtherL* in sequence. *j.Toverload* is the total value of the overloads of different attributes.

$$j.Toverload = j.CPUol + j.Bwol + j.Memol \tag{15}$$

The algorithm is listed as follows:

Algorithm 2: SelectJobList( )

```
1.  CountCPUOverloaded(Joblist Jlist, Resourcelist Rlist);
2.  CountMemoryoverloaded(Joblist Jlist, Resourcelist Rlist);
3.  CountBandwidthoverloaded(Joblist Jlist, Resourcelist Rlist);
4.  Get ThreeOL as Formula (11);
5.  Get TwoOL as Formula (12);
6.  Get OneOL as Formula (13);
7.  Get OtherL as Formula (14);
8.  While (lengh(ThreeOL)!=0)
9.        Scheduling(ThreeOL);
10.       SelectJobList( );
11. Endwhile
12. While (lengh(TwoOL)!=0)& (lengh(ThreeOL)= =0)
13.       Scheduling(TwoOL);
14.       SelectJobList( );
15. Endwhile
16. While (lengh(OneOL)!=0) & (lengh(TwoOL)= =0) & (lengh(ThreeOL)= =0)
17.      Scheduling(OneOL);
18.      SelectJobList( );
19. Endwhile
20. While (lengh(OtherL)!=0) & (lengh(OneOL)= =0)& (lengh(TwoOL)= =0)& (lengh(ThreeOL)=
    =0)
21.      Scheduling(OtherL);
22.      SelectJobList( );
23. Endwhile
```

Lines 1-3 of algorithm 2 calculate the value of *j.CPUol*, *j.Memol* and *j.Bwol*. Lines 4-7 calculate of *ThereOL*, *TwoOL*, *OneOL* and *OtherL*. Lines 8-11, lines 12-15, lines 16-19, and lines 20-23 are the scheduling of *ThereOL*, *TwoOL*, *OneOL* and *OtherL*, respectively. We can use the scheduling method that we have discussed in Section 2. And we will introduce a new method in the following section 3.2 (Scheduling( ) in Algorithm 3). Our method is a depth-first search algorithm: first of all, we schedule the resource in *ThereOL*, then, we calculate the four set again, and we schedule the resource in *ThereOL* again, repeat the operation until no jobs in the set *ThereOL*; same to other three sets.

3.2 Enhance Grid scheduling with multiple attributes

The second step is to allocate a Grid resource to a job. There are many methods in scheduling, such as fastest-first, best-fit, static intelligent, ASJS [21] and et al.

Algorithm 3 depicts the detailed information of our scheduling. First of all, we need to investigate the Grid resource *r* whether satisfies the requirement of the job *j*. *crq*(*j*, *r*) checks the requirement, if *r* can satisfy *j*, it returns true; otherwise, it returns false. The requirements include CPU, memory, bandwidth, and the deadline of the job. The *r* must promise that *j* can be finished before the deadline. *sumtime* is the execution time of *j* and it must be less than the value of its deadline. It also includes the waiting time.

Algorithm 3: Scheduling(Joblist *Jlist*)

```
1.  minvalue=+∞;
2.  selectj=null;
3.  For every resource r in Rlist;
4.     For every job j in Jlist;
5.        If crq(j, r)
6.           levfra = +∞;
7.           If ∂−l(j,r) ≤ minfrac and β−m(j,r) ≤ minfram and δ−n(j,r) ≤ minfrab
8.              levfratemp = ∂−l(j,r) + β−m(j,r) + δ−ℵ(j,r);
9.              If levfratemp < levfra
10.                levfra = levfratemp;
11.                selectfr=r;
12.                selectfj=j;
13.             EndIf
14.          Endif
15.          If levfra!=+∞;
16.             temp=mt(j, r);
17.                if (temp<minvlue);
18.                   minvlue= temp;
19.                   selectr=r;
20.                   selectj=j;
21.                Endif
22.             Endif
23.          Endif
24.     Endfor
25.  Endfor
26.  If (Selectfr!=null)
27.     Assign selectfj to selectfr, delect selectfj for Jlist;
28.  Else
29.     Assign selectj to selectr, delect selectj for Jlist;
30.  Endif
```

Line 5 of algorithm 3 (*crq*(*j*, *r*)) checks whether the attributes of the Grid resource *r* can satisfy the requirements of the job. Our scheduling algorithm has two considering aspects: the resource fragment and other parameter related to executing time and the value of over-load of different attributes.

Lines 6-14 is the first aspect. *levfra* records the minimum value of the resource fragment of different parameters. *minfrac*, *minfram* and *minfrab* are the minimums of resource fragment

for different attributes(CPU, memory, bandwidth). Resource fragment is the leaving value to the request of the user. Suppose the request of memory is 4G, the request of memory is 3G, then from the view of memory, the resource fragment is 1G. If a resource has 4G memory, and a job needs 3.9G memory, then the leaving resource fragment of memory is 0.1G (4-3.9), and if the system sets the smallest fragment of memory to 0.2G (4*5%), the resource satisfies the job (According to line 7 in Algorithm 3). If the values of load ($l(j, r)$, $m(j, r)$ and $n(j, r)$) are less than those parameters, the job will get the resource. If there are multiple resources meet the requirement, we select the job and the resource which has the minimum value of resource fragments (Lines 9-13).

Lines 16-21 is the second aspect. *temp* is a parameter when j is assigned to r. The value of *temp* is decided by j and r. The attributes of r and the requirements of j include the processing ability, the memory capacity, and the hard disk capacity and so on. In MTS (multiple attributes scheduling method) (Algorithm 3) algorithm, we give two rules for the second aspect:

(1) Scheduling the job that which only has a few resources can execute it first.

(2) Job with a lower value of execution time scheduled first.

So, we set

$$mt(j,r) = \frac{f(j,r)}{e^{j.CPUol} + e^{j.Bwol} + e^{j.Memol.}} \tag{16}$$

$f(j,r)$ is the execution time of j when j is assigned to r. Our target is to find the resource-job mapping set which makes $mt(j,r)$ gets the minimum value. When the number of the resource in overload state is the same, the job with the minimum execution time will be executed firstly; when the execution times of the jobs are the same, the jobs with a larger value of over-load will be executed firstly. At the same time, with the dropping of the number of resources which ensure the jobs be finished as the request, the values of overload increases gradually, the job has more possibility to be executed.

Lines 26-30 is the scheduling. First of all, we schedule the resource from the aspect of the resource fragment of different attributes (line 27) and then we schedule from the second aspect (line 29).

4. Experimental Classification Results and Analysis

In this section, we introduce the simulation environment in section 4.1, section 4.2 gives the performance metrics of simulations, section 4.3 discusses the simulation results of the different cases, section 4.4 is the analysis of time complexity and simulation about the computation time of the scheduling method, and section 4.5 gives a simulation result based on a true log.

4.1 Enhance Grid scheduling with multiple attributes

Previous works show that: (1) Min-min has good performance under most of cases; (2) ASJS [21] gives attentions to multiple attributes of the Grid; (3) MRS [22] tries to determine a best fit of resources for the job according to the computing capacity and bandwidth of the resource. So, in the experiment, we compare our work MTS with Min-min, ASJS, and MRS. In ASJS, users can submit different types of jobs at the same time containing computing-intensive jobs or data-intensive jobs. The computing-intensive job means that jobs need lots of computing power to complete it. In addition, the data-intensive job means that the resource has to take a

lot of bandwidth to transmit files. ASJS gives different weights to different attributes according to the ratio between different jobs. In our simulation, the processing time of computing-intensive job is decided by the processing ability of the resource, and the processing time of data-intensive is the sum of processing time of the resource and the time of transmitting the input file and output file. The score of a resource is composed of the score of the processing ability (α) and the score of the bandwidth (β). Every part has separate weights. According to [21], the parameters of ASJS are selected as below (**Table 1**):

Table 1. Parameters for ASJS

Data-intensive/Computing-intensive	3:7 (case 1)	1:1 (case 2)	7:3 (case 3)
α	0.3	0.5	0.7
β	0.7	0.5	0.3

Table 2. The information about the job

Requirement	Data-intensive	Computing-intensive
Memory(G)	[1, 5]	[1, 5]
Instructions(PEs)	[1,11]	[1,11]
Deadline(time unit)	[1, 5]	[1, 5]
Filesize(100M)	[5, 15]	0
Bandwidth(100M/s)	[1, 5]	[1, 5]

Table 3. The attributes of the Grid resource

Attributes	Grid Resource
CPU (PEs Per. time unit)	[5,15]
Memory (G)	[1, 5]
Bandwidth (100M/s)	[5, 15]

We conduct the simulation 10 times and **Fig. 2**, **Fig. 3**, **Fig. 4**, and **Fig. 5** are the average values of those metrics. All the resources in the Grid are space-shared. This means only one job on a resource (PE) at a time.

We suppose that the functions of this paper are defined as follows:

$$j.sfs=j.JobFileSize+j.JobOutputSize \tag{17}$$

$$j.trftime=\frac{j.sfs}{r.bandwidth\times\delta}$$

$$j.sumtime=j.trftime+\frac{j.JobLength}{r.pa\times\partial} \tag{18}$$

$$j.CPUload=\frac{j.JobLength/\ j.sumtime}{r.pa\times\partial} \tag{19}$$

$$j.Memload=\frac{j.maxmemory}{r.memory\times\beta} \tag{20}$$

$$j.Bwload = \frac{j.sfs}{r.bandwidth \times j.trftime \times \delta} \tag{21}$$

$j.sfs$ is the total filesize of the input files ($j.JobFileSize$) and output files ($j.JobOutputSize$).

The *sumtime* of the job *j* includes the time of finished the instructions and the time of files transforming time (*j.trftime*). If the value of *j.CPUload* is greater than 1, the resource *r* cannot meet the requirement of the job *j*. Same as the value of *j.Memload* and *j.Bwload*, we set the upper bound of $\partial = 1, \beta = 1, \delta = 1$ in the simulation. Namely, the Grid resource (machine) can give all of its ability to the job. In fact, the user can give the values of them. *minfrac*, *minfram* and *minfrab* are set to 0.05 in the simulations which is the samllest resource fragement about different attributes.

In our simulation, we randomly generate jobs, though every simulation experiment yields roughly the same result. The parameters in **Table 4** are used to investigate the performance of our method. In the simulation, if the requirement of a job to the memory and bandwidth is satisfied, the executing time of the job is decided by the number of instructions of the job, the input and output file size of the job, the processing ability and the bandwidth of the Grid resource. If the requirement is not to be satisfied, the job would not be finished on the resource.

The information of the jobs and the resource is listed in **Table 2** and **Table 3** respectively. They are generated randomly and their scopes are shown in **Table 2** and **Table 3** respectively as well. The number of instructions of every job is between 1 and 11 grade (1 million instructions are assigned grade 1, 2 million instructions are assigned grade 2, etc.). Since the same calculation ability of every PE (1 Million Instructions Per. second), we can express the number of the instructions of the job in PEs (1 million instructions or standard machine). Every job has a deadline with a range of [1, 5] time units. Every job length is between 1 and 11 grade (1 Million Instructions is assigned to grade 1; 2 Million Instructions is assigned to grade 2, etc.). The processing ability (CPU) of every Grid resource is a random between 5 and 15 (1 PE Per time unit is assigned to grade 1, 2 PEs Per time unit is assigned to grade 2, etc.). There are 120 resources in the system. The total number of jobs is 100000 and the deadline is a random number between 1 and 5 (see **Table 2** and **Table 3**).

We utilized Matlab language to implement these algorithms and ran them on an Intel CORE, 2.66 Ghz, 4 GB RAM desktop PC with 100 MB/s Ethernet card, Window 8.

4.2 Performance metrics

Table 4. Notations used in the simulation

Name	**Meaning**
AVE	Average execution time
AAR	Average arrival rate
FJB	The number of finished jobs
UFJB	The number of unfinished jobs
SFS	The sum file size of finished jobs
SIN	The total number of instructions of finished jobs
q	The ratio of data-intensive job to all jobs

AVE is the average execution time of the finished jobs, it includes the waiting time:

$$AVE = \frac{\sum_{i=1}^{jend} (j_i.endtime * j_i.sed)}{\sum_{i=1}^{gend} (j_i.sed)} \tag{22}$$

FJB and *UFJ* are the number of the finished jobs and the unfinished jobs.

$$FJB = \sum_{i=1}^{jend} (j_i.sed) \tag{23}$$

SFS is the total file size of finished jobs and *SIN* is the total number of instructions of the finished jobs. The metrics used in the paper are listed in **Table 4**.

4.3 Simulation results

Fig. 2, **Fig. 3**, **Fig. 4** and **Fig. 5** are the simulation results when the ratios of data intensive job (*q*) are 0.3, 0.5 and 0.7.

Fig. 2 depicts the average execution time (*AVE*) of Min-min, ASJS, MRS, and MTS when the value of *q* is 0.3, 0.5, and 0.7 respectively. The *AME* of MTS is always less than that of Min-min, ASJS, and MRS. *AVE* of MTS average reduces 26.85%, 33.97%, and 17.31% to Min-min, ASJS and MRS in *AVE*. MTS has the lowest *AVE* under all the cases, especially when the system has a higher arrival rate (*AAR*>80). The reasons are: (1) MTS schedules a job to the resource which has the smallest resource fragment; (2) MTS ensures that the short job be scheduled first; (3) MTS takes account of the requirements of a job to different attributes. Though Min-min ensures every job has the lowest execution time, it does not consider the influence of the scheduling of the prior jobs on the rest jobs. ASJS always schedules jobs to the highest score, and thus the job needs waiting time. So ASJS also has a higher *AVE*. In fact, same to ASJS, MRS also assigns jobs to the highest computation index, which also does not take into account some jobs that only have a few resources that can ensure them can be completed as the request. So, the *AVEs* of three methods are more than the value of MTS.

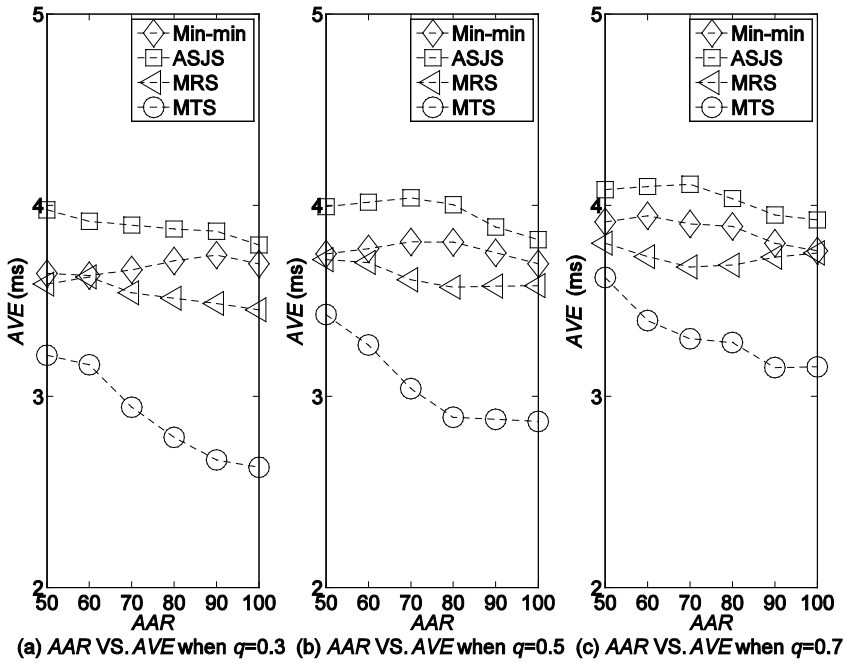

Fig. 2. *AVE* when (a) *q*=0.3, (b) *q*=0.5 and (c) *q*=0.7

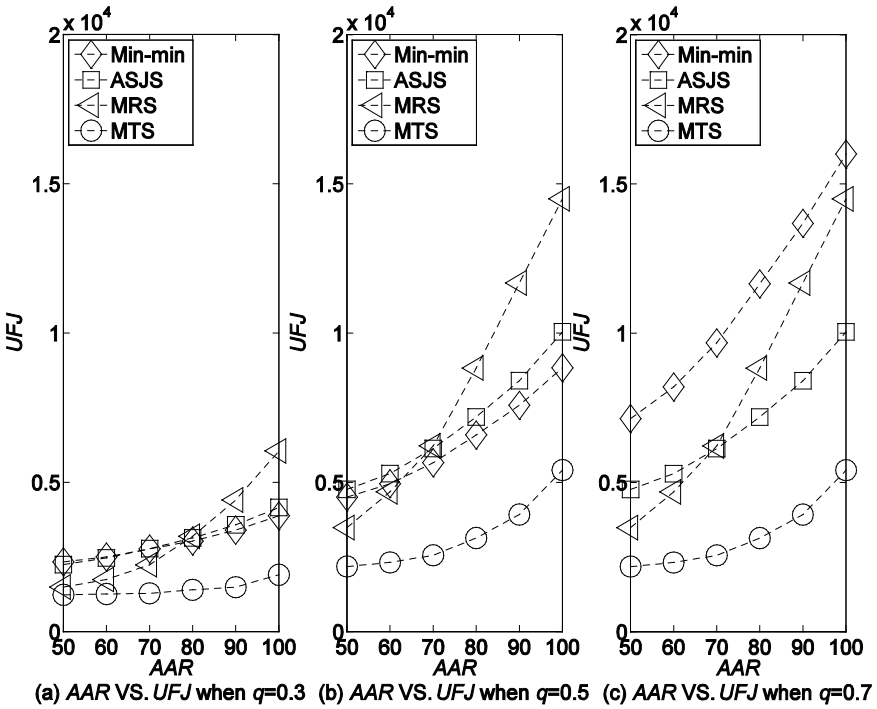

Fig. 3. *UFJ* when (a) q=0.3, (b) q=0.5 and (c) q=0.7

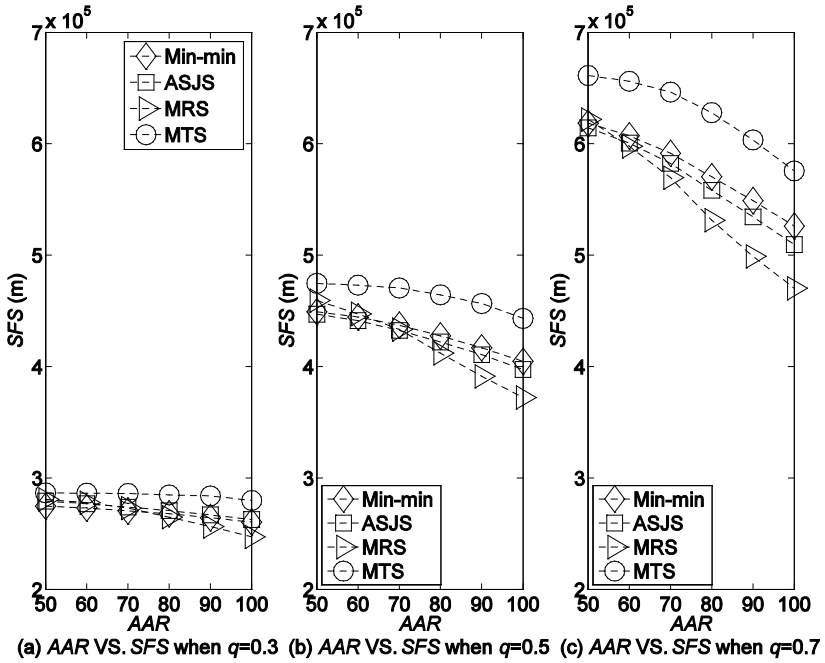

Fig. 4. *SFS* when (a) q=0.3, (b) q=0.5 and (c) q=0.7

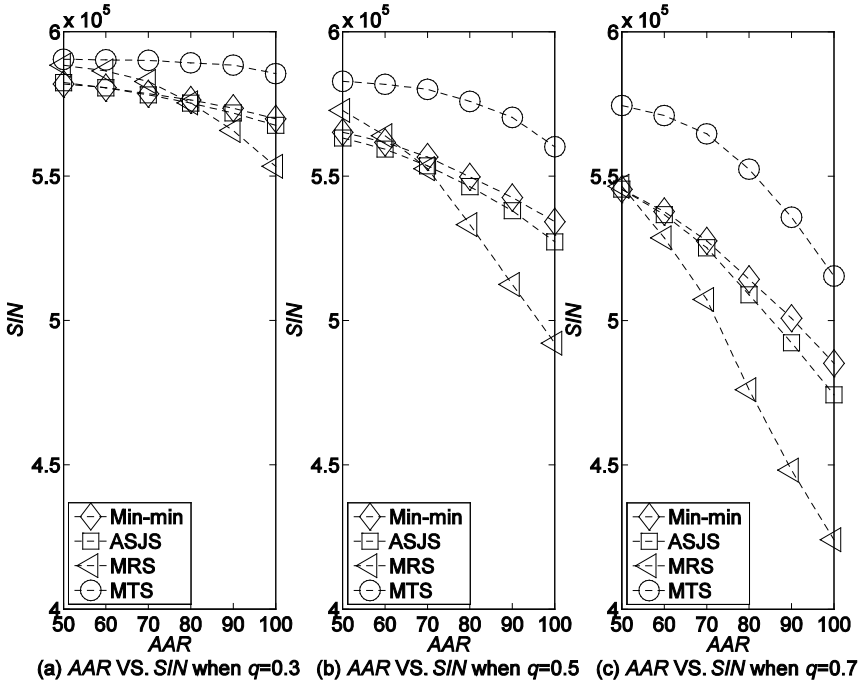

Fig. 5. *SIN* when (a) q=0.3, (b) q=0.5 and (c) q=0.7

Fig. 3 is the *UFJ* of the four methods. All the values of *UFJ* of the three methods increase with the increase of the arrival rate. Those figures show *UFJ* of MTS is always less than the value of ASJS and Min-min. MTS finishes more jobs than ASJS under the equivalent condition shown in **Fig. 3**, as MTS takes into account the system state, it works well in reducing the value of *UFJ*. MTS has the lowest value in *UFJ*. To Min-min, ASJS and MRS, MTS average reduces 3061, 3701, 5122 in *UFJ*, and average reduces 4.23%, 5.11% and 7.08% respectively. MRS has a good performance when the arrival rate has a low value. With the increase of the arrival rate, MRS loses its advantage to Min-min and ASJS. This is because that the computation indexes used in MRS of all the job-resource mappings are closely to each other. Thus it is very difficult for MRS to make the best decision of the scheduling. ASJS always gives priority to the best resource (the highest computing ability, the highest bandwidth, the highest memory capacity), so it makes the leaving jobs do not have good enough resources to complete them. Same to ASJS, Min-min gives the priority to the job with the smallest execution time. It makes some jobs that have not any resources that can ensure them to be finished as their requests. So, the performance of Min-min in *UFJ* is worse than MTS.

Fig. 4 and **Fig. 5** show the values of *SFS* and *SIN* of the four methods respectively. From those figures, we can see that MTS has transmitted more files and calculated more instructions. With the increase of q, *SFS* has an increasing trend and *SIN* has a decreasing trend. This is because that the Grid resource gives more time to transmit the files. For all the cases, to Min-min, ASJS and MRS, MTS average improves 6%, 4.75%, and 11.56% in *SFS* and average improves 2.27%, 2.12%, and 7.17% in *SIN*. MTS has finished more jobs than others, so, it has the highest value of *SFS* and *SIN* under all the cases (**Fig. 3**). In addition to that, MTS not only ensures the short job be finished firstly, but also ensures the jobs with only a few resources can guarantee it be finished first.

In conclusion, MTS reduces the average execution time and the number of uncompleted

jobs and enhances the *SIN* and *SFS*. The reasons are: (1) the scheduling orders ensure the job that has a few resources that can satisfy is scheduled first; (2) MTS also gives high priority to the job that has a low value of execution time, (3) MTS considers the resource fragment which saves small resource fragment. On the contrary, the target of Min-min is to minimize the execution time and it does not take into account the condition of the whole system; the target of ASJS is to find the highest score of the resource and it may give the best resource to a job that a normally resource can satisfy it. They bring the wasting of resources. MRS tries to find the best fit of resources for jobs according to the computing capacity and bandwidth of the resource, which also does not support some jobs that only have a few resources that can ensure them to be completed as the request.

4.3 Analysis of the time complexity of our method and the simulation

There are four key steps in our methods:

(1) Calculating the overload value of different attribute. The number of jobs is $jend$ and the number of resources is $mend$, and we consider three attributes: memory, bandwidth and CPU. So, the time complexity of the first step is $O(3*jend*mend)$.

(2) Calculating the item in different groups. There are four Groups: *ThreeOL*, *TwoOL*, *OneOL* and *OtherL*. The maximum length of the Group is $jend$, in other words, it equals to the number of the resources. In this step, we only take some execution of different sets, and the time complexity of the second steps is $O(4*jend)$

(3) Assigning the resource to the job. The maximum of the number of jobs is $jend$ and the maximum of the number of the resources is $mend$, so, the time complexity of Algorithm 3 is $O(jend*mend)$.

(4) Repeating execute step 1, 2 and 3. The maximum of the repeating time is $jend*mend$. So the total time complexity of our method is

$$O(jend*mend*(3*jend*mend+4*jend+jend*mend))=O(jend^2*mend^2)$$

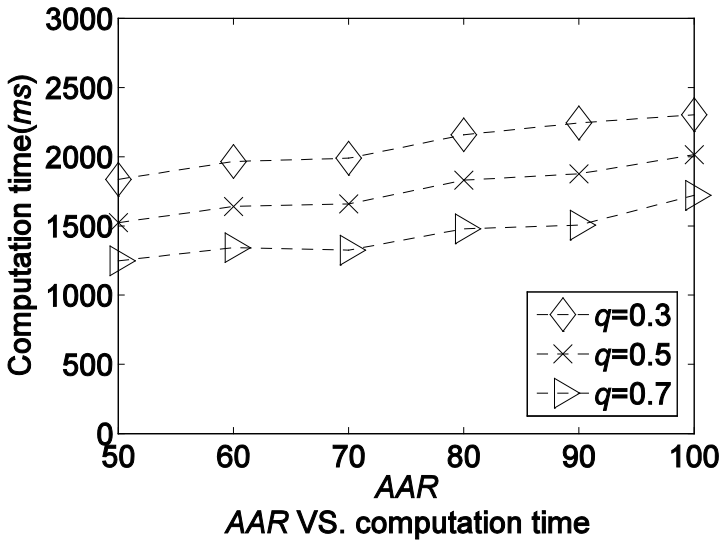

Fig. 6. The computation time under different arrival rates

Fig. 6 is the average scheduling time of the scheduling method. The simulation is the same as section 4.3. Even under our environment, there are 100000 resources in the system, and we consider three attributes, the maximum computation time is less than 2500 *ms*. So, our method is feasible in the system.

4.4 Simulations on a true log

This log contains several months accounting records from the National Grid of the Czech Republic, called MetaCentrum (http://www.cs.huji.ac.il/labs/parallel/workload/). This Grid is composed of 14 clusters (called nodes), each with numerous multiprocessor machines, for a total of 806 processors. Every processor has different processing ability. The resources and the jobs in the log have no information about the memory and bandwidth. We generate the information of them as the methods introduced in section 4.1. The deadline is set as a random number between the execution time (column 5 of the log) and the double execution time. There are 103 656 jobs in total. The log has no information of the input files and output files, and in the simulation, we do not pay much attention to the *SFS*.

Table 5. The simulation results of the true log

	Min-min	**ASJS**	**MRS**	**MTS**
AVE	1.7321+e4	1.7502+e4	1.7471+e4	1.6549+e4
UFJ	21359	22130	21031	20302
CT(ms)	722	906	943	815

Table 5 lists the simulation results of the three methods. The result shows MTS has advantages than the other methods in *AVE* and *UFJ*. To Min-min, ASJS and MRS, MTS reduces 772 seconds, 953 seconds and 922 seconds in *AVE*, and reduces 4.66%, 5.76%,and 5.57% respectively; MTS reduces 828, 1057 and 729 in *UFJ*, and reduces 4.08%, 5.21%, and 3.59% respectively. Section 4.3 also shows MTS has an advantage when the arrival rate has a lower value. The major reason for job failure scheduling is the system cannot provide enough resources in quality, such as the memory or the bandwidth is not enough, etc.

The *CT* (computation time) of Min-min, ASJS, MRS and MTS is 722 *ms*, 906 *ms*, 943 *ms* and 815 *ms*, respectively. MTS has a lower value of *CT* in the four methods and it only is more than Min-min. Comparison to Section 4.4, we find the computation time is less than the time in the simulation of section 4.4, two reason for this: the first is that we only consider on attributes in the log, and the second is that we just assign the job to different clusters, and there are only 14 clusters in the log.

5. Conclusion

In this paper, we try to consider the attributes of the resource and the requirement to the QoS of the job at the same time. According to the attribute of the resource and the requirement of the job in a Grid environment, we category jobs into different kinds. Then according to those classifying, jobs are inserted into different sets: *ThreeOL*, *TwoOL*, *OneOL*, and *OtherL*. We schedule the job in *ThreeOL* first and until there aren't any jobs in *ThreeOL*, and following by *TwoOL*, *OneOL*, and *OtherL*. We propose a scheduling method based on multiple attributes. Simulations prove that our method has a good performance in improving the number of finished jobs and reducing the execution time of the finished jobs.

In the future, we will apply MTS into a real Grid. This paper focuses on job scheduling.

Jobs are independent in this paper, but they may have some precedence relations in real-life situation. We will study and improve MTS for such kinds of jobs in the future. More attentions should be given to other attributes of the resource, such as the economy and power consumption.

Acknowledgments

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

References

[1] M. Maheswaran, S. Ali, H. J. Siegal, D. Hensgen, R. F. Freund, "Dynamic matching and scheduling of a class of independent tasks onto heterogeneous computing systems," in *Proc. of Heterogeneous Computing Workshop, 1999*, Proceedings. Eighth, pp. 30-44, 1999. Article (CrossRef Link)

[2] V. Hamscher, U. Schwiegelshohn, A. Streit, R. Yahyapour, "Evaluation of Job-Scheduling Strategies for Grid Computing," *Grid Computing-GRID 2000*, R. Buyya and M. Baker, Springer Berlin Heidelberg, 191-202, 1971. Article (CrossRef Link)

[3] S. García-Galán, R.P. Prado, J.E. Muñoz Expósito, "Rules discovery in fuzzy classifier systems with PSO for scheduling in grid computational infrastructures," *Applied Soft Computing*, Volume 29, Pages 424-435, ISSN 1568-4946, , April 2015. Article (CrossRef Link)

[4] K. Huang, P. Shih, C. Wu, "Towards Feasible and Effective Load Sharing in a Heterogeneous Computational Grid," *Advances in Grid and Pervasive Computing*, C. Cérin and K.-C. Li, Springer Berlin Heidelberg, 4459: 229-240, 2007. Article (CrossRef Link)

[5] Y.-H. Lee, S. Leu, R. Chang, "Improving job scheduling algorithms in a grid environment," Future Generation Computer Systems 27(8), 991-998, 2001. Article (CrossRef Link)

[6] T. Rings, G. Caryer, J. Gallop, J. Grabowski, T. Kovacikova, S. Schulz, I. Stokes-Rees, "Grid and cloud computing: opportunities for integration with the next generation network," *Journal of Grid Computing 7*, 375–393, 2009. Article (CrossRef Link)

[7] P.-C. Shih, K.-C. Huang, Y. Chung, "Improving grid performance through processor allocation considering both speed heterogeneity and resource fragmentation," *The Journal of Supercomputing 60(3)*, 311-337, 2012. Article (CrossRef Link)

[8] 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. Article (CrossRef Link)

[9] R. Webster, K. Munasinghe, A. Jamalipour, "A population theory inspired solution to the optimal bandwidth allocation for Smart Grid applications," *Wireless Communications and Networking Conference (WCNC), 2014 IEEE* , pp.2958,2963, 6-9 April 2014. Article (CrossRef Link)

[10] O. Ron, K. David, "Armada: a parallel I/O framework for computational grids," *Future Generation Computer Systems*, Volume 18, Issue 4, Pages 501-523, March 2002. Article (CrossRef Link)

[11] B. Schnizler, D. Neumann, D. Veit, C. Weinhardt, "Trading grid services-a multi-attribute combinatorial approach," *European Journal of Operational Research*, 187(3), 943-961, 2008. Article (CrossRef Link)

[12] K. Qureshi, B. Majeed, J. H. Kazmi and S. A. Madani, "Task partitioning, scheduling and load balancing strategy for mixed nature of jobs," *The Journal of Supercomputing*, Volume 59, Number 3, Pages 1348-1359, 2012. Article (CrossRef Link)

[13] R. Albodour, A. James, N. Yaacob, "High level QoS-driven model for Grid applications in a simulated environment," *Future Generation Computer Systems 28(7)*,1133-1144, 2012.

Article (CrossRef Link)
[14] C. Joe-Wong, S. Sen, T. Lan, M. Chiang, "Multi-resource allocation: fairness-efficiency tradeoffs in a unifying framework," Princeton Uni-versity, Tech. Rep., 2011. Article (CrossRef Link)
[15] A. Sulistio, G. Poduval, R. Buyya, C. Tham, "On incorporating differentiated levels of network service into GridSim," *Future Generation Computer Systems*, 23(4): 606-615, 2007. Article (CrossRef Link)
[16] Y. Hao, G. Liu, R. Hou, Y. Zhu, J. 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)
[17] V. Chinnaiah, T. S. Somasundaram, "A Grid resource brokering strategy based on resource and network performance in Grid," *Future Generation Computer Systems*, 28(3), 491-499, 2012. Article (CrossRef Link)
[18] F. Xhafa, A. Abraham, "Computational models and heuristic methods for Grid scheduling problems," *Future Generation Computer Systems*, 26(4), 608-621, 2010. Article (CrossRef Link)
[19] H. Sun, J. Huai, Y. Liu, R. Buyya, "RCT: A distributed tree for supporting efficient range and multi-attribute queries in grid computing," *Future Generation Computer Systems*, 24(7), 631-643, 2008. Article (CrossRef Link)
[20] L. Wang, J. Cao, "Design of attribute-based access control model in collaborative grid environment," *Energy Procedia*, Volume 13, Pages 529-536, ISSN 1876-6102, 2011. Article (CrossRef Link)
[21] R. Chang, C.-Y. Lin, C. Lin, "An Adaptive Scoring Job Scheduling algorithm for grid computing," *Information Sciences*, Volume 207, Pages 79-89, 10 November 2012. Article (CrossRef Link)
[22] B. T. B. Khoo, B. Veeravalli, T. Hung, C. W. Simon, "Multi-dimensional scheduling scheme in a Grid computing environment," *Journal of Parallel and Distributed Computing*, Volume 67, Issue 6, Pages 659–673, June 2007. Article (CrossRef Link)
[23] Y. Hao, G. Liu, N. Wen, "An enhanced load balancing mechanism based on deadline control on GridSim," *Future Generation Computer Systems*, 28(4), 657-665, 2012. Article (CrossRef Link)
[24] M. Malawski, G. Juve, E. Deelman, J. Nabrzyski, "Algorithms for cost- and deadline-constrained provisioning for scientific workflow ensembles in IaaS clouds," *Future Generation Computer Systems*, Volume 48, Pages 1-18, July 2015. Article (CrossRef Link)
[25] X. Zhao, N. Jamali, "Supporting Deadline Constrained Distributed Computations on Grids," *Grid Computing (GRID), 2011 12th IEEE/ACM International Conference on*, pp.165,172, 21-23 Sept. 2011. Article (CrossRef Link)
[26] R. Garg, A. K. Singh, "Adaptive workflow scheduling in grid computing based on dynamic resource availability," *Engineering Science and Technology, an International Journal*, Volume 18, Issue 2, Pages 256-269, June 2015. Article (CrossRef Link)
[27] T. M. Hansen, R. Roche, S. Suryanarayanan, A. A. Maciejewski, H. J. Siegel, "Heuristic Optimization for an Aggregator-Based Resource Allocation in the Smart Grid," *IEEE Transactions on Smart Grid*, Volume: PP, Issue: 99, 2015. Article (CrossRef Link)
[28] M. B. Qureshi, M. M. Dehnavi, N. Min-Allah, M. S. Qureshi, H. Hussain,I. Rentifis, N. Tziritas,T. Loukopoulos, S. U. Khan, C. Xu, A. Y. Zomaya, "Survey on Grid Resource Allocation Mechanisms," *Journal of Grid Computing*, Volume 12, Issue 2, pp 399-441, June 2014. Article (CrossRef Link)
[29] D. Kaur, J. SenGupta, "P2P Trust and Reputation Model for securing Grid resource management," *Advances in Engineering, Science and Management, 2012 International Conference on*, pp.524,529, 30-31 March 2012. Article (CrossRef Link)
[30] K. Qureshi, A. Rehman, M. Paul. "Enhanced GridSim architecture with load balancing," *The Journal of Supercomputing*, 57(3), 265-275, 2011. Article (CrossRef Link)
[31] K. B. Bey, F. Benhammadi, A. Mokhtari, Z. Gessoum, "Mixture of ANFIS systems for CPU load prediction in metacomputing environment," *Future Generation Computer Systems*, 26(7), 1003-1011, 2010. Article (CrossRef Link)

Yongsheng Hao received his MS Degree of Engineering from Qingdao University in 2008. Now, he is a 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.