A parallel tasks Scheduling heuristic in the Cloud with multiple attributes

Qin Wang[1], Rongtao Hou[1], Yongsheng Hao[1], Yin Wang[2, 3]
[1]School of computer and software, Nanjing University of Information Science & Technology, Nanjing, 210044, China
[e-mail: 001392@nuist.edu.cn, 001452@nuist.edu.cn, 002004@nuist.edu.cn]
[2] School of public administration, Nanjing University of Information Science & Technology, Nanjing, 210044, China
[3]Chizhou weather bureau, Anhui, China
*Corresponding author: Q.Wang, R. Hou and Y. Hao

Received May 11, 2017; revised July 19, 2017; accepted August 30, 2017; published January 31, 2018

Abstract

There are two targets to schedule parallel jobs in the Cloud: (1) scheduling the jobs as many as possible, and (2) reducing the average execution time of the jobs. Most of previous work mainly focuses on the computing speed of resources without considering other attributes, such as bandwidth, memory and so on. Especially, past work does not consider the supply-demand condition from those attributes. Resources have different attributes, considering those attributes together makes the scheduling problem more difficult. This is the problem that we try to solve in this paper. First of all, we propose a new parallel job scheduling method based on a classification method of resources from different attributes, and then a scheduling method-CPLMT (Cloud parallel scheduling based on the lists of multiple attributes) is proposed for the parallel tasks. The classification method categories resources into different kinds according to the number of resources that satisfy the job from different attributes of the resource, such as the speed of the resource, memory and so on. Different kinds have different priorities in the scheduling. For the job that belongs to the same kinds, we propose CPLMT to schedule those jobs. Comparisons between our method, FIFO (First in first out), ASJS (Adaptive Scoring Job Scheduling), Fair and CMMS (Cloud-Minmin) are executed under different environments. The simulation results show that our proposed CPLMT not only reduces the number of unfinished jobs, but also reduces the average execution time.

Keywords: parallel tasks, Cloud resources, multiple attributes, job requirements

http://doi.org/10.3837/tiis.2018.01.014 **ISSN : 1976-7277**

1. Introduction

Recent researches in Cloud include reliability [1], resource virtualization [2], service selection [3], energy saving [4], cost analysis [5] and so on. This paper focuses on the scheduling of parallel tasks in Cloud. The parallel tasks scheduling ensures that the tasks of a job are assigned to different VMs at the same time and executed synchronous. The parallelism brings a new challenge and it makes the scheduling more difficult. Y. Hao et al. [6] take the job with many tasks as a "Gang", and the parallel scheduling is Gang scheduling. In Cloud, related tasks of the same job run simultaneously on different VMs (Virtual machines). Usually these jobs hold all the VMs that have been assigned to until all of the job have been finished.

Maximum throughput and minimum execution time with multiple attributes are the two most important scheduling of parallel tasks [7]. Much work has been done from different aspects of the scheduling problem and on different platforms. Y. Wang et al. [8] propose MOWS (Mixed-Parallel Online Workflow Scheduling) for the scheduling of parallel workflows in a speed-heterogeneous multi-cluster environment. MOWS divides the entire scheduling process into four phases: task prioritizing, waiting queue scheduling, task rearrangement, and task allocation. They propose four methods for the scheduling: shortest-workflow-first, priority-based backfilling, preemptive task execution and All-EFT (All-Earliest-Finish-Time) task allocation. J. Xu et al. [9] propose a hierarchical task mapping strategy, which not only focuses on the task mapping between compute nodes (i.e., inter-node mapping), but also focuses on the mapping within a node (i.e., intra-node mapping). They propose a hierarchical task mapping strategy, which performs both internode and intra-node mapping at the same time. Two mapping algorithms is used in the scheduling: (1) a generic recursive tree mapping algorithm is used to handle both inter-node mapping and intra-node mapping; (2) a recursive bipartitioning mapping algorithm is used to efficiently partitions the compute nodes according to their coordinates. L. Liu et al. [10] propose FDMHSV (Fairness of Dynamic Multiple Heterogeneous Selection Value) to solve the scheduling of parallel jobs from two aspects: heterogeneity and fairness. They use HPRV (Heterogeneous Priority Rank Value) and HSV (Heterogeneous Selection Value) for task ordering and processor assignment to improve the calculation of computation heterogeneity. At the same time, a fairness policy is proposed to allow each job to be scheduled with a fair opportunity without blocking and waiting when a new job arrives. K. Huang et al. [11] propose two task-ranking mechanisms (bottom rank and top rank) and one task allocation method for the two major steps in list-based workflow scheduling under a parallel computing platform. The scheduling method pays more attention to the amount of available resources. Most of the past work focuses on the speed of the resource. Those methods always schedule resources from the computing speed of resources without considering the supply-demand of bandwidth, memory and so on. In the Cloud, the VMs in a Cloud center share the resources including the CPU, memory and the bandwidth, this makes it more important to consider all the attributes. In the paper, we try to consider different attributes of the resources and the supply-demand balance between the resources and the job. Our method considers more attributes that makes the scheduling problem more difficult [7]. Different to the previous work, our work is based on a new classification method of the job which considers the number of resources that ensures the job can be finished as request. Our objective is to give a scheduling method to ensure that we can maximize throughput and minimize the execution time of the job. For those targets, first of all, we category jobs into

different kinds according to the urgent condition, then we give different orders for different kinds of jobs, and last, a new scheduling method is proposed based on our classification.

The main contributions of the paper include: (1) we give a detailed analysis of parallel scheduling in the Cloud; (2) a new classification method for jobs is proposed which takes account of the requirements to different QoSs (Quality of Services) of jobs and the multiple attributes of resources at the same time; (3) a scheduling order for different kinds of jobs is given from the analysis; (4) CPLMT is proposed to solve the problem of parallel tasks scheduling in the Cloud; (5) simulations on a simulated environment and to a real system are executed to test the performance of our method.

The rest of the paper is organized as follows: section 2 is the related work, section 3 illustrates the framework of our system and the scheduling method of parallel tasks, section 4 is the simulation and the evaluation, section 5 discusses the paper, section 6 is the conclusion and the future work.

2. Related Work

Past work on parallel tasks has been executed in different areas including clusters, Grid, multiple processors. The work also has been executed from different aspects. K. Oh-Heum et al. [12] try to solve the problem of scheduling independent parallel tasks with individual deadlines. To maximize the total work performed by the tasks which completes their executions before deadlines, two polynomial-time approximation algorithms are proposed for nonmalleable parallel tasks and malleable tasks. H. Ting et al. [13] pay attentions to the problem of scheduling low-priority tasks onto resources already assigned to high-priority tasks. They formulate the problem as a Markov Decision Process (MDP) whose solution gives the optimal scheduling policy. At the same time, they discover structures of the problem in the special case of homogeneous availability patterns that enable a simple threshold-based policy that is provably optimal. Kurowski et al. [14] propose a simple on-line scheduling policy and generic advices that reduce the negative impact of advance reservations on a schedule quality for parallel tasks. They also propose novel data structures and algorithms for efficient scheduling of advance reservations. Y. Hao [15] et al. try to give a scheduling method under multi-Cloud environment. Y. Xia [16] et al. propose an approach using parallelized fusion on multi-sensor transportation data for intelligent transportation systems (ITS). The system consists of four components, which are sensor data input, bootstrapping rough conversion, hierarchical evidential fusion, and traffic state output. Their computation intensity is centered on conversion and fusion components, which can be optimized by the algorithm- and data-centric parallelization, respectively.

The Cloud environment brings both challenges and opportunities to the parallel tasks scheduling [13,17]: On the one hand, computing intensive Cloud jobs, such as weather model computing [18], large-scale graph processing, satellite data processing, and Map-Reduce applications, often contain tasks with synchronization requirements that needs multiple VMs (virtual machines). On the other hand, the virtualized Cloud environment allows us to manage tasks more actively. Sometimes, it is possible to migrate the VMs of interrupted backend tasks to other available VMs instead of waiting indefinitely. Migration, however, comes at an operational

cost due to the data and VM state transferring. It also needs time and resources support. Due to the parallelism of the job, the migration of one VM may influence more VMs that the tasks of the same jobs have been assigned to. Thus, judicious decisions must be made when using migration to improve the performance of backend tasks. This paper does not take account of migration because of its complexity in parallel scheduling.

Even there is difficult for the scheduling of parallel tasks in a Cloud, recent researchers have been done from different aspects. Most of work focuses on the performance of different aspects: the number of finished jobs, the average execution time, the energy consumption and so on. Most of times, those targets are conflicting with each other, so, a tradeoff always is used to schedule those jobs. H. Ting et al. [17] study the problem of scheduling parallel (backend) tasks onto opportunistically available server resources to focus on the tradeoff between migration and waiting. Their proposed heuristic scheduling policy based on Whittle's index that greatly reduces the complexity of the optimal policy while achieving good performance under a variety of servers. The target of ASQ (Adaptive Scheduling with QoS Satisfaction algorithm) [23] is to reduce the cost of parallel tasks in a hybrid Cloud environment. For the economy and the efficiency reasons, the hybrid Cloud environment should be able to automatically maximize the utilization rate of the private Cloud and minimize the cost of the public Cloud when users submit their computing jobs to the environment. For the tasks that have to be dispatched to the public Cloud, the minimum cost strategy reduces the cost of using public Clouds based on the requirements of tasks such as memory, hard disk and so on. X. Liu et al. [19] propose a consolidation-based parallel job scheduling algorithm based on a prioritized two-tier virtual machines architecture for parallel workload consolidation. The algorithm employs tentative run and workload consolidation under such a two-tier virtual machines architecture to enhance the popular FCFS (First Come First Served) algorithm. To schedule parallel scientific Workflows, C. Coutinho et al. [20] use GraspCC-fed: a Greedy Randomized Adaptive Search Procedure that estimates costs based on two factors: total workflow execution time and financial cost of the workflow execution. GraspCC-fed is responsible for determining the best configuration for the environment before a parallel workflow execution by using provenance information for the estimations. The scheduling of parallel tasks in the Cloud, most of research either focus on the speed of resources, or the management of the VMs (virtual machine), and those research is from the view of the computing speed of resources. OMO [29] is used to improve the makespan of batch jobs by optimizing the overlap between two active consecutive stages. The basic approach is to let multiple jobs fairly share the system resource, and then focus on the resource allocation for consecutive stages in each job. OMO considers dynamic factors at the run time and allocates the resources based on the dependency of stages in every job. In general, those methods always based on the attributes of jobs, they try to propose a scheduling method that can meet the requirement of jobs. Those requirements include: QoS requirement to different aspects, energy consumption, QoE (Quality of Experience) and so on. They do not consider that the different attributes of resources always influence the scheduling results.

In conclusion, no matter which platforms, most of the work for the scheduling of parallel tasks is from the request of the user, such as the deadline, priorities and so on. They do not consider the supply and demand of the whole system. Especially, most methods do not take account of the dynamic of the system. One job with a long deadline, with time going, it also can be an urgent job in the system. Our target is to consider the two aspects at the same time, and decide which job should be executed first. Based on the number of resources that can ensure the job be finished from

different aspects, such as memory, hard disk, and CPU, we classify jobs into different kinds, and schedule them with different orders.

3. Scheduling framework of parallel tasks

3.1 Cloud frameworks and scheduling method

Fig. 1 is the system framework of the Cloud. In **Fig. 1**, the five tasks of the job 1 are assigned to different VMs (VM 1~ VM 5) and every job may have different requirements to the resource. And in this paper, we suppose the task that belongs to the same job has the same requirement and the task belongs to different jobs has different requirement. **Fig. 2 i**s an example of three jobs.

jend is the total number of jobs. All jobs (j_{jtemp}) are listed in the jobs list *jlist*:

$$jlist=\{j_1, j_2, \ldots, j_{jtemp}, \ldots, j_{jend}\} \quad (1)$$

The parallelism of the job j_{jtemp} is p_{jtemp}. In Fig. 2, the parallelisms of job 1, 2 and 3 are 5, 4 and 6, respectively. The parallelisms of all jobs are listed in *plist*:

$$plist=\{p_1, p_2, \ldots, p_{jtemp}, \ldots, p_{jend}\} \quad (2)$$

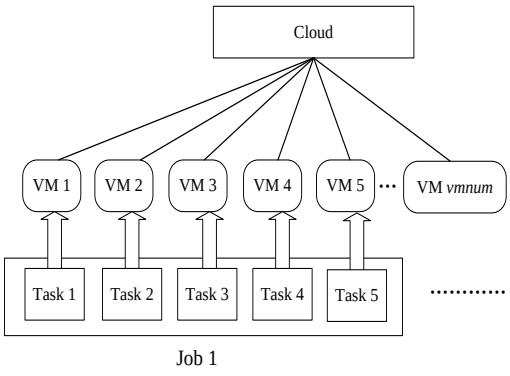

Fig. 1. The Framework of Cloud

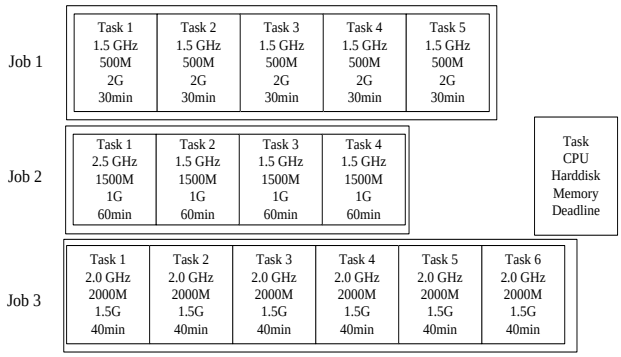

Fig. 2. Example of three jobs with different attributes

For the job j_{jtemp}, it has p_{jtemp} tasks:

$$j_{jtemp} = \{t^1_{jtemp}, t^2_{jtemp}, \ldots, t^{ptemp}_{jtemp}, \ldots, t^{pt}_{jtemp}\} \quad (3)$$

where $pt = pj_{ptemp}$.

The requirements of the task t^{pt}_{jtemp} about different attributes are:

$$t^{pt}_{jtemp} = \{HD^{pt}_{jtemp}, MEM^{pt}_{jtemp}, BW^{pt}_{jtemp}, DL^{pt}_{jtemp}\} \quad (4)$$

HD^{pt}_{jtemp},MEM^{pt}_{jtemp}, BW^{pt}_{jtemp} and DL^{pt}_{jtemp} are the hard disk, memory, bandwidth and the deadline to the requirement of the resource of the task t^i_{jtemp}.

The total amount of VMs is *vend*. All the VMs are listed in set *Vlist*

$$Vlist = \{v_1, v_2, \ldots, v_{vtemp}, \ldots, v_{vend}\} \quad (5)$$

The attributes of the VM v_{vtemp} are :

$$v_{vtemp} = \{VHD_{vtemp}, VMEM_{vtemp}, VCPU_{vtemp}, VBW_{vtemp}\} \quad (6)$$

VHD_{vtemp}, $VMEM_{vtemp}$, $VCPU_{vtemp}$,andVBW_{vtemp} are the attributes of the hard disk, memory, CPU and bandwidth of v_{vtemp}.

In this paper, we suppose that all the resources are space-shared, in other words, a VM only has one task at a time.

Algorithm 1 is the details of calculating $t_{jtemp}^{pt}.ovCPU$. $t_{jtemp}^{pt}.ovCPU$ is the number of resources that the load of CPU is more than α when the resource is assigned to the job t_{jtemp}^{pt}. The upper limit ratio of the VMs of the processing ability is α , *CPUload* (Line 3, Algorithm 1) is the value of the load of the CPU when the task t_{jtemp}^{pt} is assigned to the VM v_{vtemp}. If the value is more than α, the value of $t_{jtemp}^{pt}.ovCPU$ is increased by 1 (Line 5, Algorithm 1). $t_{temp}^{pt}.ovCPU$ records the number of the resources that cannot give enough processing ability to the task.

Algorithm 1: CptCPU (*Vlist*, t_{jtemp}^{pt})

Input: *Vlist*, t_{jtemp}^{pt}
Output: $t_{jtemp}^{pt}.ovCPU$
1. $t_{jtemp}^{pt}.ovCPU = 0$;
2.For every VM v$\in$ Vlist
3. $CPUload = l(t_{jtemp}^{pt}, v)$;
4. **If**($CPUload \geq \alpha$)
5. $t_{jtemp}^{pt}.ovCPU = t_{jtemp}^{pt}.ovCPU + 1$;
6. **Endif**
7. Endfor

In the same way to $t_{jtemp}^{pt}.ovCPU$, $t_{jtemp}^{pt}.ovMEM$, $t_{jtemp}^{pt}.ovBW$ and $t_{jtemp}^{pt}.ovHD$ record the number of resources that cannot satisfy the job from the views of the memory, bandwidth and hard disk. They have the same calculation method with $t_{jtemp}^{pt}.ovCPU$ and they are not been introduced here. The parameters are used in the calculation of them include:

α, β, γ and ϑ are the upper limit ratios of the number of the resources that satisfy the task in the aspect of the CPU, memory, bandwidth and hard disk;

CPUload, *MEMload*, *BWload* and *HDload* are the loads of the VM *v* of the CPU, memory, bandwidth and hard disk when the task t_{jtemp}^{pt} is assigned to the VM *v*;

l(*t*, *v*), *m*(*t*, *v*), *n*(*t*, *v*) and *o*(*t*, *v*) are the loads of the CPU, memory, bandwidth and hard disk when the task is assigned to the resource.

Different to non-parallel tasks, if there are not enough VMs for the parallel tasks, the whole parallel job cannot be finished. In Fig. 2, for the job 3, if there are only three VMs for the job,

though they can provide enough hard disk, memory and bandwidth, the job cannot be finished under this case.

According to the number of resources that can satisfy the job, we classify jobs into two kinds: *ujobs* and *unjobs*. *ujobs* refers to the job that there is only a few resources can satisfy the tasks of the job. *unjobs* refers to the job that most of the resources can satisfy the tasks of the job. We get the classification from the ratio between the number of resources that can satisfy all the tasks of the job (num_{jtemp}) and the value of the parallelism of the job (p_{jtemp}). If the value is more than K, the job belongs to *unjobs*, otherwise, the job belongs to *ujobs*. K is the upper limit of the ratio between the resource and the parallelism of the job.

$$job_{jtemp} = \begin{cases} ujobs, if \frac{num_{jtemp}}{p_{jtemp}} \leq K \\ unjobs, \ if \frac{num_{jtemp}}{p_{jtemp}} > K \end{cases} \tag{7}$$

At the same time, we will give other detailed classifications from different aspects.

For the job j_{jtemp}, $j_{jtemp}.jovCPU$, $j_{jtemp}.jovBW$ and $j_{jtemp}.jovHD$ are the overload value of the job for the CPU, memory, bandwidth and hard disk. The values of them are the maximum value of the relative values of the task that belong to the job:

$$j_{jtemp}.jovMEM = \max\{t^i_{jtemp}.ovMEM | i \leq p_{jtemp}\} \tag{8}$$

$$j_{jtemp}.jovHD = \max\{t^i_{jtemp}.ovHD | i \leq p_{jtemp}\} \tag{9}$$

$$j_{jtemp}.jovCPU = \max\{t^i_{jtemp}.ovCPU | i \leq p_{jtemp}\} \tag{10}$$

$$j_{jtemp}.jovBW = \max\{t^i_{jtemp}.ovBW | i \leq p_{jtemp}\} \tag{11}$$

(i) According to the requirement to the capacity of hard disk of the job

We can get the number of VMs that cannot provide enough hard disk to the job, and it is denoted by $j_{jtemp}.olHD$. If $j_{jtemp}.olHD$ is close to the number of VMs, it means that only a few VMs can satisfy the job j_{jtemp}. We set a boundary parameter ε to express the upper limit of the number of resources that the resource can provide enough hard disk. If $j_{jtemp}.olHD$ is more than $\varepsilon \times vend$, we add j_{jtemp} into *ulHD*, otherwise, we add the job j_{jtemp} to *unulHD*. Jobs in *ulHD* mean there are enough resources to satisfy those jobs, on the country, jobs in *ulHD* mean there are only a few resources can satisfy those jobs.

$$ulHD=\{j_t | j_t \in jlist \wedge j_t.olHD > \varepsilon \times vend\} \tag{12}$$

$$unulHD=\{j_t | j_t \in jlist \wedge j_t.olHD <= \varepsilon \times vend\} \tag{13}$$

ulHD refers to the jobs that there are enough resources can satisfy them, on the contrary, *unulHD* means the job only has a few resources can ensure them be completed as the request of the jobs. In the same way, we can category jobs into different sets according to the requirements of the memory and the bandwidth. ω is the upper limit of the number of resources that can provide enough memory and η is the upper limit of the number of resources that can provide enough bandwidth. Formulas 14~15 are the two sets that from the view of the memory of resources, and Formulas 16~17 are the two sets that from the view of the bandwidth of resources.

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

$$ulMEM = \{j^t | j^t \in jlist \wedge jt.olMEM > \omega \times vend\} \tag{14}$$

$$unulMEM = \{j^t | j^t \in jlist \wedge jt.olMEM \leq \omega \times vend\} \tag{15}$$

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

$$ulBW = \{j^t | j^t \in jlist \wedge j^t.olBW > \eta \times vend\} \tag{16}$$

$$unulBW = \{j^t | j^t \in jlist \wedge\ jt.olBW \leq \eta \times vend\} \tag{17}$$

From the three kinds of categories, we list the jobs that belong to *ulHD*, *ulMEM* and *ulBW* in *ThreeOL*, list the jobs that only belong to two of them in *TwoOL*, and the jobs that only belong to one of them in *OneOL*. Others jobs are listed in *OtherL*. *ThreeOL* refers to the jobs only have a few resources to ensure that they can be finished from three aspects: memory, hard-disk, CPU. *TwoOL* and *OneOL* refer to the jobs only have a few resources ensure them can be finished from two of the three aspects, or one of the three aspects respectively.

$$ThreeOL = ulMEM \cap ulBW \cap ulHD \tag{18}$$

$$\begin{aligned} TwoOL = {} & (ulMEM \cap ulBW \cap unulHD) \cup \\ & (ulMEM \cap unulBW \cap ulHD) \cup \\ & (unulMEM \cap ulBW \cap ulHD) \end{aligned} \tag{19}$$

$$\begin{aligned} OneOL = {} & (ulMEM \cap unulBW \cap unulHD) \cup \\ & (unulMEM \cap ulBW \cap unulHD) \cup \\ & (unulMEM \cap unulBW \cap ulHD) \end{aligned} \tag{20}$$

$$OtherL = jlist - ThreeOL - TwoOL - OneOL \tag{21}$$

In the scheduling, first of all, we schedule *ujobs*, and then we schedule *unjobs*. For *unjobs*, from above analysis, we know that only a few VMs can meet the requirement of the job in *ThreeOL*, so we firstly schedule the jobs in *ThreeOL*, and then we select the job in *TwoOL*, *OneOL*, and *OtherL* in sequence. $j_{jtemp}.tovload$ (Formula 22) is the total value of the overload of different attributes.

$$j_{jtemp}.tovload = j_{jtemp}.jovMEM + j_{jtemp}.jovBW + j_{jtemp}.jovHD \tag{22}$$

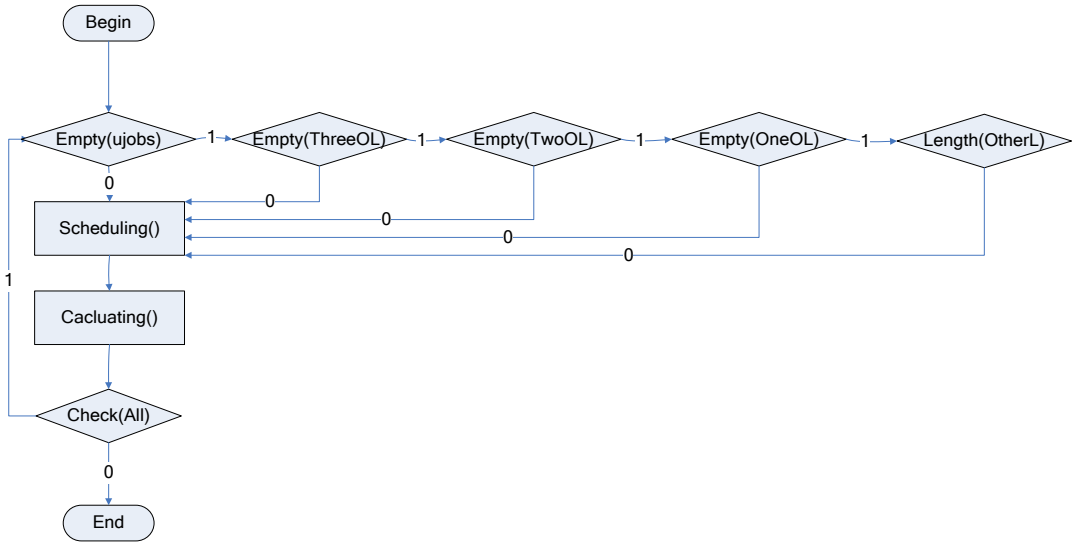

Fig. 3. The scheduling order of different sets

Fig. 3 is the detailed scheduling order of different sets. "Scheduling()" is in charge of scheduling of jobs in the same kind. In the next section, we will give detailed information about the scheduling. "Empty(*X*)" checks that the set *X* whether is empty, if it is empty, the function returns true (1), otherwise, it returns false (0). "Check(*ALL*)" checks whether there are more jobs can be finished before their deadlines.

If two kinds of jobs require different attributes of resources, such as one is computation-intensive, another is memory-intensive, and then which have a priority is decided by: (1) the number of jobs belong to computation-intensive and memory-intensive, and (2) attributes of those jobs and the resources. From Formulas 10~16, we know that the job belongs to which kinds not only decided by the attributes of jobs, but also related to the resources. An example is for computation-intensive jobs, if there are many resources with high processing ability, there are no problems to execute the job (it belongs to *ulHD*). So, our method can give the jobs different orders according to the system load, and give more attention to the attributes of jobs.

3.2 Enhanced parallel tasks scheduling with multiple attributes in the Cloud

There are two steps in the scheduling: (1) selecting a job; (2) assigning the tasks of the selected job to different VMs.

Algorithm 2 is the detailed information about selecting job:

Algorithm 2: select job(*Vlist*, *jlist*)

Input: *minvalue*=0;
Output: *selectj*; /* select the job */
1. For every job job_i in *jlist*
2. *temp*=*mt*(job_i, *Vlist*);
3. **If** (*temp*<*minvlue*);
4. *minvalue*= *temp*;
5. *selectj*= job_i;
6. **Endif**
7. Endfor
8. Return(*selectj*)

The value of *temp* is decided by v_{vtemp} and j_{jtemp}. The attributes of j_{jtemp} and the requirements of v_{vtemp} include processing ability, memory capacity, and hard disk capacity and so on. In Algorithm 2, we give two rules for the scheduling:

(1) Minimizing the execution time of the job;

(2) Giving the job that only has a few resources can satisfy a higher priority.

So, we set

$$mt(job_i, Vlist) = \frac{\max(\min(f(t,Vlist))}{e^{job_i.tovload}} \quad (23)$$

Where,

-*t* is a task of job_i;

-min$(f(t, Vlist)$ returns a set of the minimum execution times of every task in job_i;

-max(min($f(t,Vlist)$) returns the maximum of the execution time of the jobs, the execution time is decide by the largest execution time of the tasks of the job;

-$job_i.tovload$ is the total overload value of the job, see formula (22).

The second problem is how to assign the task of the job (*selectj*) to different VMs (Algorithm 3). First of all, we calculate the minimum execution time of every task; then we select the maximum value in all the minimum execution time as a standard (lines 1-12); For a task, if we can find a VM that makes the task can be finished before *maxminet* and the time is the most nearest to *maxminet*, we assign the task to the VM ; if we cannot find enough resources that satisfy the job, we try to find a VM that makes the execution time most nearest to the *maxminet* (but more than *maxminet*) (lines 13-33).

Algorithm 3: assign (*selectj*, *Vlist*)

```
1.  Input: selectj, Vlist;
2.  Output: selectpos, selectneg;
3.  maxminet=0;
4.  For every task t in selectj
5.    For every VM V in Vlist
6.      Cacluate the execution time (et) of the task t on V;
7.      If et< maxminet or maxminet==0
8.        maxminet=et;
9.        selectj=t;
10.     Endif
11.   Endfor
12. Endfor
13. minneg=0;
14. selectneg=0;
15. minpos=0;
16. selectpos=0;
17. For every task t in selectj
18.   For every VM V in Vlist
19.     Cacluate the execution time (et) of the task t on V;
20.     If et< maxminet or minneg=0
21.       minpos= maxminet-et;
22.       selectpos=V;
23.     Else
24.       minneg=et-maxminet;
25.       selectneg=V;
26.     Endif
27.   Endfor
28. If minpos!=0
29.   Assign the task t to selectpos;
30. Else
31.   Assign the task t to selectneg;
32. Endif
33. Endfor
```

In General, our method- CPLMT (Cloud parallel scheduling based on the lists of multiple attributes) is proposed for the scheduling of parallel tasks. First of all, we judge that the job belongs to which kind from different aspects of resources and jobs. Formulas 9~17 judge the job from CPU, memory, bandwidth and hard disk aspects, and according to those formulas a new classification method is used in formula 18~21. According to those mentioned classification methods, jobs which have a lower number of resources (from different aspects) to satisfy always has a higher priority. After we get the scheduling order of jobs, then how to select the resources is the next important problem. We select resources to jobs according to Algorithm 2 and 3. Algorithm 2 and 3 are heuristics and try to consider two aspects at the same time: minimizing the execution time and saving computing resources.

4. Evaluations

In this Section, we will compare our method (CPLMT) with other methods. Section 4.1 gives a simulation environment based on Matlab. Section 4.2 gives comparisons from different aspects. Section 4.3 gives results of different method in Methodological cloud center and we will give the performance of different methods.

4.1 Simulation environments

In this section, we will introduce the simulation environment and the performance metrics of simulations in this section.

We give a comparison between our method and FIFO (First in First out) [6, 23], ASJS (Adaptive Scoring Job Scheduling) [23], Fair [22, 23] and CMMS (Cloud-Minmin) [24]. We select those methods because:

FIFO is widely used in the Cloud and it is easy to use in a real system;

Fair is used in Cloud and it has been applied in some true Cloud platforms;

ASJS is a scheduling method based on a scoring of different attributes, and our method also focuses on different attributes, so we also give a comparison between them;

Min-min is widely used in Grid and the past work shows that it has advantages under most cases. It also has been extended to the Cloud and the simulation also shows that it has a good performance.

The configuration of the resources is used in our simulations: Windows XP on an Intel Core (2.66 GHz and 2.66 GHz), with 2048 MB of RAM and 600 GB of hard disk.

Let us suppose that the system has 120 resources. This is a limitation posed by Amazon EC2 which allows up to 20 “Regular” and up to 100 “Spot” VMs which can be leased under certain conditions [6], hence the maximum number of VMs is 120. We run the simulation 50 times. The values in all the figures are the average values.

When the job j_{jtemp} is assigned to the VM $Vtlist = \{v_1, v_2 \ldots, v_{pal}\}$, etc. the task t_{temp} is assigned to the VM v_{temp}, the functions of this paper are defined as follows:

$$CPUload = \max_{t<pal}(\frac{Pj_t}{Pv_t \times \alpha}) \quad (24)$$

$$Memload = \max_{t<pal}(\frac{Mj_t}{Mv_t \times \beta}) \quad (25)$$

$$Bwload = \max_{t<pal}(\frac{Bj_t}{Bv_t \times \gamma}) \quad (26)$$

$$Hdload = \max_{t<pal}(\frac{Hj_t}{Hv_t \times \vartheta}) \quad (27)$$

Where,

pal is the parallelism of the job j_{jtemp};

Pvlist ={Pv_1, Pv_2…, Pv_{pal}} is the processing ability of CPUs of different VMs;

Mvlist ={Mv_1, Mv_2…, Mv_{pal}} is the capacity of memory of different VMs;

Bvlist ={Bv_1, Bv_2…, Bv_{pal}} is the bandwidth of different VMs;

Hvlist ={Hv_1, Hv_2…, Hv_{pal}} is the capacity of hard disk of different VMs;

Pjlist ={Pj_1, Pj_2…, Pj_{pal}} is the requirement to computing speed of different tasks;

Mjlist ={Mj_1, Mj_2…, Mj_{pal}} is the requirement to the memory capacity of different tasks;

Bjlist ={Bj_1, Bj_2…, Bj_{pal}} is the requirement to the bandwidth of different tasks;

Hvlist ={Hj_1, Hj_2…, Hj_{pal}} is the requirement to the hard disk of different tasks.

Today, because of the cheap of the harddisk and the easy management of harddisk, we do not consider the requirement to the harddisk.

We set $\partial = 1$, $\beta = 1$, $\delta = 1$ in the simulation. In other words, the VM can give all of its ability to the task. In fact, the user can set its value as wishes. $\varepsilon = 0.8$, $\omega = 0.8$, $\eta = 0.8$, it means that, if there are more than 80% resources cannot provide enough resources according to a special QoS, the job will be inserted into an urgent set of the QoS.

Four evaluation parameters are selected in our simulations:

(1) AVE: average execution time, it includes the waiting time and the computing time;

(2) AWT: average waiting time;

(3) UFJ: the number of unfinished jobs;

(4) SIN: the total number of instructions of finished jobs.

Table1. Parameters about Cloud

Attributes	VMs	Jobs
Memory (G)	[1, 5]	[1, 5]
Processing ability	[1, 6]	[5, 45]
Bandwidth (100M/s)	[1, 20]	[1, 20]
VMs number	120	[1, 6]
Parallelism	-	[1, 8]

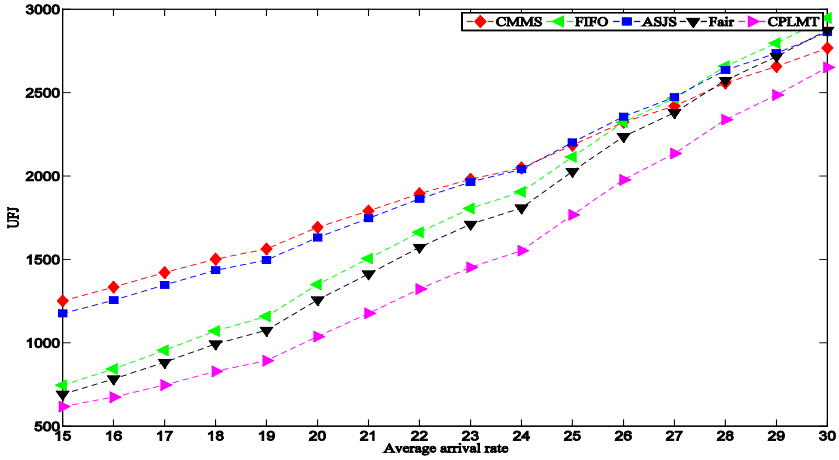

Fig. 4. The UFJ of different arrival rates

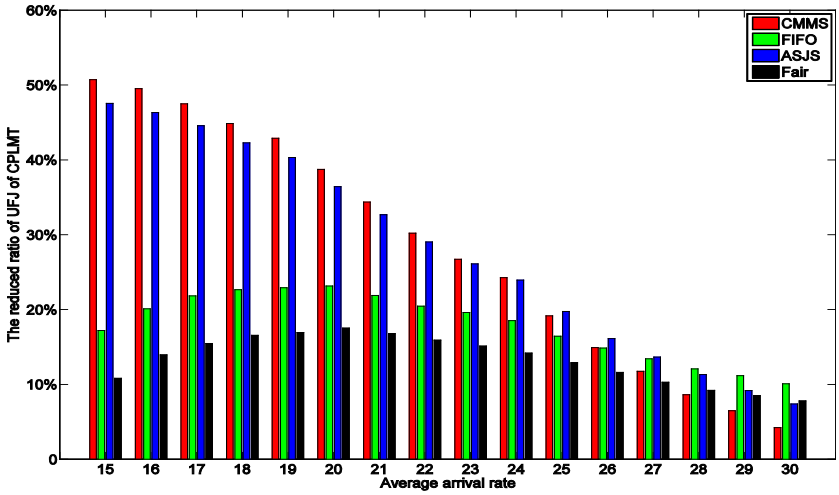

Fig. 5. The reduced rate of UFJ of CPLMT to different methods

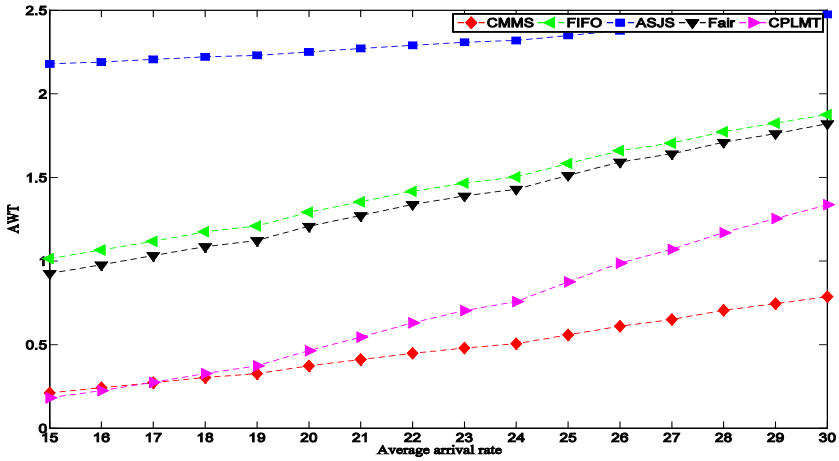

Fig. 6. The AWT of different methods

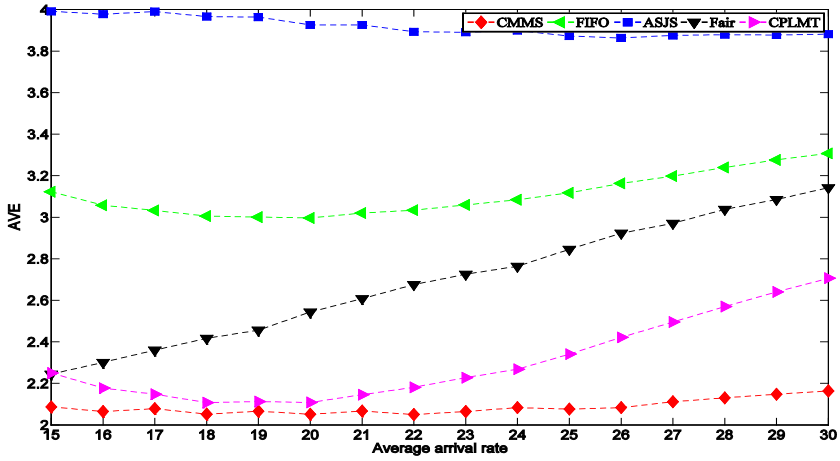

Fig. 7. The AVE of different methods

4.2 Simulation results

The parameters in our simulation are selected as Table 1. They follow uniform probability distribution. The memory of the VM is changed from 1G to 5 G. The processing ability of every VM has a range from 1 to 6 (1 is the processing ability of a standard resource). The bandwidth of every VM is a random number between 1 and 20. The memory and the bandwidth of every job have the same range with the value of VMs. The number of instructions of every task has a range of [5, 45] (5 means that a task needs 5 standard time units to execute it on a standard machine). The deadline of every job is a random integer in [1 6].

We set $\kappa = 1.5$, because if there are not enough resources for the job, the job should be executed as soon as possible (formula 7). The parallelism of jobs is a random integer number between 1 and 8. The simulation results are shown in **Figs. 4~10**. The X-axis denotes the average arrival rates of parallel jobs. It denotes the average number of jobs that arrive in a standard time unit.

Fig. 4 is the UFJ of all the five methods. Y-axis is the total number of unfinished jobs. The values of all the methods have an increasing trend with the increase of the arrival rate, because the load of the system becomes larger when the arrival rate gets large, no matter which scheduling method has been selected. CPLMT always has the smallest value of UFJ in all the methods. Fair and FIFO do not consider the requirement of the future coming jobs, so both of them have a larger value in UFJ. CMMS always scheduling the fast resource from the computing speed, so, for the coming jobs, they may have not enough resources to ensure that they can be finished as request. CPLMT always makes the urgent jobs first in the scheduling, so it saves resources and ensures finishes more jobs than others.

Fig. 5 is the reduced ratio of UFJ of CPLMT to all the other methods. Y-axis is the ratio under different arrival rates. All reduced ratios are more than 5%. To CMMS and ASJS, the reduced rate keeps dropping as the arrival rate gets larger. The values drop from 50% to 10%. To FIFO and Fair, the reduced rates get larger before arrival rate is more than 20, and then the values drop slowly. In all the cases, to CMMS, FIFO, ASJS and Fair, UFJ of CPLMT average reduces 32.73%, 19.68%, 32% and 14.14% respectively. CPLMT always schedules the "urgent" job first, so it has the smallest value in UFJ.

Fig. 6 is the AWT of all the methods and Y-axis is the value of AWT. Totally, all the value jumps when the arrival ratio becomes larger. ASJS always has the largest value of all the method. FIFO and Fair are in the middle and FIFO is more than Fair about 0.1 (time unit). CPLMT and CMMS have the same value basically when the arrival rate is less than 19, and then the AWT of CPLMT increases quickly than CMMS.

Fig. 7 is the AVE of all the methods and Y-axis is the value of AVE. The values of AVE of CMMS, ASJS and FIFO keep steady. Others have an increasing when the arrival rate is changed larger. ASJS always has the largest value of all the methods and followed by FIFO. CMMS has the smallest value of AVE of all the methods. **Fig. 8** is the reduced ratio of AVE of CPLMT to different methods. Y-axis is the value of the reduced ratio of AVE under different arrival rates. To ASJS and FIFO, CPLMT average reduces AVE more than 20% of all the cases. The value of AVE of CMMS always is less than CPLMT. When the arrival rate is a lower value, the AVE of Fair is less than CPLMT. Because Fair fits the condition when the arrival rate is a low value, under this condition, Fair shares more resources to reduce the execution time. Such as a job with the parallelism is 4, when the arrival rate is a low value, it can share 8 VMs even 12 VMs, it ensures reducing the execution time; but when the arrival rate is a high value, it does not work well, because sharing more leads to no enough VMs for the coming jobs. The AVE of CMMS is always less than the value of CPLMT, so, in the following paper, we will give more analysis between our method and CMMS.

We add a new parameter to compare CMMS and CPLMT:

$$ratt = AVER / FJR \tag{28}$$

Where,

AVER is the increased ratio of AVE of CPLMT to AVE of CMMS;

FJR is the enhanced ratio of the number of finished jobs of CPLMT to the number of finished jobs of CMMS.

If the value of *ratt* is less than 1, it means that, to CMMS, the enhanced ratio of the number of finished jobs of CPLMT is more than the increased ratio of AVE, so it worth using more time for the job. **Fig. 9** is the value of *ratt* (Y-axis) under different arrival rates. The values are less than 1 and it shows the longing execution time is worth in the scheduling. It ensures more jobs can be finished before their deadline.

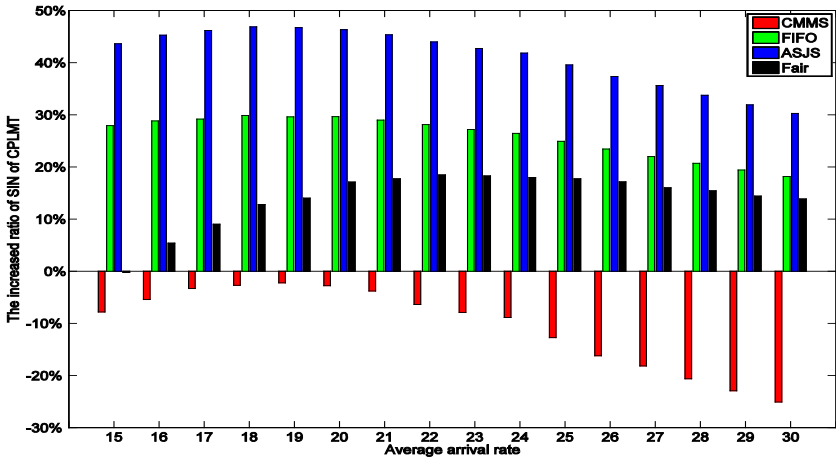

Fig. 8. The reduced rate of AVE of CPLMT to different methods

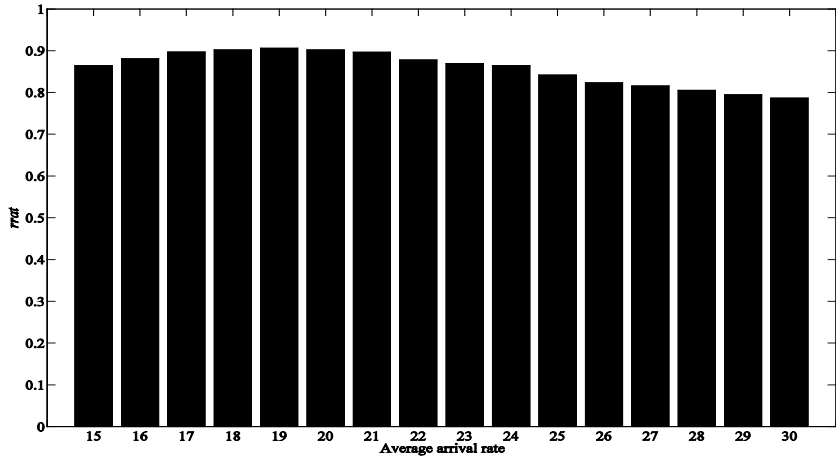

Fig. 9. The value of *rrat* of different methods

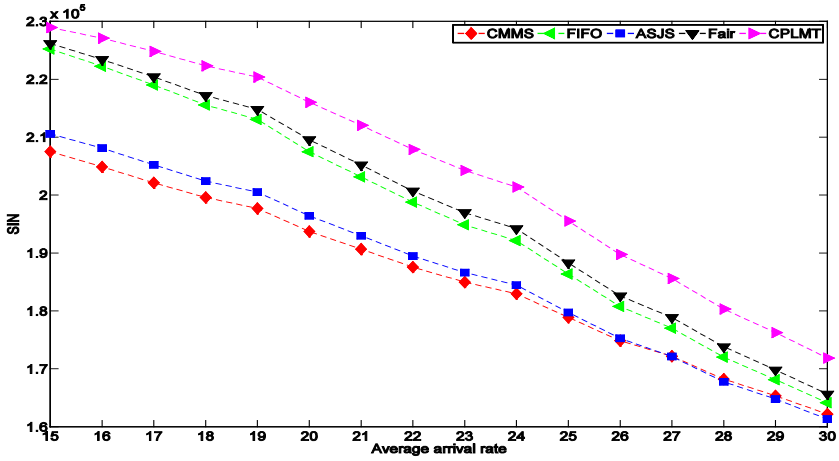

Fig. 10. The SIN of different methods

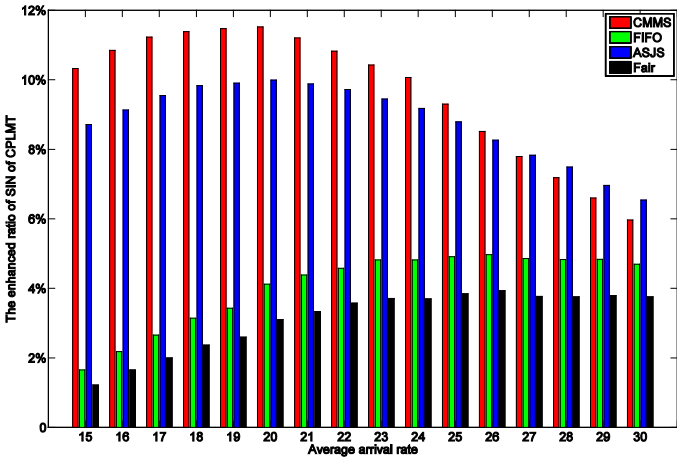

Fig. 11. The enhanced rate of SIN of CPLMT to different methods

Fig. 10 is the SIN of all the methods and Y-axis is the value of SIN. All the values of different methods drop gradually with the increasing of the arrival rate. CPLMT always has the largest value in all the five methods, and followed by Fair and FIFO. CMMS and Fair always have the smallest values in the five methods. **Fig. 11** is the enhanced ratio (Y-axis) of SIN of CPLMT to other methods. All the values are more than 1.5% in **Fig. 10**. To the value of SIN of CMMS, FIFO, ASJS and Fair, SIN of CPLMT average enhances 8.91%, 3.81%, 8.17% and 2.97% respectively.

CPLMT performances well in parallel scheduling because: (1) it takes account of the requirement of tasks; (2) it takes account of the attributes of the VM; (3) the job with only a few VMs can satisfy are scheduled first; (4) the job with shorter execution time be scheduled first. CPLMT has the lowest value in UFJ and the highest value in SIN; it also has the relative good performance in other parameters. FIFO does not take account of the details of the VMs and the job, so it has a higher value in AVE and AWT in the simulation. Fair scheduler in parallel scheduling works well when the workload of the system is in a low level, when the workload becomes larger, it does not show good performance as expectation. ASJS always tries to find the best resource, so it always longs the waiting time and it also brings the wasting of the resource, so that it leads to the increasing of AWT, AVE and UFJ. The target of CMMS is to short the execution time, so it has the shortest execution of all the methods; but at the same time, CMMS also has the lowest value in SIN and UFJ.

4.3 Analysis results in a methodological Cloud computing center

Methodological computing [18, 28] is a very important area in parallel computing. The target of our methodological computing center is to provide resources for the methodological computing for the teachers and students in our university [18]. There are many large jobs in the system and most of jobs need much time. The system is built by blade servers and blade servers are connected by a high bandwidth network. From the log of the methodological computing center, we know that:

(1) The input file size has a range from 1G to 20G.

(2) The bandwidth of the resource is changed from 1G/s to 10 G/s.

(3) The system can provide 24 cores CPU (maximum for students) or 48 cores CPU (maximum for teachers). So, the processing ability of every node is defined as 24 SP (standard machine: meaning the processing ability of a standard in a standard time unit) or 48 SP.

(4) The instruction number, the deadline and the parallelism of the job is a random in [24, 240], [1, 6] and [1, 8] respectively.

(5) Now, the load of the system is low and the average arrival rate is about 40 jobs per. time unit (day).

(6) The system can provide 120 computing nodes totally.

Most of time, the user knows that the system whether satisfies the memory of the program, so in the simulation, we also suppose the system can support enough memory for every user. We get the result of different methods when the number of jobs gets 10000.

Table 2. Simulation on Methodological Cloud

	CMMS	FIFO	ASJS	Fair	CPLMT
UFJ	3908	3096	3868	3073	2904
AWT	0.2674	0.6990	1.7414	0.7036	0.1807
AVE	3.2606	4.8396	4.6626	3.0218	3.2571
SIN(e+05)	6.6135	7.9834	6.6937	8.0382	8.1211

Table 2 Is the result of our simulation. CPLMT has the lowest value in UFJ, AWT and SIN, at the same time, CPLMT has a middle value in AVE. So, CPLMT ensures that most of the jobs being finished and keeps the execution time with a relatively low value. Because the system has a low load value, so Fair also has a good performance (as in the section 4.2, Fair will lose its advantage when the system load becomes larger).

5. Discussion

Q. Kalim et al. [25] also give the classifications of CPU-intensive jobs, Memory-intensive jobs, I/O-intensive jobs and mixed jobs. They give the classifications method which is based on the number of instructions of different operators. A task is classified as a CPU intensive task, if the job contains arithmetic operators like addition, subtraction, multiplication, and division more than 40%. For memory intensive jobs, assignment operators like equal and comparison operators are determined. If these operators occur more than 40% in the incoming 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 good for non-parallel jobs. It works but not very well for parallel tasks because of the parallelism of the job. Those definitions are not related to the resource. An example is a system where every resource has high processing ability; even a CPU-intensive job has no problems in the scheduling. If there is only one I/O-intensive task, the I/O-intensive tasks also have no problem in the read and write operations of the file. So the job needs not to be scheduled first. In this paper, taking account of the resources that can be used by the parallel tasks, we classify jobs into different kinds according to the requirement of the jobs and the QoS of VMs. In our method, a job is a CPU-intensive job (the job belongs to the set *ujobs*, see section 3) only when the number of the VMs that can satisfy the parallel tasks of a job is less than a specific ratio of the total number of VMs. Our method pays more attention to the number of VMs that ensure the job can be finished under the request of different attributes. Our method categories jobs into different kinds of urgent jobs according to the supply-demand of different attributes, this makes us more easy to decide the right scheduling order of resources. At the same time, we also consider the how to select the right resources to save wasting resource in the scheduling.

In the paper, we examine the requirements of the task include: hard-disk, memory, CPU and bandwidth. In a true system, we can consider some of them or add some other attributes in the scheduling. With the help of the classification, we schedule some jobs first, which only has a few resources can execute them as the request of the job. In this way, we try to finished more jobs. We also hope we can reduce the waiting time of finished jobs; this is the other targets of our scheduling method.

6. Conclusions and future work

This paper proposes a new scheduling method for parallel tasks based on the classification according to the attributes of the resource and the requirement of the job. Jobs are inserted to different sets according to different classification standard. Different sets have different priorities in the scheduling. Simulations from different aspects are executed to test the performance of our method. The simulation results show that our method can finish more jobs than others and at the same time, it also keeps the execution time in a lower value than others.

In the future work, we hope we can consider both performance and energy consumption at the same time [26, 27]. Energy consumption has become a hot topic in the Cloud. We hope we can find scheduling method has good performance both for the user and in the energy consumption. Job migration brings challenge in Cloud, especially for the parallel scheduling, because a migration of a task influences the task belonging to the same job. We also hope that migration can help the parallel scheduling in the big data processing platform [29] or under multi-Cloud environment [30]. This is the other problem that we try to research in the future.

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. Thanks to Dr. Yongsheng Hao (yongshenghao@yahoo.com), who helps me to improve the paper in different aspects.

References

[1] X. Qiu, Y. Dai, Y. Xiang, and L. Xing, "A hierarchical correlation model for evaluating reliability, performance, and power consumption of a cloud service," *IEEE Transactions on Systems Man & Cybernetics Systems*, Vol. 46, No.3, pp. 401-412, 2016. Article (CrossRef Link).

[2] C. Liuhua, S. Patel, S. Haiying and Z. Zhongyi, "Profiling and Understanding Virtualization Overhead in Cloud," in *Proc. Parallel Processing (ICPP), 2015 44th International Conference on, Beijing*, pp. 31-40, 2015. Article (CrossRef Link).

[3] A. Goscinski, and M. Brock, "Toward dynamic and attribute based publication, discovery and selection for Cloud computing," *Future Generation Computer Systems*, Vol.26, No.7, pp. 947-970, 2010. Article (CrossRef Link).

[4] W. Wang, Y. Jiang, W. Wu, "Multiagent-Based Resource Allocation for Energy Minimization in Cloud Computing Systems," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, Vol.47, No.2, pp. 205-220, 2017. Article (CrossRef Link).

[5] E. Filiopoulou, P. Mitropoulou, A. Tsadimas, C. Michalakelis, M. Nikolaidou and D. Anagnostopoulos, "Integrating cost analysis in the cloud: A SoS approach," in *Proc. Innovations in Information Technology (IIT), 2015 11th International Conference on Dubai*, pp. 278-283, 2015. Article (CrossRef Link).

[6] 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*, Vol. 23, No. 3, pp. 650-672, July 2015. Article (CrossRef Link).

[7] J. Paudel, J. N. Amaral, "Hybrid parallel task placement in irregular applications," *The Journal of Parallel & Distributed Computing*, Vol. 76, 94-105, 2014. Article (CrossRef Link).\

[8] W. Yi-Rong, H. Kuo-Chan, W. Feng-Jian, "Scheduling online mixed-parallel workflows of rigid tasks in heterogeneous multi-cluster environments," *Future Generation Computer Systems*, Volume 60, pp. 35-47, 2016. Article (CrossRef Link).
[9] W. Jingjin, X. Xuanxing, L. Zhiling, "Hierarchical task mapping for parallel applications on supercomputers," *The Journal of supercomputing*, Vol. 71, pp. 1776–1802, 2015. Article (CrossRef Link).
[10] L. Liu, G. Xie, L. Yang and R. Li, "Schedule Dynamic Multiple Parallel Jobs with Precedence-Constrained Tasks on Heterogeneous Distributed Computing Systems," in *Proc. Parallel and Distributed Computing (ISPDC), 2015 14th International Symposium on Limassol*, pp. 130-137, 2015. Article (CrossRef Link).
[11] H. Kuo-Chan, T.Ying-Lin, L.Hsiao-Ching, "Task ranking and allocation in list-based workflow scheduling on parallel computing platform," *The Journal of Supercomputing*, Vol. 71, No.1, pp. 217–240, 2015. Article (CrossRef Link).
[12] K. Oh-Heum, C. Kyung-Yong, "Scheduling parallel tasks with individual deadlines," *Theoretical Computer Science*, Vol. 215, No.1–2, pp. 209-223, 1999. Article (CrossRef Link).
[13] T. He, S. Chen, H. Kim, L. Tong, KW. Lee, "Scheduling Parallel Tasks onto Opportunistically Available Cloud Resources," in *Proc. 15st IEEE International Conference on Cloud Computing*, 2012. Article (CrossRef Link).
[14] K. Kurowski, , A. Oleksiak, W. Piątek, J Węglarz, "Hierarchical scheduling strategies for parallel tasks and advance reservations in grids," *Journal of Scheduling*, Vol. 16, No. 4, pp. 349-368, 2011. Article (CrossRef Link).
[15] Y. Hao, M. Xia, N. Wen, "Parallel task scheduling under multi-Clouds," *Ksii Transactions on Internet & Information Systems*, Vol. 11, No. 1, 2017. Article (CrossRef Link).
[16] Y. Xia, X. Li, Z. Shan, "Parallelized Fusion on Multisensor Transportation Data: A Case Study in CyberITS," *International Journal of Intelligent Systems*, Vol. 28, No. 6, pp. 540-564, 2013. Article (CrossRef Link).
[17] H.Ting，C. Shiyao, H. Kim, L. Tong, "To Migrate or to Wait: Bandwidth-Latency Tradeoff In Opportunistic Scheduling of Parallel Tasks," in *Proc. 31st Annual IEEE International Conference on Computer Communications: Mini-Conference*, 2012. Article (CrossRef Link).
[18] Y. Hao, L. Wang, M. Zheng, "An adaptive algorithm for scheduling parallel jobs in meteorological Cloud," *Knowledge-Based Systems*, Vol. 98, pp. 226-240, 2016. Article (CrossRef Link).
[19] L.Xiaocheng, Z. Yabing, Y. Quanjun, P. Yong , Q. Long, "Scheduling parallel jobs with tentative runs and consolidation in the cloud," *Journal of Systems and Software*, Vol. 104, pp. 141-151, 2015. Article (CrossRef Link).
[20] Rafaelli de C. Coutinho, Lúcia M.A. Drummond, Yuri Frota, Daniel de Oliveira, "Optimizing virtual machine allocation for parallel scientific workflows in federated clouds," *Future Generation Computer Systems*, Vol. 46, pp. 51-68, 2015. Article (CrossRef Link).
[21] R. S. Chang, C.-Y. Lin, and et al, "An Adaptive Scoring Job Scheduling algorithm for grid computing," *Information Sciences*, Vol. 207, p. 79-89, 2012. Article (CrossRef Link).
[22] Hadoop fair scheduler, http://hadoop.apache.org/common/docs/r0.20.1/fair_scheduler.html. Article (CrossRef Link).
[23] W. Wang, Y. Chang, and et al, "Adaptive scheduling for parallel tasks with QoS satisfaction for hybrid Cloud environments," *The Journal of Supercomputing*, Vol. 66, No. 2, pp. 783-811, 2013. Article (CrossRef Link).
[24] L. Jiayin, Q. Meikang, Mi. Zhong, Q. Gang, Q. Xiao, G. Zonghua, "Online optimization for scheduling preemptable tasks on IaaS Cloud systems," *Journal of Parallel and Distributed Computing*, Vol. 72, No. 5, pp. 666-677, 2012. Article (CrossRef Link).
[25] Q. Kalim, M. Babar, H. K. Jawad and A. M. Sajjad, "Task partitioning, scheduling and load balancing strategy for mixed nature of tasks," *The Journal of Supercomputing*, Vol. 59, No. 3, pp. 1348-1359, 2012. Article (CrossRef Link).
[26] Z. Longxin, L. Kenli, X. Yuming, M. Jing, Z. Fan, L. Keqin, "Maximizing reliability with energy conservation for parallel task scheduling in a heterogeneous cluster," *Information Sciences*, Vol. 319, pp. 113-131, 2015. Article (CrossRef Link).

[27] Y. Xia, M. Zhou, X. Luo, S. Pang and Q. Zhu, "A Stochastic Approach to Analysis of Energy-Aware DVS-Enabled Cloud Datacenters," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, Vol. 45, No. 1, pp. 73-83, 2015. Article (CrossRef Link).
[28] Y. Xia, T. Zhang T, S. Wang, "A Generic Methodological Framework for Cyber-ITS: Using Cyber-infrastructure in ITS Data Analysis Cases," *IOS Press*, 2014. Article (CrossRef Link).
[29] W. Jiayin, "Building Efficient Large-Scale Big Data Processing Platforms," *Graduate Doctoral Dissertations*, 2017. Article (CrossRef Link).
[30] Y. Hao, M. Xia, N. Wen, R. Hou, H. Deng, L. Wang, Q. Wang, "Parallel task scheduling under multi-Clouds," *KSII Transactions on Internet and Information Systems*, Vol. 11, No.1, pp. 39-60, 2017. Article (CrossRef Link).

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.

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.

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 more than 20 papers in international conferences and journals.

Yin Wang received his MS Degree of Engineering from Anhui University in 2003. Now, he is an engineer of Chizhou weather bureau, and a student of Nanjing University of Information Science & Technology. His current research interests mainly focus on the resource scheduling on different platforms.