Journal of Engineering Science and Technology Review 10 (1) (2017) 61- 67

Research Article

JOURNAL OF
Engineering Science and Technology Review

www.jestr.org

A Novel Optimization Strategy for Job Scheduling based on Double Hierarchy

Guozeng Zhao[1,*], Xiang Gao[1], Weidong Zheng[1,2] and Zhiguo Lv[3]

[1]School of Computer Science and Engineering, Luoyang Institute of Science and Technology, Luoyang 471023, China
[2]Department of Control and Computer Engineering, Politecnico di Torino, Corso Duca degli Abruzzi 24, 10129 Torino, Italy
[3]State Key Laboratory of Integrated Services Networks, Xidian University, Xi'an 710071, China

Received 4 January 2017; Accepted 27 February 2017

Abstract

At present, the high-performance cluster system has been widely applied to multitask and multi-user data processing procedures. However, computation loads can be influenced by the job scheduling optimization strategy (JSOS) of the cluster system, which can cause imbalance during job scheduling process, job starvation, and resource fragmentation. This situation can further result to problems, such as dissatisfactory resource utilization and lengthy job response, turnover, and completion times. First, a double hierarchical job scheduling model was proposed in this study to optimize the job scheduling strategy of the cluster system. Second, this study analyzed the hierarchical tasks in the scheduling model and the factors, namely, resource utilization and job completion time that influence them. The reasonability of the JSOS was also verified. Finally, the optimization strategy for job scheduling was compared with the first-come, first-served (FCFS) and FirstFit strategies. Result shows that compared with FCFS and FirstFit strategies, the proposed job scheduling strategy increased resource utilization by 6.3% and 0.8%, and reduced average response time by 22.05% and 1.12%, average turnover time by 9.82% and 2.01%, and completion time by 10.45% and 1.11%, respectively. Thus, the proposed double hierarchical job scheduling strategy not only improves system resource utilization but also reduces job response, turnover and job completion times. The experimental result is consistent with the expected requirements, and the study provides a feasible scheme for the job scheduling optimization problem in the cluster system.

Keywords: Cluster, Scheduling Strategy, Double Hierarchy, Optimization Strategy

1. Introduction

Along with the development of computer technology and communication technology, the high-performance cluster system has been widely applied in large-scale data processing. Specifically, the cluster system is oriented for multitask and multi-user massive job and data processing. To improve system performance and obtain the best job scheduling effect, jobs are allocated in the queue to the computation unit through an optimal strategy, which is based on job characteristics. Specifically, job scheduling is one of the core functions of the cluster system [1]. The rationality of a job scheduling strategy directly influences quality of service (QoS) of users and system performance [2]. Therefore, effectively optimizing the job scheduling strategy and the computation capability of the cluster system is a significant problem that should be urgently solved.

Job scheduling strategy is an indispensable part of network computation. This strategy receives a user's job request, ranks the job queue according to job characteristics, selects suitable resources from the global resource pool, allocates reasonable resources for the jobs, and monitors job execution. Generally, the job scheduling system should select suitable jobs for operation and suitable nodes for the job operation, and allocate the necessary system resources. Fortunately, along with the rapid development of computer hardware technology and system structure, multicore processors have been gradually developed and widely applied to data processing. In the cluster system, multicore and many-core processors are configured for a single computation resource node to establish a multipath and multicore hybrid structure with shared memory. Integrating resource characteristics in the cluster system with heterogeneous computation nodes, excavating the parallel granularity of the computation nodes, balancing the job loads of the heterogeneous resource nodes, and utilizing cluster system performance are all significant for the job scheduling system [3].

Accordingly, the job scheduling problem of the cluster system was abstracted into an optimization model composed of a job scheduling and distribution layers. The key factors that influence job scheduling were analyzed, and job priority was dynamically adjusted to utilize data resources maximally. To reduce user waiting time and improve resource utilization and system performance, resource occupancy information was adopted to distribute the job queues to the resource units evenly. This study aims at finding the job scheduling optimization strategy (JSOS) suitable for high-performance cluster system.

2. State of the Art

The unbalanced distribution of resource nodes during the job scheduling process of the cluster system causes low system resource utilization and lengthy job completion. Thus, job

E-mail address: ly_zgz@163.com

scheduling modeling and system parameter optimization have been conducted by several scholars to find an optimal job scheduling strategy [4]. Job scheduling strategy is related to the cluster system performance; an excellent job scheduling strategy can maximally reduce system resource competition and user cost, improve system resource utilization, and reduce job completion time. Evidently, finding an optimal scheduling strategy is an NP-complete problem.

At present, the frequently used scheduling strategies in the cluster management system mainly include first-come, first-served (FCFS) scheduling strategy, scheduling strategy based on priority, FirstFit strategy, and BestFit strategy. FCFS is the most common classical scheduling strategy [5]. This strategy is aimed at scheduling jobs according to arrival sequence. FCFS can ensure job scheduling fairness; however, it causes several idle computation nodes, especially when a job with large resource occupancy is ranked at the head of the queue. The waiting time of subsequent jobs is obviously prolonged, thus reducing system resource utilization and throughput rate. To overcome this disadvantage, a short job priority scheduling strategy [6] was proposed; however, this method might delay a large job to arrive at the earliest time, which can cause job scheduling unfairness. The FirstFit strategy aims to evaluate jobs in the queue and then scheduling the first job with available resources, whereas the BestFit strategy aims to determine the job with the maximum resource satisfaction in the present system [7]. However, these two strategies might delay the jobs with high resource demand due to the execution of the jobs with low resource demand, cause job starvation, or increase the average waiting time of the jobs. The scheduling strategy based on priority aims to execute jobs in the queue according to the job priority defined in the system; job fairness is considered in this method. However, when the system fails to provide sufficient resources for jobs with high priority, the system cannot run the job with low priority, thus causing idle resources and job starvation [8].

Along with the development of cluster technology, the job scheduling technology is also continuously improved to maximize resource utilization, reduce job waiting time in the cluster system, and gradually develop certain advanced strategies accordingly. For example, Mehta et al. proposed a time-delay dynamic load balancing model based on feedback control theory [9]. The authors adopted a discrete event to simulate a time-delay load balancing system through a provided computation method of optimal load balancing gain. For parallel files with large communication time delay, they did not do an in-depth analysis of the influence of the time-delay factors on information accuracy. Guobin Zhang et al. researched the combination of the priority scheduling strategy and backfilling scheduling strategy [10]. For a queue with mainly small jobs, this strategy is a good supplementation to the priority strategy; however, if a job with high resource demand is ranked ahead of the queue without any resource appointment, then the large job would be pending for a long time, causing job starvation. Guotao Zhang et al. comprehensively considered the present load of a node based on traditional backfilling algorithm; they proposed a scheduling algorithm that integrates appointment and backfilling strategies on account of the time slot between the job operation resource and the appointed resource [11]. This strategy could have improved resource utilization; however, the FCFS strategy provided by the scheduler was adopted for the queuing algorithm during job submission, thus, job sequencing selection function was unavailable. Moreover, the authors adopted the FirstFit strategy for resource scheduling; thus, although the queue included several small jobs, the degree of job urgency could not be distinguished well when the job ahead of the queue was blocked [12]. Shuren Bai et al. considered the product of CPU quantity and idle time as job submission filling condition based on backfilling strategy [13]. To meet the allowable requirements of the cluster system, this method was used to reduce the CPU core number of the job while increasing the CPU computation time when the job filling condition exceeded the actual available condition of the cluster system. Compared with the backfilling algorithm, the method was an active filling scheduling algorithm; but the author only verified the algorithm theoretically rather than analyzed the actual job scheduling request [14]. According to previous research, the existing job scheduling strategies fail to consider the application characteristics comprehensively and cause problems, such as job starvation, resource fragmentation, and unfairness. Moreover, the existing task distribution methods fail to analyze job complexity or consider the influence of multithread and multi-process execution. Thus, these methods insufficiently utilize computation resources, causing load imbalance and increasing overall job completion time.

Therefore, data scheduling and distributing processes were comprehensively considered in this study. Specifically, job priority and resource occupancy mechanism were dynamically adjusted to ensure fairness in job scheduling. Meanwhile, system resource occupancy information was obtained according to feedback concept through the task distribution mechanism. Thus, the job sequence was evenly distributed to the resource units to maximally optimize the job scheduling effect.

The remainder of this study is organized as follows. Section 3 establishes the job scheduling model based on double hierarchy for the job scheduling problems in the cluster system. Job scheduling and task distribution processes, as well as the evaluation indexes of job scheduling performance, are also described in Section 3. Section 4 verifies the effectiveness of the proposed algorithm through the conducted experiment. Section 5 presents the conclusion of this study.

3. Methodology

3.1 Double hierarchical job scheduling model

The job scheduling model includes user, job scheduling, task distribution, and resource node layers. Herein, the user layer was used for submitting job requests. The user set was assumed as $U_i(i=1,2,...,u)$. The job submitted by user U_i was assumed as $J_i(i=1,2,...,m)$. The sub-task included in each job J_i was assumed as $F_i(i=1,2,...,f)$. The resource node requested was assumed as $R_i(i=1,2,...,n)$. In the system, a user's job was formalized into a quintuple group $<U_i,J_i,F_i,R_i,P_i>$, where P_i is the scheduling priority allocated by the system to the user. The resource node layer was used to run the data job submitted by the user and return the corresponding result.

3.1.1 Job scheduling layer

The job scheduling layer was used to receive jobs from the user, inquire the present idle system resources according to the resource requirements of the user, and schedule jobs according to a certain scheduling strategy. According to the analysis of job characteristics, a job was composed of a group of independent sequences and could be divided in parallel; thus, the dynamically adjustable job priority and the resource occupancy mechanism were adopted to ensure fairness. Hence, any job could not delay the execution of any other job with higher priority. The purpose of the appointment concept was to ensure the maximization of the resource utilization, and preferentially allocate the idle resources to jobs with low priority and low resource demand, as well as reserve the corresponding resources for jobs with high priority and high resource demand. The steps of the job scheduling layer are described as follows.

Step 1: Rank the jobs to be scheduled from high to low priority. Then, select the job ahead of the ranked queue and allocate the corresponding quantity of resources needed by this job.

Step 2: Inquire the present idle resources.

Step 3: If the idle CPU can meet the resource demand of this job (namely, the quantity of idle CPU is greater than the quantity of CPU needed by this job) then, allocate the idle CPU to this job.

Step 4: If the quantity of the idle CPU of the present system is less than the quantity of CPU needed by this job, then judge whether the sum of the present idle CPU and the reserved CPU can meet the resource demand of this job.

Step 5: If the reserved CPU can meet the resource demand, then select the corresponding jump job according to the selection strategy of the suspended job; if a suitable jump job is available, then suspend the jump jobs with low priority and set the appointed CPU to run the jump jobs as idle state. Subsequently, allocate all idle CPU to the present job waiting to be scheduled, wherein the completed parts of the suspended jump jobs will be recorded and saved in the system, and the job state thereof will be set as scheduling waiting state.

Step 6: If the sum of the idle and reserved CPUs cannot meet the resource demand, then schedule the next job selected from the job queue waiting to be scheduled, and repeat Step 2.

This way, the job with low priority can temporarily occupy the resource reserved for the job with high priority and is accordingly executed in advance. Meanwhile, the job with high priority will not be delayed for the execution of the job with low priority. When the idle CPU is allocated to a job, the job will be decomposed into sub-tasks at the task distribution layer and then the sub-tasks will be evenly allocated with system resources.

3.1.2 Task distribution layer

At the task distribution layer, a job was reasonably divided into sub-tasks, which were evenly allocated with the corresponding data resources. The above process was completed as follows: the job sequence was allocated to the resource units; finally, the remaining part, which could not be exactly divided by the resource units, was allocated to the idle resource units according to the division strategy.

At the task distribution layer, resource granularity was initially divided, and one CPU core was considered as the basic unit of resource granularity. According to Reference [15], in allocating the computation resource for each running program under the multicore and multiprogram environment, the feedback concept was adopted to obtain the resource occupancy information of the system. Thereafter, accurate resource occupancy could be obtained through the feedback information mentioned. The granularity of the resource units was divided considering the memory usage. If the threads running on all resource units of a resource node exceeded the physical memory capacity, the system would employ an exchange partition to ensure that the processes were running normally; however, such frequent data exchange with the magnetic disk would also bring significant additional overhead, thus greatly increasing data time. Therefore, in the data unit division strategy, the sum of the memory occupied by all task processes running on the resource units should not exceed the physical memory capacity of the resource nodes; moreover, the sum of all threads running on the resource units should not exceed the core number of the processors of the resource nodes.

On this basis, the job scheduling problem was modeled as a bag-of-tasks (BoT) application model [16]. In the BoT model, a job could be decomposed into multiple independent tasks without data independence or communication constraint. A loosely coupled BoT universal model included one master and multiple slaves, wherein the master was used for task allocation, whereas the slaves were used for receiving tasks, executing computation operation, returning the corresponding result to the master, and waiting for the next task allocation.

Thus, the user submits the job to the cluster system by uploading the job file and configuring the relevant parameters. Then, the job submitted by the user is placed by the management node into the scheduling queue. The node is applied and allocates the job according to the resource requirements of the user. The job is divided into independent tasks to be distributed to the resource nodes for parallel processing according to a certain strategy. After receiving the distributed tasks, the engine running on the node starts task execution. After task completion, successful processing information is returned by the resource node to the management node and finally to the terminal user.

3.2 Performance evaluation index

To better evaluate and quantify the performance of the job scheduling strategy, the performance evaluation indexes were defined in Reference [17].

3.2.1 Definition 1

Resource utilization, namely CPU utilization, represents the busy-idle degree of system resources and includes real-time and average resource utilizations. Herein, the real-time resource utilization of the system is assumed as P_r, and the average resource utilization within a time interval of T is assumed as P_{r_T}. Specifically, the real-time resource utilization can be expressed as follows:

$$P_r = \frac{N_b}{N} \tag{1}$$

where N_b represents the quantity of running CPU, and N represents the quantity of all CPUs in the system.

The average resource utilization from T_a to T_b can be expressed as follows:

$$P_{r\_T} = \frac{1}{T_a - T_b}\int_{T_a}^{T_b} P_r(t)dt \tag{2}$$

When $n \to \infty$ is true, the usage time is divided into m areas, and the average resource utilization can be expressed as follows:

$$P_{r\_T} = \frac{1}{m}\lim_{\Delta t \to 0}\sum_{i=1}^{m} P_r\left(i \cdot \frac{T_a - T_b}{m}\right) \tag{3}$$

Accordingly, the average value of the real-time resource utilization was adopted to represent the resource utilization approximately within a certain period.

3.2.2 Definition 2

Job response time or job waiting time, is defined as the job's duration from the initial time to the first start time.

The initial time of a certain job J_i is assumed as $TJI(i)$; the start time is assumed as $TJS(i,k)$; and the end time is assumed as $TJF(i,k)$,where k represents the quantity of CPU needed. Additionally, $TJres(i,k)$ represents the response time of job J_i occupying k CPU resources, and $TJave\_res(n)$ denotes the average response time of n jobs. Therefore, the following results can be obtained:

$$TJave(i,k) = TJS(i,k) - TJI(i,k) \tag{4}$$

$$TJave\_res(n) = \frac{1}{n}\sum_{i=1}^{n} TJave\_res(i) \tag{5}$$

3.2.3 Definition 3

Job turnover time refers to the duration between the initial and completion times, and includes execution and response times.

$TJturn(i,k)$ is assumed as the completion time of job J_i occupying k CPU resources, and $TJave\_turn(n)$ is assumed as the average turnover time of n jobs. Thus, the following results can be obtained:

$$TJturn(i,k) = TJF(i,k) - TJI(i,k) \tag{6}$$

$$TJave\_turn(n) = \frac{1}{n}\sum_{i=1}^{n} TJturn(i) \tag{7}$$

3.2.4 Definition 4

Overall job completion time refers to that of all jobs in the job queue. Overall job completion time $TAJF(n)$ of all n jobs in the job queue can be expressed as follows:

$$TAJF(n) = \text{MAX}(TJF(i,k)) - \text{MIN}(TJI(i)) \tag{8}$$

To utilize the multi-core computation resources fully and improve system rendering efficiency and throughput rate, the multi-data job submitted by the user was divided into multiple sub-tasks. Each sub-task corresponded to a data process for independent operation. The sub-tasks were independent of each other and no communication overhead was needed among multiple sub-task processes, so the additional time overhead was generated by processor switching, IO competition and memory wall during multi-process execution on the same server. As a result, the relationship between the time and the process count could be modeled as follows:

$$T_{pt} = \frac{T_{1t} + T_s + T_{io} + T_m}{p} \tag{9}$$

where T_{pt} represents the job completion time of the processes of concurrent p t-threads, T_s is the switching overhead of the processers, T_{io} denotes IO overhead, T_m represents the magnetic disk switching overhead under memory insufficiency, and T_{lt} is the job completion time of one t-thread. Specifically, the corresponding model is as follows:

$$T_{1t} = (1 - P + P/t)T_o \tag{10}$$

where T_o represents the completion time of a process of a single thread, and P is the proportion of the parallel part.

4. Result Analysis and Discussion

4.1 Experimental environment

In this study, a Lenovo System x3950 X6 high-performance server of the network experiment center of the Luoyang Institute of Science and Technology was adopted as the experiment equipment. The system configuration was as follows: Intel Xeon E7 processor and dominant frequency of 2.4 GHz. Gigabit internet was adopted for the nodes. To evaluate the performance of the scheduling strategy proposed in this study, the representative test case used by Patoli et al. in Reference [18] was adopted to verify the effectiveness of the proposed strategy experimentally.

The proposed strategy has a dynamically adjustable priority, so the job priority of the test case is correspondingly divided into three types of priorities, namely jobs with high, medium, and low priorities. Different jobs have different resource demands, which are defined as follows: a job needing 7 or more CPU resources with high resource demand; a job needing 3 or less CPU resources with low resource demand; other jobs have medium resource demand. In order to test the performance of the job scheduling strategy proposed in this paper, a test case which includes 200 jobs is adopted in this paper, where each group of jobs has different priorities and resource demand proportions. Table 1 shows a total of six groups of jobs.

Table 1. Job Test Set

No.	Priorities			Demands		
	High	**Medium**	**Low**	**High**	**Medium**	**Low**
Js1	30%	30%	40%	40%	30%	30%
Js2	30%	30%	40%	30%	50%	20%
Js3	30%	40%	30%	40%	30%	30%
Js4	30%	40%	30%	30%	50%	20%
Js5	40%	30%	30%	40%	30%	30%
Js6	40%	30%	30%	30%	50%	20%

Multiple groups of experiments were designed in this study to verify the double hierarchical job scheduling strategy. The six groups of data jobs presented in Table 1 were submitted to the data management platform upon completing the previous group of jobs. During job operation, FCFS and FirstFit strategies, and the proposed JSOS were adopted for algorithm performance comparison. Meanwhile, multi-user jobs were adopted to test the effectiveness of the double hierarchical job scheduling strategy.

4.2 Experimental result and analysis
The JSOS based on double hierarchy, FCFS and FirstFit strategies were compared with each other in resource utilization, job turnover time, overall job completion and average response time.

4.2.1 Comparison of resource utilization
First, resource utilization was tested, wherein each group of jobs had different priorities and resource demand proportions. According to the resource utilization index in Formula (3), Fig.1 shows the average resource utilizations of the FCFS and FirstFit scheduling strategies and the proposed JSOS during job set scheduling.

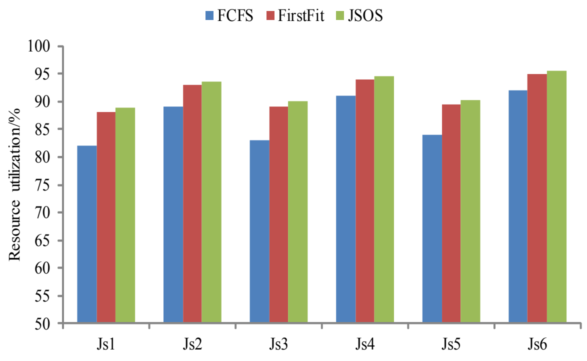

Fig.1. Comparison of Resource Utilizations

As shown in Fig.1, the proposed strategy has better performance and higher resource utilization compared with the other two strategies. According to data comparison, the proposed double hierarchical JSOS has increased resource utilization by 6.3% and 0.8% on average, compared with the FCFS and FirstFit scheduling strategies, respectively. Along with the increase of the job resource demand and the proportion of jobs with high priority, the proposed strategy could obtain higher resource utilization for the following reasons: (1) FCFS scheduling strategy aims to schedule the jobs according to job arrival sequence without considering the job resource demand, thereby blocking the jobs with high resource demand due to resource insufficiency and causing resource waste; and (2) FirstFit scheduling strategy aims to execute the jobs with low resource demand in the job set with the same priority, thus probably making the jobs with low priority and low resource demand unable to occupy fully the resource interspace brought by resource allocation and causing a small amount of idle data resources. Essentially, job priority and job resource demand were comprehensively considered in the proposed strategy. In the case of insufficient resource for the job with high priority and high resource demand, the system would initially schedule one or more data jobs with low priority and satisfactory resource demand according to the scheduling strategy. Meanwhile, the advanced scheduling operation of these jobs with low priority would not delay the execution of the jobs with high priority and high resource demand to ensure fairness. This way, the idle resources generated by resource blockage could be comprehensively utilized to improve the overall resource utilization of the system.

4.2.2 Comparison of average response time
According to the average job response time index in Formula (5), Fig.2 shows a comparison of the average job response times of the proposed JSOS, and FCFS and FirstFit scheduling strategies. According to quantitative computation, the proposed JSOS reduced the average job response time by 22.05% and 1.12% compared with the FCFS and FirstFit scheduling strategies, respectively.

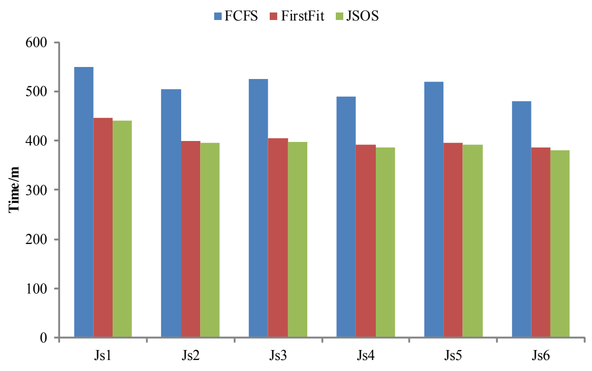

Fig.2. Comparison of Average Response Times

Compared with traditional FCFS and FirstFit scheduling strategies, the proposed JSOS has a shorter response time. Job priority and job resource demand were considered for the test job set in this study. However, in the job scheduling process of the FCFS strategy, the job set might cause job starvation due to insufficient system resources, and the idle resource generated may increase the job completion time, as well as the response time of the unscheduled job with low priority. Moreover, in the FCFS strategy, the jobs were scheduled according to their arrival sequence. A job with high priority arriving at an earlier time might be delayed to wait for system resources, but the job with low priority might also have an increased response time due to the abovementioned delay. Compared with the FirstFit strategy, the proposed strategy could fully utilize system resources to reduce job completion and response times of subsequent jobs to be scheduled. Thus, the proposed strategy had a shorter average response time.

4.2.3 Comparison of turnover time
According to the average job turnover time index in Formula (7), Fig.3 shows the comparison of the three strategies in their average job turnover times. According to quantitative computation, the proposed JSOS reduced the average job turnover time by 9.82% and 2.01% compared with the FCFS and FirstFit scheduling strategies, respectively.

For the general comparison in average turnover time, the FCFS job scheduling strategy was adopted for different data jobs; however, the difference between the priority proportions and resource demand proportions of different job sets was not considered in this strategy. Thus, the resources might not be fully utilized due to job starvation and resource fragmentation, and the jobs might wait for a long time before being scheduled due to resource blockage, thereby increasing the turnover time of most jobs. For the FirstFit job scheduling strategy, system resource matching was considered as the scheduling principle to avoid the resource blockage caused by system resource insufficiency; however, idle resources might be generated in the resource matching process. Thus, the resource interspace could not be fully utilized. Nevertheless, the proposed JSOS not only avoided the idle waiting state of the idle resources caused by the blockage of the jobs with high priority and high resource demand, but also adopted the jobs with low priority and high idle resource matching degree to "fill" the idle resources. This way, resource utilization was improved, and each job could be operated in advance, thereby preventing them from entering the scheduling waiting state for jobs with low priority. Evidently, jobs with low priority and scheduled in advance would actively release the data resources when the system resources were available for jobs with high priority. This strategy also conformed to the fairness principle. Such an advanced jump execution mode could reduce the average job turnover time.

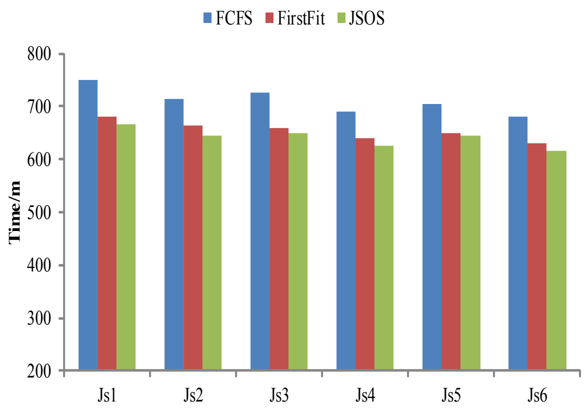

Fig.3. Comparison of Average Turnover Times

4.2.4 Comparison of overall job completion time

According to the overall job completion time index in Formula (8), Fig.4 shows the comparison of the three strategies in the overall job completion time. According to the quantitative computation, the proposed JSOS reduced average job turnover time by 10.45% and 1.11% compared with the FCFS and FirstFit scheduling strategies, respectively.

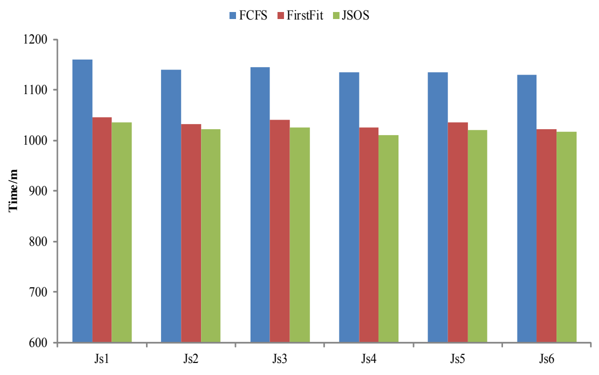

Fig.4. Comparison of Overall Job Completion Times

Overall job completion time is one of the key indexes considered by the users for the performance of the data management system. Traditional FCFS job scheduling strategy could ensure absolute fairness; however, it failed to comprehensively consider the improvement of the system performance. Compared with FCFS strategy, FirstFit scheduling strategy could increase system throughput by readjusting the scheduling sequence of the job scheduling queue; however, it failed to consider fairness comprehensively. Nevertheless, the proposed JSOS was based on job priority, and such absolute fairness was disrupted only due to job blockage to make the job with high priority and high resource demand temporarily enter the waiting state while scheduling the job with low priority and low resource demand in advance. Specifically, the proposed scheduling strategy was designed from the perspective of system performance to accelerate the job scheduling process, reduce idle time of resources, and fully utilize the present idle system resources.

For the multi-job case, six groups of job test sets with different priority proportions and resource demands were adopted to compare the proposed algorithm with existing FCFS and FirstFit scheduling algorithms. The double hierarchical JSOS not only improved system resource utilization but also reduced job response, job turnover, and job completion times.

5. Conclusions

To solve low resource utilization and lengthy job turnover time in the cluster system, the job scheduling process and the system task distribution of the cluster system were initially analyzed in this system. Then, the scheduling algorithm based on the dynamic adjustment of job priority and occupancy mechanism was adopted to describe the influence of job characteristics on system resource utilization. Subsequently, to establish the job allocation model, a feedback concept based on the basic job distribution strategy was combined to obtain resource occupancy information. Finally, the following conclusions were obtained:

(1) The dynamically adjustable job priority not only reduces average job waiting time but also ensures job scheduling fairness.

(2) Resources can be reserved for jobs with high priority and high resource demand by allowing the job with low priority and low resource demand to be scheduled in advance, thus avoiding job starvation and resource fragmentation.

(3) Division based on a single CPU core as the resource granularity unit, and the application of the resource occupancy information and resource distribution strategy not only balances resource loads but also improves resource utilization.

In conclusion, the proposed double hierarchical job scheduling model can improve system resource utilization and QoS for the users, thereby providing a feasible solution for the job scheduling optimization problem in the cluster system. However, in an actual cluster system, system resources have significantly different performance due to the continuous upgrade of the hardware resources of the server. Such performance difference may influence job completion time and resource utilization. Therefore, in the future, we should focus on evaluating resource performance reasonably, and guiding high-efficiency job scheduling process according to system performance and job loads.

References

1. Lee, Y. C., Zomaya, A. Y., “Energy Conscious Scheduling for Distributed Computing Systems under Different Operating Conditions”. *IEEE Transactions on Parallel and Distributed Systems*, 22(8), 2011, pp. 1374-1381.
2. Mezmaz, M., Melab, N., Kessaci, Y., et al., “A parallel bi-objective hybrid metaheuristic for energy-aware scheduling for cloud computing systems”. *Journal of Parallel and Distributed Computing* , 71(11), 2011, pp.1497-1508.
3. Quezada-Pina, Ariel., Tchernykh, A., González-García, J. L., et al., “Adaptive parallel job scheduling with resource admissible allocation on two-level hierarchical grids”. *Future Generation Computer Systems* , 28(7), 2012, pp.965-976.
4. Zhou, Q., Liu, R., “Strategy optimization of resource scheduling based on cluster rendering”. *Cluster Computing*, 19(4), 2016, pp.1-9.
5. Xu, X., Cao, L., Wang, X., “Adaptive Task Scheduling Strategy Based on Dynamic Workload Adjustment for Heterogeneous Hadoop Clusters”. *IEEE Systems Journal* , 10(2), 2014, pp.471-482.
6. Tamm, G., Krüger, J., “Hybrid Rendering with Scheduling under Uncertainty”. *IEEE Transactions on Visualization and Computer Graphics*, 20(5), 2014, pp. 767-780.
7. Torkestani, J. A., “A new approach to the job scheduling problem in computational grids”. *Cluster Computing*, 15(3), 2012, pp.201-210.
8. Ghanbari, S., Othman, M., “A Priority Based Job Scheduling Algorithm in Cloud Computing”. *Procedia Engineering*, 50(9), 2012, pp.778–785.
9. Mehta, M., Jinwala, D., “A Hybrid Dynamic Load Balancing Algorithm for Distributed Systems”. *Journal of Computers*, 9(8), 2014, pp.1825-1833.
10. Zhang, G. B., Pan, J. G., “Design and Analysis of Prority-based Preemtiv Parallel Scheduling Algorithm”. *Computer Science*, 34(7), 2007, pp.279-281.
11. Zhang, G. T., Zhao, J. Y., Bai, Z. Y., “Cluster Job-scheduling System Based on LT-backfilling Algorithm”. *Computer Engineering*, 33(21), 2007, pp.69-71.
12. Qureshi, K. Majeed, B., Kazmi, J. H., et al., “Task partitioning, scheduling and load balancing strategy for mixed nature of tasks”. *The Journal of Supercomputing*, 59(3), 2012, pp.1348-1359.
13. Bai, S. R., Yun-Hong, F. U., “An Algorithm for BACKFILL-Based "Take Ten into Five" Parallel Job Scheduling”. *Journal of Hunan University* (*Natural Sciences*), 34(1), 2007, pp.81-84.
14. Chandio, A. A., Bilal, K., Tziritas, N., et al., “A comparative study on resource allocation and energy efficient job scheduling strategies in large-scale parallel computing systems”. *Cluster Computing*, 17(4), 2014, pp.1349-1367.
15. Maggio, M., Hoffmann, H., Agarwal, A., et al., “Control-theoretical cpu allocation: Design and implementation with feedback control”. *Proceedings* the 6th *International Workshop on Feedback Control Implementation and Design in Computing Systems and Networks*, Karlsruhe, Germany: ACM, 2011, pp.81-130.
16. Bertin, R., Hunold, S., Legrand, A., et al., “Fair scheduling of bag-of-tasks applications using distributed Lagrangian optimization”. *Journal of Parallel and Distributed Computing*, 74(1), 2014, pp. 1914-1929.
17. Hsu, W. H., Wang, C. F., Ma, K. L., et al., “A Job Scheduling Design for Visualization Services Using GPU Clusters”. *2012 IEEE International Conference on Cluster Computing*, Beijing, China: IEEE, 2012, pp. 523-533.
18. Patoli, M. Z., Gkion, M., AI-Barakati, A., et al., “An open source Grid based render farm for Blender 3D”. *2009 IEEE/PES Power Systems Conference and Exposition*, Seattle, WA, USA: IEEE, 2009, pp.1-6.