JOURNAL OF INFORMATION SCIENCE AND ENGINEERING 33, 463-483 (2017)

DOI:10.1688/JISE.2017.33.2.11

An Efficient Two-Level Hierarchy Job Scheduling and Task Dispatching Strategy for Cluster Rendering System*

QIAN LI, WEI-GUO WU, ZE-YU SUN AND JIAN-HANG HUANG
Department of Computer Science and Technology
Xi'an Jiaotong University
Xi'an, 710049 China
E-mail: {qian.l; wgwu}@stu.xjtu.edu.cn; lylgszy@163.com; huangjhsx@gmail.com

When high performance computing platforms are widely used for rendering application, scheduling of rendering jobs is crucial due to job starvation and resource fragmentation problems in the existing scheduling strategies. Meanwhile, these strategies rarely consider a tradeoff with fairness and performance. Moreover, the traditional task-centered assignment method and naïve load balancing strategy cannot make full use of rendering resources. Aiming at solving these issues, an efficient two-level hierarchy job scheduling and task dispatching strategy (RF-FD) for cluster rendering system is proposed. Specifically, the reservation-based FirstFit (RF) job scheduling strategy and feedback-based tasks distribution (FD) strategy are integrated to obtain the maximization of system performance and load balancing. The RF strategy uses the ideology of reservation on low-priority jobs to maximize the resources utilization rate when the jobs are blocked. And the FD strategy uses feedback of resource usage to choose an appropriate number of threads for the renderer and then divides rendering nodes into fine-granularity rendering units to balance load distribution. With different evaluation metrics, the experimental results demonstrate that the proposed strategy outperforms the existing scheduling strategies combining with the fixed threads and naïve load balancing method while guaranteeing the fairness.

Keywords: cluster scheduling strategy, tasks assignment, rendering, resource reservation, feedback, load balancing

1. INTRODUCTION

With the development of high performance computing (HPC), various scientific organizations have spent ample budget to carry out research projects using the supercomputers [1]. Currently, people focus on how to manage and use these resources in an effective way and what application could be deployed on these HPC platforms.

Rendering is the process of creating an image from a model by means of computer programs and a very complex and time-consuming computation [2]. Due to the independence among image frames, intensive computation and massive data access, rendering is specifically suitable for the parallel processing in a cluster computing environment. A render farm is a cluster of interconnected computers which are employed to rendering images in parallel [3]. Because the rendering of frames and tiles is a highly-parallelized work by nature, the rendering time can be greatly reduced through the render farm. The render farm actually consists of a pool of finite rendering resources under the demands of render users. Therefore, there will be a conflict if the user's requests exceed the availa-

Received December 10, 2015; revised May 29, 2016; accepted June 25, 2016.
Communicated by Sheng-Wei Chen.
* This work is supported by the National Natural Science Foundation of China under Grant No. 61202041, No. 91330117 and the 863 High-Tech Program under Grant No. 2012AA01A306, No. 2014AA01A302.

bility of these resources.

For a render farm, fulfillment of the demand for all render users and utilizing adequately the tremendous capabilities of rendering systems is of far greater importance. An efficient and optimal scheduling of jobs and distribution of resources is urgently needed to improve the overall performance of the system [4, 5]. Scheduling jobs in HPC systems is always an important issue and various scheduling policies can result in different user experiences and resource utilizations. A scheduling process involves assigning resources to jobs in specific time intervals such that the capacity of resources should meet jobs' needs and no other jobs access the resources at the same time interval [6, 7]. However, due to the dynamic nature of the workloads, the scheduling problem is hard to solve [8].

Traditional rendering cluster system usually adopts the priority-based non-preemptive scheduling policies [9], *i.e*., the low-priority jobs would be executed after the high-priority jobs finish. But the high-priority jobs with more resource demands may delay the execution of the following low-priority jobs when the system lacks resources, which leads to resource fragmentations [10], decline of resource utilization rates and the extension of response time.

FirstFit and BestFit algorithm are known for its capability to promote job response time and throughput of the system, but they are unable to handle the job "starvation" problem [11] and lead to unfairness. The fairness [12] means that no job is delayed by any low-priority jobs. For example, FCFS produces better results with respect to fairness, but FCFC is not satisfactory in terms of job queue time, response time and resource utilization. Backfilling [6, 13] was proposed to help improve system utilization and have been implemented in most production schedulers [14, 15]. The backfilling policy needs the runtime prediction of each job and then the suitable backfilling job will be chosen. A user estimation for rendering job executing time may be used, but most of the estimations are incorrect [13] and the less knowledgeable is the user, the more prone is his estimates to error [16]. Furthermore, the complexity of frames makes it difficult to predict the time, and also the prediction accuracy cannot be guaranteed.

Additionally, each rendering job with multi frames is divided into a quantity of tasks, and these tasks are assigned to the idle resources whose quantity is specified by the users. Various dispatching policies for the frames also influence rendering time of the single job and load balancing among rendering nodes due to different rendering time of the frames. Traditional inter-frame dispatching defines each frame's rendering as a task, and intra-frame dispatching divides each frame to a finer-granularity level. These rendering resources accept tasks and run multi-thread renderers. Although multi-thread can accelerate the rendering process, the communication and synchronization among threads may lead to poor utilization of resources and low throughput of the system.

In short, it is not a trivial task to share a render farm among multiple users with high efficiency. Although many sophisticated job scheduling techniques have been developed to improve the efficiency of systems with multi-user support, most of current approaches found in literatures are not directly applicable to render farm and do not consider the characteristics of independence among different rendering jobs and the temporal coherence among frames. Therefore, it is significant to design a policy of scheduling rendering jobs for promoting system performance and a task assignment method with the objective of utility maximization in a rendering farm.

This work concentrates on optimizing of scheduling and dispatching for rendering

jobs at a B/S architecture platform, where all jobs submitted by different users are initially placed on the management node and then are to be dispatched to the rendering nodes which actually execute the calculating. An efficient two-level hierarchy job scheduling and task dispatching strategy (RF-FD) for cluster rendering system is proposed. This approach takes into account the stage of rendering job scheduling and task dispatching to obtain the maximization of system performance and balancing load distribution of multi frames. And this work consists of three parts:

(1) A reservation-based FirstFit (RF) job scheduling strategy, called the first level strategy, is presented to achieve optimal scheduling for various rendering jobs with different priorities. The RF strategy ensures that the high-priority rendering jobs will not be delayed, *i.e*., with respect to fairness. And the low-priority jobs can be performed in advance in order to avoid starvation and reduce resource fragmentation in traditional scheduling methods.
(2) A feedback-based task distribution (FD) strategy with consideration of memory limitations and load balancing among cores, called the second level strategy, is presented to make better use of frame-to-frame coherence and the relationship between rendering time and the number of thread to improve rendering performance.
(3) Two scene models with different complexity are used, and the effects of rendering time reduction for a rendering job and load balancing among cores is verified by the comparison with the fixed threads and naïve load balancing method. The effectiveness of the proposed strategy compared with FCFS and FirstFit combining with the fixed-threads and naïve load balancing method in two different situations with different evaluation metrics are also validated. The proposed method outperforms other two methods and can increase system resources utilization while maintaining satisfactory responsive time.

The rest of the work is organized as follows: Section 2 provides a summary of related works. Then the impacts of the number of rendering threads on the completion time and CPU utilization of the multi-frames rendering job are analyzed in Section 3. In Section 4, the proposed two-level hierarchy job scheduling and task dispatching strategy (RF-FD) is introduced in detail. In Section 5 the universal evaluation metrics are presented, and the proposed strategy is compared with traditional scheduling approaches by the experimental results in Section 6. Section 7 makes a conclusion and suggests a few directions for further research.

2. RELATED WORKS

2.1 Job Scheduling

The problem of scheduling jobs in cluster system has instigated researchers to propose different approaches. The main purpose of job scheduling is to achieve high performance computing and high throughput computing. The former aims at increasing execution efficiency and minimizing the execution time of jobs, whereas the latter aims at decreasing processor idle time and scheduling a set of various jobs to improve the processing capacity of the systems. Job scheduling strategy is a major component for HPC

environment, the general concept of job scheduling strategy is to select the resources for job submitted and to find the perfect job for available computation resources. These jobs may be shuffled in any order or submitting time for giving the priority for jobs.

Scheduling rendering jobs on a given set of computing resources is a key issue for cluster rendering systems. And various scheduling algorithms have been proposed [17, 18] for the cluster environment and minimizing mean job completion time is an important objective in both systems and theory literature. To improve the utility of the system, Zhou *et al*. [19] presented a novel and efficient two-phase scheduling method in DreamWorks animation's production environment. Beaudoin *et al*. [20] addressed the problem of distributing rendering computations in the graphics cluster through a novel in-frame two-phase load balancing technique with the purpose of making use of the resources effectively.

Besides, a research direction focuses on providing fair scheduling [21, 22] among users in clusters. Backfilling algorithm leverages fairness and performance in a simple and efficient manner. But backfilling need the prediction by users or system, sometimes the prediction is not precisely, therefore backfilling would suffer from unfairness [15] and increase the blocked time of the system. Other job scheduling techniques have been designed to reduce the turnaround time and increase utilization in a "fair" environment. However, fairness and performance in the cluster rendering system are rarely taken into consideration simultaneously. Comparing with the previous works which usually focus on one aspect, performance or fairness, the proposed method considers comprehensively the fairness and performance. We set the priority for each job to guarantee the fairness, the jobs with high priority will be prior to be scheduled. The jobs with low priority will be scheduled in advance to avoid the starvation and reduce resource fragmentation when the current recourses cannot satisfy the demand of jobs with high priority. In this way, the performance will be ensured. And once the resources satisfy the demand of jobs with high priority, the jobs with low priority will be paused and release the resources in consideration of fairness.

2.2 Task Dispatching

Based on the master-slave architecture, Banino *et al*. [23] presented a method to forecast the amount of tasks each processor needs to dispatch in a given period of time. Concentrating on message passing disregarding computation time, the pipelining broadcast method [24] is presented. Intuitionally in their implementation, fast processor receives more tasks in the proportional distribution policy. Tasks are also prior allocated to faster slave processors and higher system throughput could be obtained.

For the past few years, many existing tasks dispatching methods have been proposed for balancing the rendering workload among a number of rendering nodes and achieving high throughput for rendering cluster system. Examples of such methods are the partitioning scheme based on rendering time of the previous frame [25], the dynamic feedback based on results of the former frame task assignment method to handle the variation of rendering frames [26], the dynamic task distribution algorithm based on predicting rendering load distribution among the screen space [27], the fair workload division method based on binary swap and direct send compositing schemes [28], task assignment method based on the critical path lengths of jobs [29], the hierarchy tasks sub-

division method which takes advantage of temporal parallelism among frame sequences and spatial parallelism inside the single frame to achieve flexible fine grain tasks distribution [30], task assignment strategy through intelligently bidding for a pending work from each rendering agent [31], and so on.

However, these task-centered methods would not make significant contributions to system performance, which means these methods may bring extra overhead, and are not suitable for the case that the complexity of each part in the scene is not known in advance. To reduce the complexity of the assignment method, we set a fixed task granularity and explore to improve the system performance by optimizing the resource distribution.

Although there are very few resource-centered tasks dispatching methods designed for rendering cluster system, the importance of resource allocation [32-34] has been testified by many research works in other areas. For example, Maggio *et al*. [33] explores to optimize the resource allocation to consume less power while meeting performance requirement and Warneke and Kao [34] dynamically allocates different computing resources from a cloud to maximize the system throughput and resource utilization. Most traditional resource management techniques are based on a priori characterizations of the expected workload, statically allocate the constant resources and are independent of the current resource needs.

Maggio *et al*. [35] have introduced a control-theoretical CPU allocation method with feedback control to distribute computing units to running applications in a multi-application multicore environment. To rescale the number of cores assigned to each application when the demand is not feasible, multiple solutions like fair distribution, weighted distribution, priority distribution, core sharing and centroid anti-windup are sketched. Unfortunately, there are three main issues related to this method and therefore it cannot be applied to rendering cluster system directly.

(1) Control-theoretical CPU allocation method ignores the memory limitation when multiple applications are running simultaneously. As such, for multiple highly memory-intensive rendering applications, the overall performance will be poor due to the bottleneck on the memory side.
(2) The number of applications in the control-theoretical CPU allocation method is independent of the number of cores assigned to each application. Unfortunately, a rendering job can be divided into several sub-jobs by the number of frames and computing units, thus the number of instances of the application cannot be known in advance.
(3) The anti-windup mechanism rescales the number of assigned cores to each application according to the application's performance levels and goals. However, each rendering sub-job cannot express its goals because these sub-jobs work together to achieve a desired performance level for the entire rendering job in the form of a desired frame rate.

Comparing with the previous works on task dispatching and the above disadvantages on CPU allocation method, the proposed method uses the scene geometry in the view frustum for better segmentation of shots, and then chooses the key frames to pre-render. And the appropriate number of threads will be determined under the limiting condition of total memory capacity and the number of cores in CPU according to the feedback of memory usage information. In this way, the multi-core resource can be utilized sufficiently to speed

up the rendering process of serial frames in a job. In the meantime, the proposed task dispatching method under the consideration of frame-to-frame coherence reduces rendering time difference of frames and balances the workload among rendering units, which brings the better improvement for system throughput and performance.

3. ANALYZING OF RELATIONSHIP BETWEEN RENDERING TIME AND THREAD

The inter-frame scheduling is opted, which means a frame is the smallest unit of assigning and the rendering request of one frame is defined as a task. This section will analyze the relationship between rendering time and the number of thread, and describe a new tasks assignment problem in order to make better use of this relationship to improve rendering performance.

Considering the multi-core friendly render, such as Arnold, we make full use of the advantage of multi-core and use the whole cores in CPU to render multiple frames in a job. The multi threads will significantly speedup the render process and reduce the rendering time. When receiving a request to render multiple frames, current renderers like Maya and Blender, which are less multi-core friendly, will use multiple threads to render each frame in proper order. However, these renderers could not give *N*x speedup where *N* is the number of threads because much effort is wasted in communicating and synchronization. For example, when there're lots of rather complex objects which depends on the same rig and one is editing the rig, objects will be updated in separate threads as well. To support this claim, a simple scene including 123 frames is chosen for test, of which a 32-frames sub-job is rendered by Blender with different number of threads from 1 to 8 on an 8-cores rendering node. Table 1 shows the relationship between rendering time and the number of threads.

As shown in Table 1, instead of 8x faster with 8 threads, the speed-up is only 3.07x faster which means poor efficiency of multiple cores and more money users will spend. Meanwhile, we find that the memory during rendering is almost the same with different thread, in this work, we suppose that the memory of a frame is the same with different number of thread.

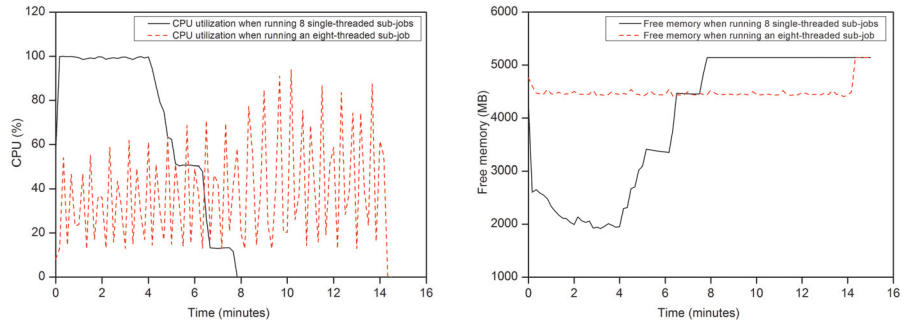
Table 1. Speedup for different thread numbers.

Threads	Rendering time	Speedup
1	43 minutes 28 seconds	1x
2	26 minutes 19 seconds	1.65x
3	21 minutes 13 seconds	2.05x
4	18 minutes 11 seconds	2.39x
5	16 minutes 10 seconds	2.69x
6	15 minutes 9 seconds	2.87x
7	14 minutes 44 seconds	2.95x
8	14 minutes 10 seconds	3.07x

In order to fully take advantage of multiple cores and improve system throughput, we divide this 32-frames rendering sub-job into 8 sub-jobs of 4 frames per sub-job and

start 8 single-threaded blender instances to render them on the same rendering node concurrently. The latest sub-job completion time is taken as the final and 7 minutes are spent on rendering the whole job which means 6.14x faster with 8 cores. Actually, if more effort is put into load balancing, the effect would be even better. Fig. 1 shows the CPU utilization and free memory space during the execution of 8 single-threaded sub-jobs and single eight-threaded sub-job. From the plots, it can be seen that more CPUs are taken and less free memory is left when running 8 single-threaded sub-jobs. Of course, after 8 minutes, running an eight-threaded sub-job will take more resources because 8 single-threaded sub-jobs have already finished.

However, there are not all rendering jobs which can improve speedup through this method because of memory limitations. When rendering complex scenes, the memory requirement is extremely high. Although increasing swap file space used by the OS would make renderer continue working, the time used for memory swapping is really expensive, which will make rendering multiple single-threaded sub-jobs not worth the candle.



(a) CPU utilization (b) Free memory

Fig. 1. CPU utilization and free memory during the execution of 8 single-threaded sub-jobs and an eight-threaded sub-job.

Therefore, in order to deal with masses of rendering jobs, maximize the resource utilization and reduce the waiting time of user's jobs, the proposed strategy tends to avoid the problem of job starvation and reduce resource fragmentations. Meanwhile, it takes memory limitation and load balancing among cores into consideration and simplifies the control process which benefits from frame-to-frame coherence to maximize system throughput. In next section, the proposed strategy in detail will be introduced.

4. THE TWO-LEVEL HIERARCHY JOB SCHEDULING AND TASK DISPATCHING STRATEGY

This section presents the two-level hierarchy job scheduling and task dispatching strategy (RF-FD) which is composed by two main components: the reservation-based FirstFit (RF) job scheduling strategy and the feedback-based task distribution (FD) strategy. When a running rendering job is finished or a new rendering job is submitted, RF-FD strategy will try to schedule the coming rendering jobs according to their priorities. The RF strategy avoids starvation, reduces resource fragmentations and also guar-

antees the fairness, especially when the idle resources are not sufficient for the highest priority job in the queue. And the FD strategy chooses an appropriate number of threads for renderer and assigns tasks to processors as evenly as possible.

4.1 The Reservation-Based Firstfit (RF) Strategy

On the basis of the above analysis, to better utilize the capabilities of rendering systems and schedule the rendering jobs, a reservation based FirstFit (RF) job scheduling strategy is proposed to achieve optimal scheduling of rendering jobs. The rendering resources and a set of jobs along with a set of frames are given. Each rendering node is homogeneous and each CPU has a limited memory. And each job can be scheduled to rendering nodes if the job's constraints (*i.e.*, number of CPUs) are fulfilled, *i.e.*, the start of a rendering job needs to meet the demand of enough CPUs due to the deadline and fee paid by users. To describe the proposed strategy, there are some definitions as follows.

Definition 1: Fill-in job, if the job is executed ahead of the jobs which have higher priority in the queue awaiting scheduling, then this job is called a fill-in job.

Definition 2: Reservation CPU, the CPU allocated to fill-in job.

According to the characteristic of rendering application, *i.e.* the rendering jobs usually include a quantity of independent frames, in the RF strategy the priority and preemption is used to guarantee that any jobs could not delay the jobs with the higher priority than them, *i.e.*, the fairness is insured. The basic idea of this RF strategy is as follows. In the procedure of scheduling, the jobs will be sorted with the order from high to low by priority in the awaiting scheduling queue. If the idle resources can meet the demand of job waiting for scheduling, the resources will be distributed to this job. Otherwise, when the number of idle CPUs plus reservation CPUs can meet the demand, the appropriate fill-in jobs will be selected and paused. Then the state of corresponding reservation CPUs becomes to free and the job awaiting scheduling is to be executed. If the idle CPUs and reservation CPUs cannot meet the demand, the next job awaiting scheduling will be selected for scheduling. Moreover, when the current job obtains idle resources, the FD strategy is employed to allocate the frames with the purpose of choosing the proper number of threads and balancing the workload among cores. In this way, the strategy ensures that the jobs with high priority have not been delayed and the jobs with low priority will utilize the resource gap after the high-priority jobs reserve rendering resources.

The procedure of selecting jobs paused is as follows. 1) sort the fill-in jobs (in interpolationJoblist) executed on the reservation CPUs with the order from low to high in priority, traverse the fill-in jobs and calculate the total number of CPUs used by these jobs. If the total number of CPUs is larger than or equal to the number of CPUs required for a rendering job, stop traversing and select the corresponding jobs. 2) compare the priority among jobs in 1) and the current job, if and only if all the priorities of these fill-in jobs are less than the current job, these fill-in jobs can be selected as the jobs paused. And the procedure of allocating resources to jobs will be presented in the next section.

4.2 The Feedback-Based Task Distribution (FD) Strategy

The FD strategy is composed by two main components: threads determiner and task dispatcher. The threads determiner chooses an appropriate number of threads for renderer. The task dispatcher assigns tasks to processors as evenly as possible. Considering the multi-core friendly render, the thread determiner will choose the fixed maximum number of threads to render a frame and then the methods in Section 4.2.2 will be used. Considering the less multi-core friendly render, the methods in Sections 4.2.1 and 4.2.2 will be used.

4.2.1 The threads determiner

The purpose of threads determiner is to choose an appropriate number of threads to start the renderer by which resources can be fully utilized and the total rendering time of the job can be reduced. Here, we distinguish the various renders with different methods. If users choose the multi-core friendly render, such as Arnold, the threads determiner will directly choose the fixed maximum number of threads to render a frame. If users choose the less multi-core friendly render, such as Maya and Blender, the thread determiner will take the following method to choose an appropriate number of threads.

For a given rendering CPU with X cores and Y (g: giga bytes) RAM memory, the minimum number of threads k is to be searched under the following constraint, U is defined as the memory usage of each task.

$$\frac{X}{Min(k)} \times U \le Y \tag{1}$$

The determination of the number of threads depends on the prediction of each task's resource usage. One method is to make predictions by using the parameters in the 3D data file that affect the resource usage. However, it is difficult to make accurate estimation by simply using parameters like the image size, the number and the type of light sources, the material properties of the objects and the geometric complexity.

To make accurate estimation, the threads decision method takes advantage of frame-to-frame coherence and tries to predict the resource usage of each frame's rendering based on some results of key frames' rendering. It does not need many pre-processing work, therefore significantly saving system overhead. Besides, it is simple to implement and can achieve better results.

Firstly, the key frames will be selected to determine the number of threads for the M frames in rendering job. We assume that R is represented the number of rendering CPUs specified by users. Generally, for the CG films, every frame might have different length, the uniform selection method could not fully exploit the temporal coherence. Therefore, we use scene geometry in the view frustum for better segmentation of frames and then choose key frames. Considering the rendering parameters feature which are typically defined in the scene model, we take the triangle count as the main factor for choosing the key frames. There are other parameters such as resolution, polygons, and the reason of choosing triangle count is that the resolution for each frame is actually identical and the

polygons are usually composed by triangles. Therefore, the triangle count could be the reasonable estimate for geometric complexity when choosing the key frames. In general, the triangle count for some continuous frames is same due to that these frames exist the effect such as translation, rotation, and we assume these frames with the same the triangle count as the same geometric complexity. The process for the key frames selection in detail is as follows. (1) Traverse the M frames of a rendering job in a ListA and measure the triangle count of each frame; (2) When the triangle count of continuous frames is identical, the ListA just reserves one frame; (3) If the frames number of ListA (N_{LA}) is less than R, we use the following steps to supplement vacant frames. First, N_{LA} is subtracted from R to obtain the number of vacant frames (N_{vf}), *i.e.*, $N_{vf}=R-N_{LA}$. Second, the $M-N_{LA}$ frames in ListB, which are the complementary set of ListA in M frame set, are sorted by the number of triangles from high to low. Third, the N_{vf} frames are selected from ListB to ListA and the R frames in ListA are chosen as key frames. If N_{LA} is greater than R, the frames in ListA are divided to R groups according to the triangle count from high to low, and each group chooses one frame with maximal triangle count as the key frame. If N_{LA} is equal to R, the R frames in ListA will be chosen as key frames. In this way, the R key frames are chosen by the scene geometry based method.

Then, each key frame will be rendered by an X-threaded renderer on each rendering CPU. After the rendering, collect the average RAM memory usage of each CPU during the execution and find the average $U(g)$. If $U(g) \le KY/X$, $K \in \{ x \mid X\%x = 0\}$, then the appropriate number of threads should be K, *i.e.*, when the memory utilization is maximum, we choose the minimum value of K. Here, we consider a special situation, we assume that when $U(g)$ equals Y, the value of K just equals X, and other situations ($U(g)$ is less than Y), there always exists the appropriate value of K.

By this way, choosing the appropriate number of threads based on the memory usage, *i.e.*, the memory usage of each rendering instance will not exceed the total memory capacity, can avoid the frequent page replacement with swap partition and reduce the I/O overhead with hard disk. When the factor of memory usage has not been considered and the current memory usage exceeds the total memory capacity, the overhead used for memory swapping is generally expensive.

4.2.2 The task dispatcher

After determining the number of threads, X cores in each rendering CPU can be divided into X/K rendering units and each unit has K cores. Task dispatcher will divide the remaining $M–R$ frames of a rendering job into RX/K sub-jobs. Since the slowest sub-job represents the bottleneck of the system, task dispatcher should put more effort into load balancing to achieve good performance.

Generally, load balancing schemes can be classified in two major categories: static and dynamic load balancing. Finally the static load balancing strategy is chosed based on the following reasons:

(1) The frame-to-frame coherence can help dispatcher assign tasks as evenly as possible before runtime.
(2) The rendering cluster system is homogeneous and it is only used for rendering which means loads will not vary significantly during runtime.

(3) The overhead of starting new rendering processes and loading scene files again makes dynamic load balancing worthless.

Formally, after threads determiner finishes rendering R key frames and determines the number of threads K, task dispatcher has to assign the remaining $(M–R)$ frames, indexed $f \in 1, \ldots, (M–R)$. Also, R CPUs have been divided into RX/K rendering units, indexed $p \in 1, \ldots, RX/K$. Because of frame-to-frame coherence, there is little difference between two consecutive frames in rendering time which means $T(f) = T(f+1)+\Delta_f$. $T(f)$ is the time required to finish rendering the frame which number is f, Δ_f represents the differences of rendering time between the f^{th} and $(f+1)^{\text{th}}$ frame for a sequence of animation, Δ_f is a low value and varies from frames to frames.

Generally, the frames in $\{f|(p–1)(M–R)/(RX/K)<f\le p(M–R)/(RX/K)\}$ will be assigned to the unit p. However, there sometimes exists a serious load imbalance problem. For example, unit 1 will render the frames from 1 to $\lfloor (M–R)/(RX/K) \rfloor$ and unit RX/K will render the frames from $\lceil (RX/K–1)(M–R)/(RX/K) \rceil$ to $M–R$. Because $T(f) = T(f+1)+\Delta_f$, therefore

$$t = \left\lceil (RX/K-1)\frac{M-R}{RX/K} \right\rceil, \tag{2}$$

$$\lambda = \left\lfloor \frac{M-R}{RX/K} \right\rfloor, \tag{3}$$

$$\sum_{f=t}^{M-R} T(f) = \sum_{f=1}^{\lambda} T(f) - \sum_{j=1}^{\lambda} \sum_{f=j}^{t+j-1} \Delta_f + \gamma T(M-R), \tag{4}$$

$$\gamma = \begin{cases} 0 & \text{if } (M-R) \bmod (RX/K) = 0 \\ 1 & \text{if } (M-R) \bmod (RX/K) \neq 0 \end{cases}. \tag{5}$$

Along with the increase of the frames, the number of frames allocated to each unit will increase. Due to the accumulation of differences among frames, the difference between unit 1 and unit RX/K in completion time is large which implies that a significant load imbalance will sometimes occur.

In order to reduce the possibility and extent of load imbalance, a new load balancing method is proposed. Unlike the naïve method, the frames in $\{f \mid f = (RX/K)z+p, z \in N, 1 \le f \le (M–R)\}$ will be assigned to the unit p which means each rendering unit will not render consecutive frames. Because the difference between unit 1 and unit RX/K is the largest, we also make a comparison of them which is more representative. Unit 1 will render the frames in $\{1, RX/K+1, \ldots, \lceil (M–R)/(RX/K)–1 \rceil RX/K+1\}$, while unit RX/K will render the frames in $\{RX/K, 2RX/K, \ldots, \lfloor (M–R)/(RX/K) \rfloor RX/K\}$. Then the following relation is shown:

$$u = \left\lfloor \frac{M-R}{RX/K} - 1 \right\rfloor, \tag{6}$$

$$v = \left\lceil \frac{M-R}{RX/K} - 1 \right\rceil, \tag{7}$$

$$\sum_{z=0}^{u} T\left(\frac{RX}{K} z + \frac{RX}{K}\right) = \sum_{z=0}^{v} T\left(\frac{RX}{K} z + 1\right) - \sum_{j=0}^{u} \sum_{f=(RX/K)j+1}^{(RX/K)(j+1)-1} \Delta_f - \gamma T\left((RX/K)v + 1\right). \quad (8)$$

In Eq. (7), γ is the same as Eq. (4). According to the above formalization, the new load balancing method with the way of interleaved distribution reduces the overall rendering time difference of frames among all the rendering units.

5. EVALUATION METRICS

Here some metrics are defined, which are also employed in [36], to quantify the efficiency of the proposed rendering job scheduling method.

Definition 3: Define P_γ to be the real-time resource utilization rate and P_{γ_a} to be the resource utilization rate over a period of time. The real-time resource utilization rate can be expressed as $P_\gamma = N_b/N$, where N_b means the number of the working CPUs and N means all the CPUs in system. And then, from time T_1 to time T_2, P_{γ_a} can be expressed as:

$$P_{r\_a} = \frac{1}{T_2 - T_1} \int_{T_1}^{T_2} P_r(t)dt. \quad (9)$$

This period of time is divided into m part and the time of each part is Δt, *i.e.* $T_2 - T_1 = m \cdot \Delta t$. When $m \to \infty$, then $\Delta t \to \infty$. And Eq. (9) can be expressed as:

$$P_{r\_a} = \frac{1}{m} \lim_{\Delta t \to 0} \sum_{i=0}^{m-1} P_r\left(i \cdot \Delta t\right). \quad (10)$$

From the above, the average value of real-time resource utilization rate is used to represent approximately the resource utilization rate over a period of time.

Definition 4: Define $T_{Init}(i)$ to be the job initial time of the rendering job i, *i.e.*, $T_{Init}(i)$ is the time when J_i is being submitted by one user and queued in the rendering system, $T_{Start}(i, k)$ to be the job start time and $T_{Finish}(i, k)$ to be the job finish time of job i running on the k rendering CPUs. The job execution time $T_{Exec}(i, k)$ of the job i allocated to k rendering CPUs is defined as $T_{Exec}(i, k) = T_{Finish}(i, k) - T_{Start}(i, k)$ and then the total finish time of all n rendering jobs can be expressed as $T_{Finish}(n) = max(T_{Finish}(i, k)) - min(T_{Init}(i, k))$.

Definition 5: Define $T_{Res}(i, k)$ to be the response time of job i with k rendering CPUs, $\overline{T}_{Res}(n)$ to be the average response time of n jobs, $T_{Turn}(i, k)$ as the turnaround time job i with k rendering CPUs and $\overline{T}_{Turn}(n)$ as the average turnaround time of n jobs, which can be expressed as follows:

$$T_{Res}(i, k) = T_{Start}(i, k) - T_{Init}(i) \quad (11)$$

$$\overline{T}_{Res}(n) = \frac{1}{n}\sum_{i=1}^{n} T_{Res}(i,k) \tag{12}$$

$$T_{Turn}(i,k) = T_{Finish}(i,k) - T_{Init}(i) \tag{13}$$

$$\overline{T}_{Turn}(n) = \frac{1}{n}\sum_{i=1}^{n} T_{Turn}(i,k) \tag{14}$$

6. RESULTS AND DISCUSSIONS

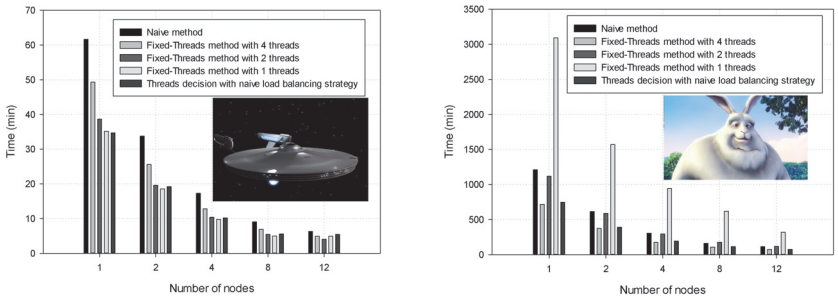
To evaluate the effect and universality of the proposed strategy, two types of test set, *i.e*. the single job and the batch jobs, are employed. The single job test set is to show the result of rendering workload balancing among CPUs when using the FD strategy in a single rendering job, and the batch jobs test set is to demonstrate the effect of performance improvement for the rendering system when adopting the proposed RF-FD strategy. The tests are conducted on cluster composed of 15 rendering nodes, each one contains two quad-core 2.4GHZ Intel Xeon processors with 8 GB of RAM memory. The 15 nodes are connected by a switched Gigabit Ethernet network, running on Linux operating system. Blender 2.60, a free and open source 3D animation suite, is used to do rendering.

6.1 The Single Job

The two 3D scene files with different complexity are used to test the proposed method. Compared with the exploding enterprise model, the rabbit model form the movie "Big Buck Bunny" is more complex and it will take up more memory.

6.1.1 Test of threads decision method

In the first study, we focus on the effectiveness and commonality of the threads determiner for different scene models and different number of rendering nodes. Based on the information feedback, the proposed method automatically chooses one thread for rendering each frame of the exploding enterprise model and four threads for the rabbit model. Fig. 2 compares the completion time of using the naïve tasks assignment method, fixed-threads method and the threads decision without the new load balancing method for rendering the exploding enterprise model and the rabbit model with nodes 1, 2, 4, 8 and 12 respectively.



(a) Exploding enterprise model. (b) Rabbit model.

Fig. 2. Comparison of completion time for two models.

It can be seen that the threads decision method always achieves better results no matter how many nodes used to finish the job. Also, the threads determiner is meaningful to both simple and moderately complex scene models. In contrast, fixed-threads method cannot have a good performance in both cases. Although fixed-threads method with fewer threads has good results for simple scenes, it will have nightmarish results when rendering complex scenes, as shown in Fig. 2 (b). Multiple few-threads processes running simultaneously will take up so much memory that much time has to be spent on memory swapping. If the fixed-threads method with more threads is chosen, system resources cannot be fully used when rendering simple scenes so that the results are worse than that of threads decision strategy, as shown in Fig. 2 (a).

Obviously, when rendering extremely complex scenes, there will be little difference between naïve method and threads decision method because our system will also choose 8 threads to start the renderer. Therefore, intelligent threads determiner makes cluster system resources be fully used while improving the performance.

6.1.2 Test of the new load balancing method

In the following study, we evaluate the performance of the proposed new load balancing method by three different targets as follows: (1) Variance: The variance of load is computed from the rendering time of each sub-job on its processor. A large variance indicates a load imbalance among the processors in the system; (2) Max-Min: It represents the distance between the slowest sub-job and the fastest sub-job. It also implies the imbalance of workloads allocated to processors; (3) Max: The slowest sub-job represents the bottleneck of the system. Max is the execution time of the slowest sub-job. The goal of load balancing is also to minimize the maximum load, thereby reducing the overall execution time of the entire job.

Table 2. Dispatching results.

Sub-jobs	Naïve load balancing method			Proposed load balancing method		
	Var	Max-Min	Max	Var	Max-Min	Max
8	29.6	17.3	34.2	0.2	1.4	29.9
16	8.8	11.2	18.5	0.6	2.2	15.5
24	3.6	6.9	12.1	0.1	1.3	10.3
32	2.9	6.5	9.6	0.5	2.1	8.0

We perform the comparison between naïve method and the proposed method to illustrate the efficacy of the proposed method on balancing the loads and improving the performance. We choose the exploding enterprise model to do the tests with nodes 1, 2, 3 and 4 respectively. As the threads decision method has chosen 1 thread for each renderer in the first stage, the 123-frames rendering job in these tests will be split into 8, 16, 24 and 32 sub-jobs respectively. And Table 2 presents the results.

In Table 2, it confirms that the proposed load balancing method has a significant effect on the balance and the execution time. Moreover, the fewer the sub-jobs, the more evident the improvement becomes. Fig. 3 shows the performance difference between the two methods by presenting CPU utilization information during execution for 8 sub-jobs.

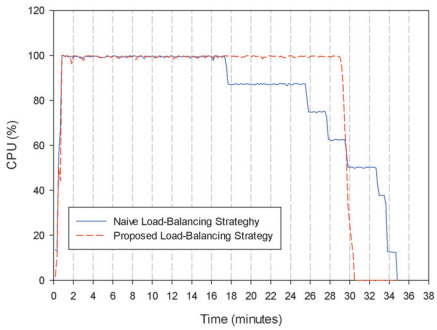


Fig. 3. Comparison of CPU utilization during execution for 8 sub-jobs.

One can see that after 17 minutes, the CPU utilization of naïve load balancing method begins to decrease, which means the earliest sub-job has been finished. However, another 18 minutes are needed before the whole job finishes which implies a significant imbalance among these sub-jobs. In all, the proposed load balancing method keeps the CPU utilization high as long as possible, which results in better performance than the naïve method.

6.2 The Batch Jobs

According to the proposed strategy described in section 4, the jobs with low priority may be executed ahead of the jobs with high priority, which seems to violate the principle of fairness, but the jobs with low priority doesn't delay the jobs with high priority. With the assistance of preemption mechanism the proposed method guarantees that no job is delayed by any jobs with lower priority which is defined in [15], *i.e.*, the proposed method follows the principle of fairness. In order to further assess the performance of the proposed scheduling method, we use two groups of different workloads rendering jobs from the movie "Big Buck Bunny" and include 50 and 100 jobs with various resources requirements. And the percentage of priority of each group is 20%, 40% and 40% which correspond to the jobs with high priority, middle priority and low priority, respectively.

For the comparison target, the following two widely used job scheduling strategies are selected. The first one is First Come First Serve (FCFS) which schedules jobs according to the arrival time in the queue. The second one is FirstFit described in the above section. At the task dispatching stage, the two methods use the fixed 8 threads and naïve load balancing method. Therefore, we call the above strategies as FCFS-fn and Firstfit-fn for short to compare with the proposed RF-FD strategy.

According to Eq. (10), the resource utilization rate of the two groups under various scheduling strategies is shown in Fig. 4. Compared with the other two methods, the resource utilization rate of group one (50 rendering jobs) has been improved about 10.95% and 2.67%. Meanwhile, the rate of group two (100 rendering jobs) has been improved about 13.79% and 2.42%. The FCFS-fn method schedules these jobs according to the arrival time, which may generate more idle resources due to some the blocked jobs that need more resources. As the Firstfit-fn method shown, the jobs with less resources demand were scheduled preferentially which may generate less idle resources. And RF-FD strategy takes a full consideration of the whole jobs with various resources requests and

the jobs required less resources would not delay the executing of the jobs required more resources at the head of the queue. And then RF-FD strategy causes higher resource utilization rate.

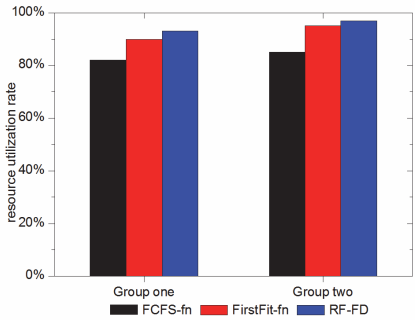
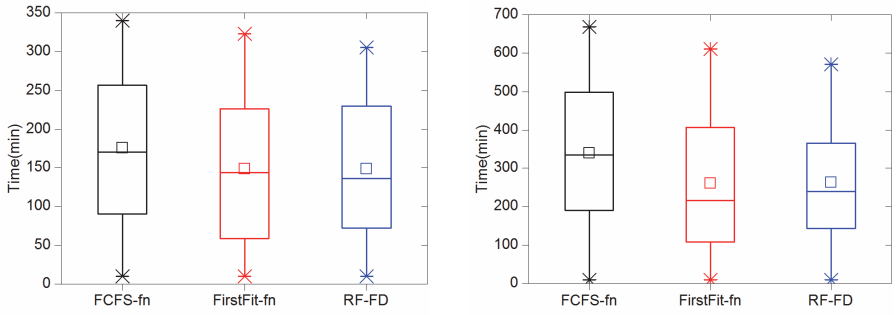


Fig. 4. The resource utilization rate comparison of two group of jobs.

In Fig. 5, the turnaround time distribution of the two groups is shown, which is calculated by Eqs. (13) and (14).



(a) Group one: 50 jobs (b) Group two: 100 jobs

Fig. 5. The boxplot of the two groups. It contains the minimum, maximum, mean and median value of the turnaround time.

Considering the mean turnaround time, Firstfit-fn and RF-FD strategy has been lower than FCFS-fn method. Using RF-FD strategy, the execution of the jobs with high recourses demand has not been delayed and these jobs occupy the rendering resources which can meet some jobs with less recourses demand. Therefore, the mean turnaround time of RF-FD strategy has been a little higher than Firstfit-fn method. The total finish time (*i.e.* the maximum value of the turnaround time) of Firstfit-fn and RF-FD strategy has been greatly shortened. This is because the FCFS-fn method may lead to the waste of idle resources as described in above section. And using RF-FD strategy, the execution of jobs with less recourses demand may not delay the jobs with high recourses demand and high priority, which contributes to the high resources utilization rate and the less total finish time of jobs.

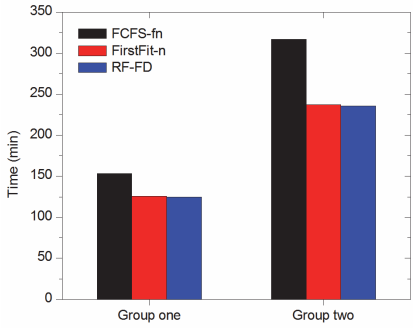


Fig. 6. The average response time comparison of two groups jobs.

Fig. 6 shows the average response time of two group jobs calculated under Eq. (11) and Eq. (12). Comparing with the average response time of FCFS-fn method, the time of Firstfit-fn and RF-FD strategy has been obviously reduced. And the average response time of Firstfit-fn and RF-FD strategy is approximate and indeterminate. RF-FD strategy considers the jobs with high recourses demand and these jobs occupy the resources which can meet some jobs with less recourses demand. Therefore, the average response time of RF-FD strategy is a little longer than that of the Firstfit-fn method. Although the jobs with high recourses demand are executed firstly, the pause of these jobs makes that the jobs with less recourses demand are not delayed to be executed. Therefore, the average response time of RF-FD strategy is shorter than that of the Firstfit-fn method.

7. CONCLUSION AND FUTURE WORKS

In this work, a novel two-level hierarchy job scheduling and task dispatching strategy (RF-FD) is designed for cluster rendering system when it manages the arrived rendering jobs submitted by various users with multiple resources demand. The proposed strategy integrates reservation-based FirstFit (RF) job scheduling strategy and feedback-based task distribution (FD) strategy. The methodology takes into consideration the stage of rendering job scheduling and frames allocation to obtain the maximization of system performance and balance load distribution of multi frames. Comparing with the two traditional scheduling method, *i.e.*, FCFS and FirstFit combining with the fixed-threads and naïve load balancing method, the test results show that the proposed RF-FD strategy can schedule rendering jobs more effectively. Meanwhile, RF-FD strategy can eliminate the starvation, reduce resource fragmentations and keep the higher throughput and resource utilization rate for rendering cluster. Moreover, this strategy guarantees fairness of scheduling jobs and leads to better overall performance of the system. In addition, it is not only that the proposed method can be used in rendering application, but also that the proposed design principle and scheduling scheme can be applicable to a wider class of applications with the characteristic of paused jobs, which gives us a highly scalable solution. Considering the limitations of the proposed method, it may not have the better effect on the scene with dramatic changes due to the unknown of the execution time of jobs.

Future research is going to focus on the scheduling issue of rendering jobs on het-

erogeneous resources with the unequal division and the dynamic frame allocation without reloading anything. Under the situation of knowing the execution and response time of a job in advance, the scheduling strategy will be focused. And the rendering time prediction method considering the parameter characteristic with the purpose of improving the prediction accuracy will be focused to schedule the jobs more precisely and effectively. Moreover, combining the mode of cloud rendering, we are going to focus on the job scheduling problem to meet the requirements of users' deadline when the local render farms lack of enough rendering resources.

REFERENCES

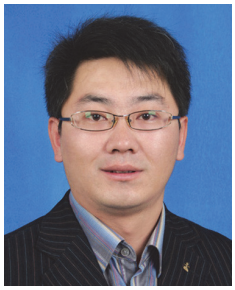
1. A. Iosup, S. Ostermann, M. N. Yigitbasi, R. Prodan, T. Fahringer, and D. H. Epema, "Performance analysis of cloud computing services for many-tasks scientific computing," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 22, 2011, pp. 931-945.
2. M. R. Baharon, Q. Shi, D. Llewellyn-Jones, and M. Merabti, "Secure rendering process in cloud computing," in *Proceedings of the 11th Annual International Conference on Privacy*, *Security and Trust*, 2013, pp. 82-87.
3. A. Chong, A. Sourin, and K. Levinski, "Grid-based computer animation rendering," in *Proceedings of the 4th International Conference on Computer Graphics and Interactive Techniques in Australasia and Southeast Asia*, 2006, pp. 39-47.
4. L. Xu, Z. Wang, and W. Chen, "An integrated dynamic resource scheduling framework in on-demand clouds," *Journal of Information Science and Engineering*, 2014, Vol. 30, pp. 1537-1552.
5. J. Kołodziej, S. U. Khan, L. Wang, A. Byrski, N. MinAllah, and S. A. Madani, "Hierarchical genetic-based grid scheduling with energy optimization," *Cluster Computing*, Vol. 16, 2013, pp. 591-609.
6. R. Baraglia, G. Capannini, P. Dazzi, and G. Pagano, "A multi-criteria job scheduling framework for large computing farms," *Journal of Computer and System Sciences*, Vol. 79, 2013, pp. 230-244.
7. A. A. Chandio, C.Z. Xu, N. Tziritas, K. Bilal, and S. U. Khan, "A comparative study of job scheduling strategies in large-scale parallel computational systems," in *Proceedings of the 12th IEEE International Conference on Trust*, *Security and Privacy in Computing and Communications*, 2013, pp. 949-957.
8. R. Jeyarani, R. V. Ram, and N. Nagaveni, "Design and implementation of an efficient two-level scheduler for cloud computing environment," in *Proceedings of the 10th IEEE/ACM International Conference on Cluster*, *Cloud and Grid Computing*, 2010, pp. 585-586.
9. G. Kannan and S. Selvi, "Nonpreemptive priority (NPRP) based job scheduling model for virtualized grid environment," in *Proceedings of the 3rd International Conference on Advanced Computer Theory and Engineering*, 2010, pp. 377-381.
10. R. Grandl, G. Ananthanarayanan, S. Kandula, S. Rao, and A. Akella, "Multi-resource packing for cluster schedulers," in *Proceedings of ACM Conference on SIGCOMM*, 2014, pp. 455-466.
11. J. Tan, X. Meng, and L. Zhang, "Coupling task progress for mapreduce resource-

aware scheduling," in *Proceedings of IEEE International Conference on Computer Communications* , 2013, pp. 1618-1626.
12. H. Sun, W. J. Hsu, and Y. Cao, "Competitive online adaptive scheduling for sets of parallel jobs with fairness and efficiency," *Journal of Parallel and Distributed Computing*, Vol. 74, 2014, pp. 2180-2192.
13. A. W. Mu'alem and D. G. Feitelson, "Utilization, predictability, workloads, and user runtime estimates in scheduling the ibm sp2 with backfilling," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 12, 2001, pp. 529-543.
14. V. J. Leung, G. Sabin, and P. Sadayappan, "Parallel job scheduling policies to improve fairness: a case study," in *Proceedings of the 39th International Conference on Parallel Processing Workshops*, 2010, pp. 346-353.
15. Y. Yuan, Y. Wu, W. Zheng, and K. Li, "Guarantee strict fairness and utilize prediction better in parallel job scheduling," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 25, 2014, pp. 971-981.
16. C. B. Lee, Y. Schwartzman, J. Hardy, and A. Snavely, "Are user runtime estimates inherently inaccurate?" in *Proceedings of the 11th International Workshop on Job Scheduling Strategies for Parallel Processing*, 2005, pp. 253-263.
17. T. D. Braun, H. J. Siegel, N. Beck, L. L. Bölöni, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys, B. Yao, D. Hensgen, and R. F. Freund, "A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems," *Journal of Parallel and Distributed computing*, Vol. 61, 2001, pp. 810-837.
18. D. Y. Chen, J. J. Wu, and P. Liu, "Data-bandwidth-aware job scheduling in grid and cluster environments," in *Proceedings of the 15th International Conference on Parallel and Distributed Systems* , 2009, pp. 414-421.
19. Y. Zhou, T. Kelly, J. Wiener, and E. Anderson, "An extended evaluation of two-phase scheduling methods for animation rendering," in *Proceedings of the 11th International Workshop on Job Scheduling Strategies for Parallel Processing*, 2005, pp. 123-145.
20. A. Beaudoin, D. Goswami, and S. Mudur, "Two-phase load distribution for rendering large 3d models on a graphics cluster," in *Proceedings of IEEE International Conference on Cluster Computing and Workshops*, 2009, pp. 355-364.
21. N. D. Doulamis, A. D. Doulamis, E. A. Varvarigos, and T. A. Varvarigou, "Fair scheduling algorithms in grids," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 18, 2007, pp. 1630-1648.
22. D. Klusácek and H. Rudová, "Performance and fairness for users in parallel job scheduling," in *Proceedings of the 17th International Workshop on Job Scheduling Strategies for Parallel Processing*, 2013, pp. 235-252.
23. C. Banino, O. Beaumont, L. Carter, J. Ferrante, A. Legrand, and Y. Robert, "Scheduling strategies for master-slave tasking on heterogeneous processor platforms," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 15, 2004, pp. 319-330.
24. O. Beaumont, A. Legrand, L. Marchal, and Y. Robert, "Pipelining broadcasts on heterogeneous platforms," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 16, 2005, pp. 300-313.
25. F. Abraham, W. Celes, R. Cerqueira, and J. L. Campos, "A load balancing strategy for sort-first distributed rendering," in *Proceedings of the 17th Brazilian Symposium*

on Computer Graphics and Image Processing, 2004, pp. 292-299.
26. Y. Zhang, T. Mao, and Z. Wang, "Parallel rendering for large-scale crowd based on dynamic feedback," in *Proceedings of the Workshop on Digital Media and Digital Content Management*, 2011, pp. 172-175.
27. C. Hui, L. Xiaoyong, and D. Shuling, "A dynamic load balancing algorithm for sort-first rendering clusters," in *Proceedings of the 2nd IEEE International Conference on Computer Science and Information Technology*, 2009, pp. 515-519.
28. S. Eilemann and R. Pajarola, "Direct send compositing for parallel sort-last rendering," in *Proceedings of the 7th Eurographics Conference on Parallel Graphics and Visualization*, 2007, pp. 29-36.
29. E. Anderson, D. Beyer, K. Chaudhuri, T. Kelly, N. Salazar, C. Santos, R. Swaminathan, R. Tarjan, J. Wiener, and Y. Zhou, "Value-maximizing deadline scheduling and its application to animation rendering," in *Proceedings of the 17th Annual ACM Symposium on Parallelism in Algorithms and Architectures*, 2005, pp. 299-308.
30. J. Yao, Z. Pan, and H. Zhang, "A distributed render farm system for animation production," in *Proceedings of the 8th International Conference Entertainment Computing*, 2009, pp. 264-269.
31. D. Vallejo, C. GlezMorcillo, J. Angulo, and J. Albusac, "Distributed rendering of images through intelligent task distribution," in *Proceedings of the 6th International Conference on P2P*, *Parallel*, *Grid*, *Cloud and Internet Computing*, 2011, pp. 242-247.
32. P. P. Hung, T. A. Bui, M. A. G. Morales, M. Van Nguyen, and E. N. Huh, "Optimal collaboration of thin-thick clients and resource allocation in cloud computing," *Personal and Ubiquitous Computing*, Vol. 18, 2014, pp. 563-572.
33. M. Maggio, H. Hoffmann, M. D. Santambrogio, A. Agarwal, and A. Leva, "Power optimization in embedded systems via feedback control of resource allocation," *IEEE Transactions on Control Systems Technology*, Vol. 21, 2013, pp. 239-246.
34. D. Warneke and O. Kao, "Exploiting dynamic resource allocation for efficient parallel data processing in the cloud," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 22, 2011, pp. 985-997.
35. M. Maggio, H. Hoffmann, A. Agarwal, and A. Leva, "Control-theoretical CPU allocation: Design and implementation with feedback control," in *Proceedings of the 6th International Workshop on Feedback Control Implementation and Design in Computing Systems and Networks*, 2011, pp. 1-6.
36. W. H. Hsu, C. F. Wang, K. L. Ma, H. Vu, and J. H. Chen, "A job scheduling design for visualization services using GPU cluster," in *Proceedings of IEEE International Conference on Cluster Computing*, 2012, pp. 523-533.

Qian Li (李谦) is a Ph.D. Candidate in Department of Computer Science and Technology at Xi'an Jiaotong University, China. His research interests include high performance computing and cluster rendering.

Wei-Guo Wu (伍卫国) received his M.S. and Ph.D. Degrees in Computer Science and Technology from Xi'an Jiaotong University, China. He is currently a Professor in the Department of Computer Science and Technology at Xi'an Jiaotong University, China. His research interests include high performance computer architecture, cloud computing and embedded system.



Ze-Yu Sun (孙泽宇) is a Ph.D. student in Department of Computer Science and Technology at Xi'an Jiaotong University, China. His research interests include high performance computer architecture.

Jian-Hang Huang (黄舰航) is an M.S. student in Department of Computer Science and Technology at Xi'an Jiaotong University, China. His research interests include parallel computing.