Future Generation Computer Systems 43–44 (2015) 51–60

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

# A self-adaptive scheduling algorithm for reduce start time

Zhuo Tang [a,*], Lingang Jiang [a], Junqing Zhou [a], Kenli Li [a], Keqin Li [a,b]

[a] College of Information Science and Engineering, Hunan University, Changsha 410082, China
[b] Department of Computer Science, State University of New York, New Paltz, NY 12561, USA

## HIGHLIGHTS

- This paper illustrates the reasons of the system slots waster for reduces tasks waiting around.
- The model can determine the start time of reduce tasks, dynamically according to job context.
- As an optimal scheduling algorithm, SARS can decrease the reduce completion time for jobs.

## ARTICLE INFO

*Article history:*
Received 28 December 2013
Received in revised form
1 August 2014
Accepted 15 August 2014
Available online 25 August 2014

*Keywords:*
Big data
Hadoop
MapReduce
Reduce
Self-adaptive
Task scheduling

## ABSTRACT

MapReduce is by far one of the most successful realizations of large-scale data-intensive cloud computing platforms. When to start the reduce tasks is one of the key problems to advance the MapReduce performance. The existing implementations may result in a block of reduce tasks. When the output of map tasks become large, the performance of a MapReduce scheduling algorithm will be influenced seriously. Through analysis for the current MapReduce scheduling mechanism, this paper illustrates the reasons of system slot resources waste, which results in the reduce tasks waiting around, and proposes an optimal reduce scheduling policy called SARS (Self Adaptive Reduce Scheduling) for reduce tasks' start times in the Hadoop platform. It can decide the start time point of each reduce task dynamically according to each job context, including the task completion time and the size of map output. Through estimating job completion time, reduce completion time, and system average response time, the experimental results illustrate that, when comparing with other algorithms, the reduce completion time is decreased sharply. It is also proved that the average response time is decreased by 11% to 29%, when the SARS algorithm is applied to the traditional job scheduling algorithms FIFO, FairScheduler, and CapacityScheduler.



## 1. Introduction

MapReduce is an excellent model for distributed computing, introduced by Google in 2004 [1]. It has emerged as an important and widely used programming model for distributed and parallel computing, due to its ease of use, generality, and scalability. Among its open source implementation versions, Hadoop has been widely used in industry around the whole world [2] and has been used/extended by scientists as the base of their own research work [3]. It has been applied to the production environments, such as Google, Yahoo, Amazon, Facebook, and so on. Because of the short development time, Hadoop can be improved in many aspects, such as intermediate data management and reduce tasks scheduling [4]. This paper mainly focuses on the reduce scheduling problem, which refers to the starting times of the reduce tasks.

Map and Reduce are the two sections in a MapReduce scheduling algorithm. In Hadoop, each task contains three functioning phases: copy, sort, and reduce [5]. The goal of the copy phase is to read the map tasks' outputs. The sort phase is to sort the intermediate data which are produced by map tasks and will be the input to the reduce phase. Finally, the eventual results are produced through the reduce phase, where the copy and sort phases are to preprocess the input data of reduce. In real applications, copying and sorting may consume considerable amount of time, especially in the copy phase. In the theoretical model, the reduce functions start only if all map tasks are finished [6]. However, in the Hadoop implementation, all copy actions of reduce tasks will start when the first map action is finished [7]. But in a slot duration, if there is any map task still running, the copy actions will wait around. This will lead to the waste of the reduce slot resources.

The existing MapReduce program frameworks often treat the jobs as a whole process. However, the differences between the map and reduce tasks are not considered. Since map and reduce task execution times are not related, it is not accurate to compute the

* Corresponding author. Tel.: +86 18627568501.
*E-mail address:* ztang@hnu.edu.cn (Z. Tang).

http://dx.doi.org/10.1016/j.future.2014.08.011
0167-739X/

average task execution time by taking map and reduce tasks together. The dynamic proportional scheduler [8] provides more job sharing and prioritization capability in scheduling and also results in increasing share of cluster resources and more differentiation in service levels of different jobs. Zaharia et al. proposed the delay scheduling algorithm [9] to address the conflict between data locality and fairness. Time estimation and optimization for Hadoop jobs have been explored in [10,11]. In [10], the authors focused on minimizing the total completion time of a set of MapReduce jobs. In [11], the authors estimated the progress of queries that run as MapReduce DAGs.

The need for a scheduling algorithm arises from the requirement for most modern systems to perform multitasking (execute more than one process at a time) and multiplexing (transmit multiple flows simultaneously) [12]. In traditional MapReduce scheduling, the reduce task should begin when all the map tasks are completed. In this way, the outputs of map tasks should be read and written to the reduce tasks in the copy process [13]. However, through the analysis of the slot resource usage in the reduce process, this paper illustrates that data transfer will result in slot idle and delay. On the other hand, various types of information and data processed in the large-scale dynamic grid environment may be incomplete, imprecise, fragmentary and overloading [14]. So as Hadoop platform in cloud computing, when the map tasks' outputs become large, the performance of a MapReduce scheduling algorithm will be influenced seriously [15]. Especially, when multiple tasks are running simultaneously, inappropriate scheduling of reduce tasks will lead to untimely scheduling of other jobs in the system, and these are all the stumbling blocks of the Hadoop popularization.

Through studying reduce task scheduling in the Hadoop platform, this paper proposes an optimal reduce scheduling policy called SARS (Self Adaptive Reduce Scheduling). SARS can reduce the waiting around of copy actions and advance the performance of a whole system. Through analyzing the details of a map and reduce two-phase scheduling process at runtime, the SARS algorithm can determine the start time point of each reduce task dynamically according to each job's context, such as the task completion time or the size of map output. This paper makes the following contributions to MapReduce performance enhancement:

(1) The analysis for the current MapReduce scheduling mechanism, and illustration of the reasons of system slot resources wasting, which results in the reduce tasks waiting around;
(2) The development of a model details the start times of the reduce tasks dynamically according to each job context, including the task completion time and the size of map output;
(3) An optimal reduce scheduling algorithm which decreases the reduce completion time and the system average response time in the Hadoop platform.

The rest of this paper is organized as follows. Section 2 reviews the related works. Section 3 analyzes the problem of the long waiting of reduce tasks. Section 4 proposes an optimal reduce scheduling algorithm for reduce tasks' start times in the Hadoop platform. Experiments and analysis which support our contributions are presented in Section 5. Section 6 concludes this paper and describes the future work.

## 2. Related work

There have been a number of proposals for task scheduling in distributed systems, which use various mathematical techniques to achieve the MapReduce scheduling process. At present, the researches on MapReduce scheduling algorithms focus on the optimization of the job computation time, cluster workloads, and data communication.

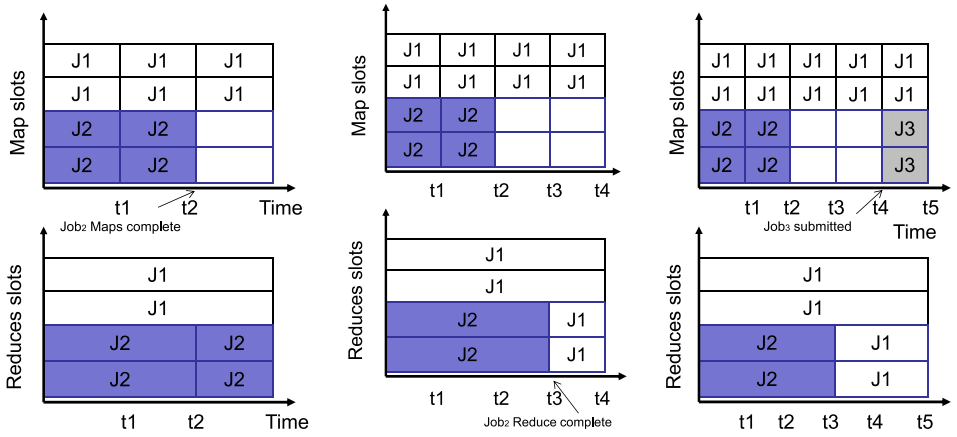
Balanced-pools efficiently utilize performance properties of MapReduce jobs in a given workload for constructing an optimized job schedule [16]. For the default method of Hadoop, which cannot schedule the tasks to the nodes with the prefetched data, a prefetching technique was proposed in [17] to hide the remote data access delay caused by the map tasks processed on the nodes without the input data. We proposed MTSD, an extensional MapReduce task scheduling algorithm for deadline constraints in the Hadoop platform [18], which allows a user to specify a job's deadline and tries to make the job to be finished before the deadline. Some of these proposals [19] presented the workload characteristic oriented scheduler, which strives for co-locating tasks of possibly different MapReduce jobs with complementing resource usage characteristics. In [20], the authors presented a scheduling technique for multi-job MapReduce workloads that is able to dynamically build performance models of the executing workloads, and then use these models for scheduling purposes. As the MapReduce distributed computations were analyzed as a divisible load scheduling problem [21], several classes of algorithms were proposed and examined for scheduling divisible loads on a heterogeneous system with memory limits [22]. Some task scheduling algorithms are to release the data communication among remote slots, e.g., the center-of-gravity reduce scheduler is a locality-aware and skew-aware reduce task scheduler for saving MapReduce network traffic [23], and MaRCO employs eager reduce to process partial data from some map tasks while overlapping with other map tasks' communication [6].

Furthermore, there are also many researches using MapReduce to resolve the big data process [24]. A MapReduce-based framework for HPC analytics was developed in [25] to eliminate the multiple scans and also reduce the number of data preprocessing MapReduce programs. Considering the dynamic resource allocation for the IaaS cloud systems, the algorithms in [26] can adjust the resource allocation dynamically based on the updated information of the actual task executions. In these data processes, the utility becomes a considerable problem naturally. The authors of [10] discussed how to increase the utilization of MapReduce clusters to minimize their cost. They optimized the execution of MapReduce jobs on the cluster through the design of a job schedule that minimizes the completion time (makespan) of such a set of MapReduce jobs. To achieve the optimal user utility, the goal of resource provisioning in [27] is to minimize the cost of virtual machines for executing MapReduce applications. For the practical applications, some aspects of reality should be taken into account, such as fault tolerance [28] and energy efficiency [29,30].

In this article, it is not possible to discuss every related scheme. Hence, we only outline a few closely related papers as above. The main focus of most current works about MapReduce scheduling algorithms appears to be job scheduling, less involving task delay, especially the consideration of the tasks with the same key for the input data block. Through analysis of the intermediate data process in Hadoop, this paper indicates that the scheduling of reduce tasks is one of the key problems which affect the performance of a system.

## 3. Problem analysis

Hadoop allows the user to configure the job, submit it, control its execution, and query the state. Every job consists of independent tasks, and each task needs to have a system slot to run. Fig. 1 shows the time delay and slot resources waste problem in reduce scheduling. $Y$-axis in Fig. 1 means the resource slots, which is represented by Map slots and Reduce slots. At first in Fig. 1(a), we can know that $Job_1$ and $Job_2$ are the current running jobs, before the time $t_2$, each job is allocated two map slots to run respective tasks. Because the reduce tasks will begin once any map task finishes,




(a) $Job_2$ map tasks finished. (b) $Job_2$ reduce tasks finished. (c) $Job_3$ submitted.

**Fig. 1.** The performance of the policies with respect to various graph sizes.

from the duration $t_1$ to $t_2$, there are two reduce tasks from $Job_1$ and $Job_2$ running respectively. Since the execution time of each task is usually not the same, $Job_2$ finishes its map tasks at time $t_2$ and reduce tasks at time $t_3$. Because the reduce tasks of $Job_1$ are not finish in this time $t_3$, as indicated in Fig. 1(b), there are two new reduce tasks from $Job_1$ started. Now all the reduce slot resources are taken up by $Job_1$. In Fig. 1(c), at the moment $t_4$, when $Job_3$ starts, two idle map slots can be assigned to it, and the reduce tasks from this job will then start. However, we can find all reduce slots are already occupied by the $Job_1$, and the reduce tasks from $Job_3$ have to wait for slot release.

The root cause of this problem is that reduce task of $Job_3$ must wait for all the reduce tasks of $Job_1$ to be completed, as $Job_1$ takes up all the reduce slots and Hadoop system does not support preemptive action acquiescently. In the early algorithm design, reduce tasks can be scheduled once any map tasks are finished [31]. One of the benefits is that the reduce tasks can copy the output of the map tasks as soon as possible. But reduce tasks will have to wait before all map tasks are finished, and the pending tasks will always occupy the slot resources so that other jobs which finish the map tasks cannot start the reduce tasks. All in all, this will result in the long waiting of reduce tasks and greatly increase the delay of the Hadoop jobs.

In practice, a shared cluster environment often runs in parallel multiple jobs that are submitted by different users. If the above situation appears among different users at the same time, and the reduce slot resources are occupied for a long time, the submitted jobs from other users will not be pushed ahead until the slots are released. Such inefficiency will extend the average response time of a Hadoop system, lower the resource utilization rate, and affect the throughput of a Hadoop cluster.

## 4. A self-adaptive reduce scheduling algorithm

### 4.1. Runtime analysis of MapReduce jobs

Through the above analysis, one method to optimize the MapReduce tasks is to select an adaptive time to schedule the reduce tasks. By this means, we can avoid the reduce tasks' waiting around and enhance the resource utilization rate. This section proposes a self-adaptive reduce task scheduling algorithm, which gives a method to estimate the start time of a task, instead of the traditional mechanism where the reduce tasks are started once any map task is completed.

The reduce process can be divided into the following several phases. Firstly, the reduce task requests to read each map output

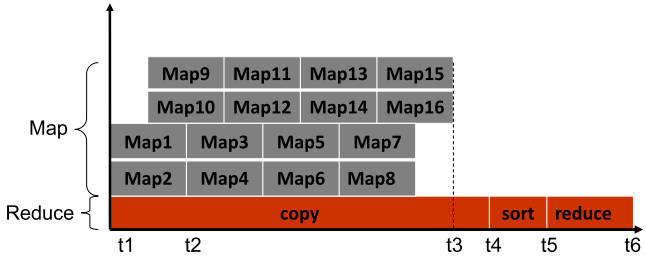



**Fig. 2.** The default scheduling for reduce task.

data in the copy phase, which belong to this reduce function in the map output data blocks. Next, in the sort process, these intermediate data are output to an ordered data set by merging, which are divided into two types. One type is the data in memory. When the data are read from various maps at the same time, the data should be merged as the same keys. The other is as like the circle buffer. Because the memory belonging to the reduce task is limited, the data in the buffer should be written to disks regularly in advance.

In this way, subsequent data need to be merged by the data which are written into the disk earlier, the so called external sorting. The external sorting need to be executed several times if the number of map tasks are large in the practical works. The copy and sort are customarily called the shuffle phase. Finally, after finishing the copy and sort process, the subsequent functions start, and the reduce tasks can be scheduled to the compute nodes.

### 4.2. An advanced reduce scheduling algorithm

Because Hadoop employs the greedy strategy to schedule the reduce tasks, to schedule the reduce tasks fastest, as described above, some reduce tasks will always take up the system resources without actually performing operations in a long time. The reduce task start time is determined by this advanced algorithm SARS (Self-Adaptive Reduce Scheduling). In this method, the start times of the reduce tasks are delayed for a certain duration to lessen the utilization of system resources. The SARS algorithm schedules the reduce tasks at a special moment, when some map tasks are finished but not all. By this means, how to select an optimal time point to start the reduce scheduling is the key problem of the algorithm. Distinctly, the optimum point can minimize the system delay and maximize the resource utilization.

As shown in Fig. 2, assuming that $Job_1$ has 16 map tasks and one reduce task, and there are 4 map slots and only one reduce slot in

this cluster system. Figs. 2 and 3 describe the time constitution of the life cycle for a special job:

$$(FT_{lm} - ST_{fm}) + (FT_{cp} - FT_{lm}) + (FT_{lr} - ST_{sr}), \quad (1)$$

where the denotations of this equation are as follows:

$FT_{lm}$: the completion time of the last map task;
$ST_{fm}$: the start time of the first map task;
$FT_{cp}$: the finish time of copy phase;
$FT_{lr}$: the finish time of Reduce;
$ST_{sr}$: the start time of reduce sort.

In Fig. 2, $t_1$ is the start time of $Map_1$, $Map_2$, and the reduce task. During $t_1$ to $t_3$, the main work of the reduce task is to copy the output from $Map_1$ to $Map_{14}$. The output of $Map_{15}$ and $Map_{16}$ will be copied by the reduce task from $t_3$ to $t_4$. The duration from $t_4$ to $t_5$ is so called the sort stage, which ranks the intermediate results according to the key values. The reduce function is called at the time $t_5$, which continues from $t_5$ to $t_6$. Because during $t_1$ to $t_3$, in the copy phase, the reduce task only copies the output data intermittently, once any map task is completed, for the most time it is always waiting around. We hope to make the copy operations completed at a concentrated duration, which can decrease the waiting time of the reduce tasks.

As Fig. 3 shows, if we can start the reduce tasks at $t_2'$, which can be calculated using Eq. (3), and make sure these tasks can be finished before $t_6$, then during $t_1$ to $t_2'$, the slots can be used by any other reduce tasks. But if we let the copy operation start at $t_3$, because the output of all map tasks should be copied from $t_3$, the delay will be produced in this case. As shown in Fig. 2, the copy phase starts at $t_2$, which just collects the output of the map tasks intermittently. By contrast, the reduce task's waiting time is decreased obviously in Fig. 3, in which case the copy operations are started at $t_2'$.

The SARS algorithm works by delaying the reduce processes. The reduce tasks are scheduled when part but not all of the map tasks are finished. For a special key value, if we assume that there are $s$ map slots and $m$ map tasks in the current system, the completion time and the size of output data of each map task are denoted as $t\_map_i$ and $m\_out_j$ respectively, where $i, j \in [1, m]$. Then, we can know the amount of the map tasks data can be calculated as:

$$N\_m = \sum_{j=1}^{m} m\_out_j, \quad j \in [1, m]. \quad (2)$$

In order to predict the time required to transmit the data, this paper defines the speed of data transmission from the map tasks to the reduce tasks as *transSpeed* in the cluster environment, and the number of concurrent copy threads with reduce tasks is denoted as *copyThread*. We denote the start time of the first map and reduce task as $start_{map}$ and $start_{reduce}$ respectively. Therefore, the optimal start time of reduce tasks can be determined by the following equation:

$$start_{reduce} = start_{map} + \frac{1}{s}\sum_{i=1}^{m} t\_map_i - \frac{N\_m}{transSpeed \times copyThread}. \quad (3)$$

As shown by the time $t_2'$ in Fig. 3, the most appropriate start time of a reduce task is when all the map tasks about the same key are finished, which is between the times when the first map is started and when the last map is finished. On the other hand, for map and copy processes start almost at the same time, which is called intermittent copy (once map has an output, copying followed),

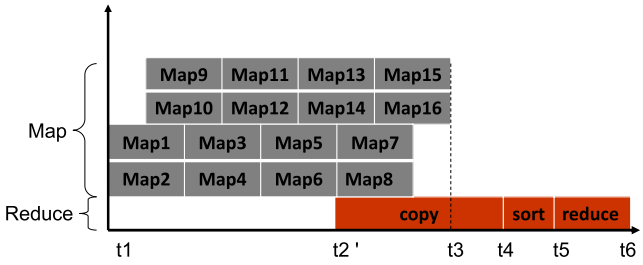



**Fig. 3.** The scheduling method for reduce task in SARS.

causes the massive waste of slot resources. For the reduce tasks will be started before the map task finished, the time cost should be cut from the map tasks completion time. Therefore, we use the average completion time of map tasks to minus data transmission time which produced by map task in Eq. (3), the value can be roughly considered as the starting time of the no-intermission copy, which is reduce tasks starting time in this paper. The waiting around of the reduce tasks may make the jobs in need of the slot resources not able to work normally. Through adjusting the reduce scheduling time, this method can decrease the time waste for data replication process and advance the utilization of the reduce slot resources effectively.

**Algorithm 1** SARS Scheduling Algorithm

**Require:**
$M$: the map slots collection;
$R$: the reduce slots collection;
$Q$: the job queue.

```
MT = obtainMapTask(M, Q);
start the map task in MT;
obtain KVS as the middle key-value set;
for each job ∈ Q do
  for each key ∈ KVS do
    S_MapTask = 0; //all maps' finish time
    S_MapOut = 0; //all maps' output
    for each i ∈ [1, m] do
      S_MapTask+ = t_map_i;
      S_MapOut+ = m_out_i;
    end for
    //get the starting time of reduce task:
    start_reduce = start_map + S_MapTask/s − S_MapOut/(transSpeed ∗ copyThread);
    if System.currentTime = start_reduce then
      if job exists waiting reduce task then
        remove a reduce slot from R;
        start the reduce task;
      else
        break;
      end if
    else
      break;
    end if
  end for
end for
```

Using the job's own characteristics to determine the reduce scheduling time can use the slot resources effectively. The improvement of these policies is especially important for the CPU-type jobs. For these jobs which need more CPU computing, the data I/O of the tasks are less, so more slot resources will be wasted in the default schedule algorithm.

**Table 1**
The software and hardware configurations in the Hadoop cluster.

| The node type | Operating system | CPU | Memory | Quantity |
|---|---|---|---|---|
| NameNode | Open suse 11 | 4-core, 3.07 GHz | 4 G | 1 |
| JobTracker | Open suse 11 | 4-core, 2.7 GHz | 4 G | 1 |
| DataNode/TaskTracker | Open suse 11 | 4-core, 2.7 GHZ | 4 G | 5 |

**Algorithm 2** obtainMapTask

**Require:**
$M$: the map slots collection;
$Q$: the job queue.
**Ensure:**
$MT$: map task set;
$m$: the count of current map tasks.
1: **for** each $job \in Q$ **do**
2: $count = 0$;
3: **if** $job$ has a waiting map task $t$ **then**
4: $MT = MT \cup \{t\}$;
5: remove a map slot from $M$;
6: $count$++;
7: **else**
8: break;
9: **end if**
10: **end for**
11: $m = count$;
12: **return** $MT$.

The pseudo code of the SARS scheduling algorithm is shown in Algorithms 1 and 2. The time complexity of the algorithm is $O(n \times K \times m)$ due to the three-level nested for-loops, where $n$ is the length of the job queue, $m$ is the number of map tasks, which outputs $k$ middle output keys.

## 5. Experimental evaluation

The purpose of the scheduling algorithm SARS is to shorten the copy duration of the reduce process, decrease the task complete time, and save the reduce slot resources. Through measuring the job completion time and the reduce process completion time, this section compares this method to the traditional scheduling algorithms like First-In-First-Out (FIFO), FairScheduler (FAIR), and CapacityScheduler (CS), and evaluates the effects for the average finish time of the submitted jobs employed by this algorithm.

### 5.1. The experimental benchmarks

We choose these typical MapReduce-based benchmarks to test the MapReduce and HDFS performance employing the SARS algorithm: *WordCount*, *Pi*, *MRBench*, *TeraSort*, and *GridMix*.

The *WordCount* benchmark reads text files and counts how often words occur. Each mapper takes a line as input and breaks it into words. It then emits a key/value pair of the word and 1. Each reducer sums the counts for each word and emits a single key/value with the word and sum.

The *Pi* benchmark is a Monte Carlo algorithm which computes the exact binary digits of the mathematical constant $\pi$. It is designed for computing the $n$th bit of $\pi$, for large $n$, say $n$ is larger than 100,000,000.

The *MRBench* checks whether small jobs are responsive and running efficiently on the cluster. It focuses on the MapReduce layer since its impact on the HDFS layer is very limited.

The *TeraSort* benchmark is to sort 1 TB of data (or any other amount of data you want) as fast as possible. It is a benchmark that combines testing the HDFS and MapReduce layers of a Hadoop cluster.

**Table 2**
Key configuration in the Hadoop cluster.

| Configuration items | Configuration properties | Value |
|---|---|---|
| Map slots | mapred.tasktracker.map.tasks.maximum | 4 |
| Reduce slot | mapred.tasktracker.Reduce.tasks.maximum | 2 |
| Copy thread | mapred.reduce.parallel.copies | 5 |
| HDFS replications | dfs.replication | 1 |
| Input file size | dfs.block.size | 64 M |
| DataNode heartbeat | dfs.heartbeat.interval | 3 s |

The *GridMix* is a benchmark for the Hadoop cluster. It is mainly used to test the performance of job execution in a cluster, which submits a mix of synthetic jobs, modeling a profile mined from production loads.

### 5.2. The experimental environment

The experiment environment is based on Hadoop platform with the version number 1.20. The hardware and software configurations are shown in Table 1. Furthermore, this Hadoop platform still has some key configurations shown in Table 2.

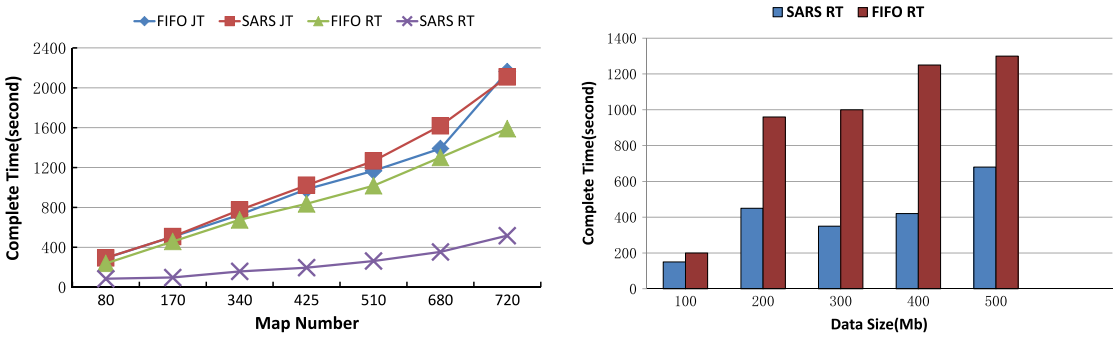
Since many kinds of MapReduce jobs are I/O-bound, an underperforming or poorly configured disk can drastically reduce overall job performance. Datanodes store block data on top of a traditional file system rather than on raw devices. Hadoop is designed to be modest in its requirements on the hosts on which it runs. In our experiments, the disk configuration is ext4 file system.

For the limitation of the cluster scale, the number of the file replications is set to 1, and the numbers of maps and reduce slots are respectively set as 4 and 2. Copy thread in this table refers to the number of the threads which fetch the data blocks from the map tasks. Because the HDFS system is composed of data block, the file size is configurable and set to 64 M in this experiment. Finally, the DataNode heartbeat time refers to the frequency of contacting with the NameNode.

### 5.3. Completion time evaluation

Completion time evaluation includes two aspects: the whole job completion time and the reduce task completion time. Comparing job completion time is to assess whether the proposed algorithm may lead to delay for job itself. And for illustrating the advantage of SARS, we also estimate the reduce completion time in this experiment. For the accuracy of the result, the running job is completely isolated, in other words, there is only one job running at all times. The legends of experiment results are as follows:

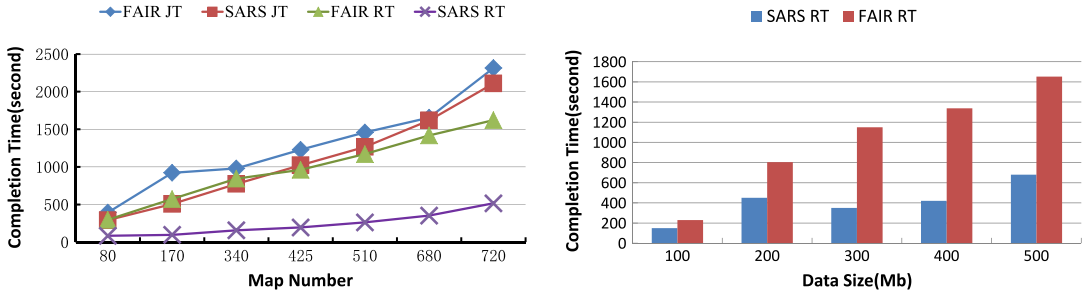
(1) FIFO JT: The completion time of the whole job scheduled by algorithm FIFO;
(2) FAIR JT: The completion time of the whole job scheduled by algorithm FAIR;
(3) CS JT: The completion time of the whole job scheduled by algorithm CS;
(4) SARS JT: The completion time of the whole job scheduled by algorithm SARS;
(5) FIFO RT: The completion time of the reduce task scheduled by algorithm FIFO;
(6) FAIR RT: The completion time of the reduce task scheduled by algorithm FAIR;




(a) Performance comparison between FIFO and SARS algorithms for WordCount.

(b) Comparison of reduce completion time between FIFO and SARS algorithms for WordCount.

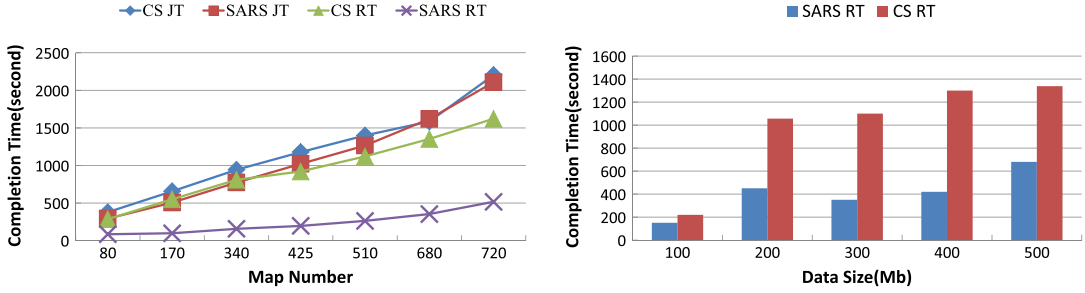
**Fig. 4.** Comparison between FIFO and SARS algorithms on WordCount.




(a) Performance comparison between FAIR and SARS algorithms for WordCount.

(b) Comparison of reduce completion time between FAIR and SARS algorithms for WordCount.

**Fig. 5.** Comparison between FAIR and SARS algorithms on WordCount.




(a) Performance comparison between CS and SARS algorithms for WordCount.

(b) Comparison of reduce completion time between CS and SARS algorithms for WordCount.

**Fig. 6.** Comparison between CS and SARS algorithms on WordCount.

(7) CS RT: The completion time of the reduce task scheduled by algorithm CS;
(8) SARS RT: The completion time of the reduce task scheduled by algorithm SARS.

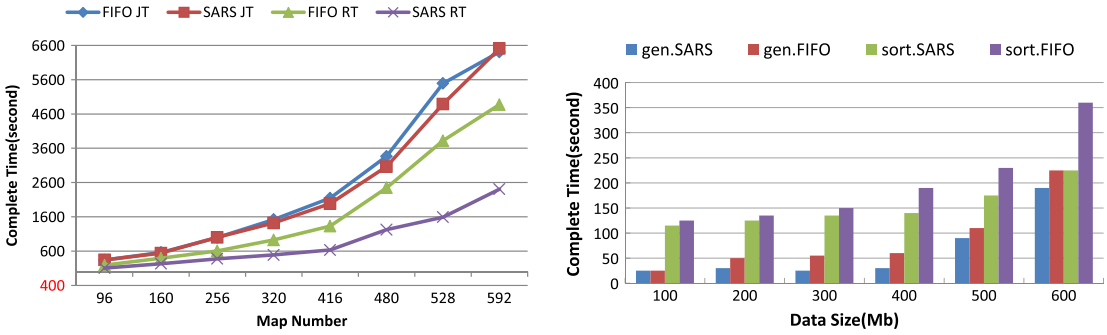
In these experiments, map number represents the number of the current map tasks, and data size represents the size of the map task processing data. For the limitation of the slot resources, we can find that the job complete time will increase when the map number substantially exceeds the system capacity.

Fig. 4(a) shows the comparison between algorithms FIFO and SARS using the benchmark WordCount. From the curves, we can know that their completion times of the whole job scheduled are almost equal, except when the map number is 680, the value of SARS JT is greater than FIFO JT distinctly. However, when comparing the reduce task completion time, the result reveals that the completion time SARS RT is far less than FIFO RT. According to our statistics, 3–4 times speedup is achieved when comparing the SARS RT to FIFO RT.

Fig. 4(b) shows the WordCount performance when employing the FIFO and SARS algorithms with 16 nodes. The input data is chosen from the GENIA corpus v.3.02 (a semantically annotated corpus for bio-text mining). From the figure, it is obvious that the running time increases as the size of input data scales. Furthermore, the SARS algorithm acquires better performance compared to the normal case, which means the MapReduce performance can be obviously affected by the starting moments of the reduce tasks.

Figs. 5 and 6 show the comparison among FAIR, CS, and SARS algorithms using the benchmark WordCount. The whole job completion time of SARS is noticeably and consistently shorter than that of FAIR and CS. Furthermore, compared with both FAIR and CS algorithms, SARS decreases the reduce execution time significantly.

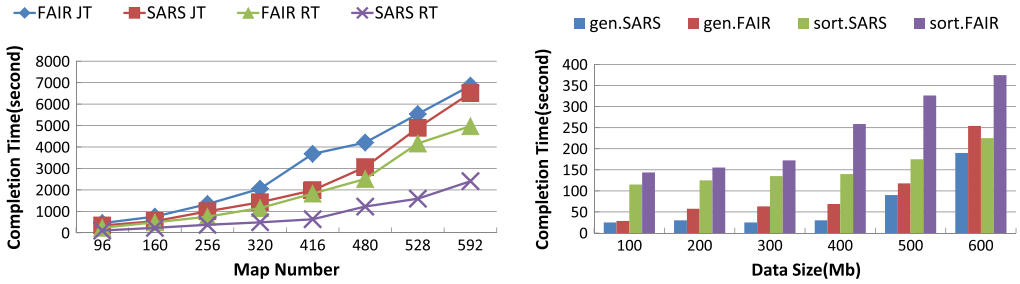
The comparison results between FIFO and SARS algorithms on TeraSort are shown in Fig. 7. Similar to the WordCount job, the sort completion time of FIFO and SARS algorithms are nearly equal, with SARS JT slightly shorter. However, the reduce task completion




(a) Performance comparison between FIFO and SARS algorithms for TeraSort.

(b) Comparison of reduce completion time between FIFO and SARS algorithms for TeraSort.

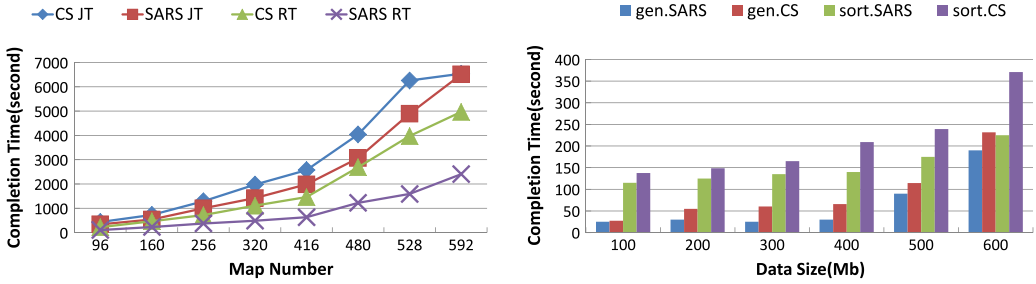
**Fig. 7.** Comparison between FIFO and SARS algorithms on TeraSort.




(a) Performance comparison between FAIR and SARS algorithms for TeraSort.

(b) Comparison of reduce completion time between FAIR and SARS algorithms for TeraSort.

**Fig. 8.** Comparison between FAIR and SARS algorithms on TeraSort.




(a) Performance comparison between CS and SARS algorithms for TeraSort.

(b) Comparison of reduce completion time between CS and SARS algorithms for TeraSort.

**Fig. 9.** Comparison between CS and SARS algorithms on TeraSort.

time of FIFO is around 1–2 times higher than that of SARS. Fig. 7(b) shows both the data generation time and the sort time of TeraSort benchmark. From the figure, we find that when the data size is small, both the data generation time and sort time are relatively small. However, when the data size exceeds 400 MB, the running time increases quickly. And from the experimental data, we can acquire that the performances of data generation and sort process are all advanced by the SARS algorithm.

Figs. 8 and 9 show the comparison among FAIR, CS, and SARS algorithms using the benchmark TeraSort. Again, we observe noticeable and consistent reduction in job completion time. The data generation time and sort time are also reduced. Furthermore, the SARS algorithm is more appropriate in large-scale data processing.

Figs. 10–12 show the performance comparison among FIFO, FAIR, CS, and SARS for the benchmark *Pi*, where we scale the number of maps from 100 to 800. Except the map number near to 600, SARS can achieve significant reduction in job completion time. Furthermore, the reduce completion time is reduced by a factor of 1–6.

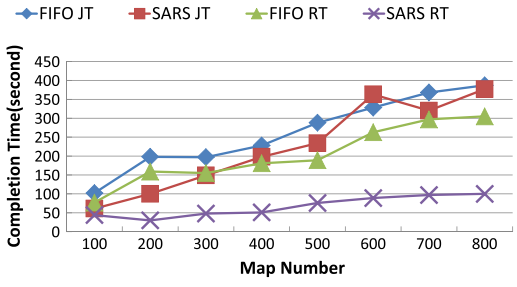



**Fig. 10.** Comparison between FIFO and SARS algorithms for *Pi*.

Figs. 13–15 show the MRBench performance, where we scale the number of maps from 100 to 600. Except map number 300, SARS achieves noticeable reduction in job completion time. Furthermore, the reduce completion time is reduced by a factor of 1–2.

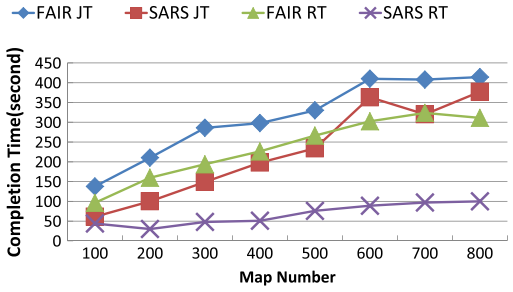



**Fig. 11.** Comparison between FAIR and SARS algorithms for *Pi*.

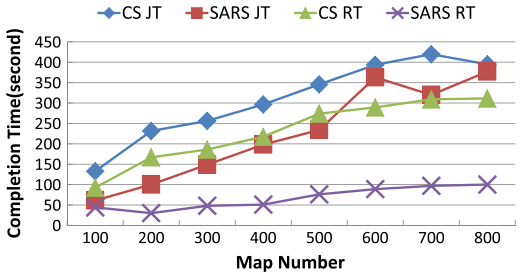



**Fig. 12.** Comparison between CS and SARS algorithms for *Pi*.

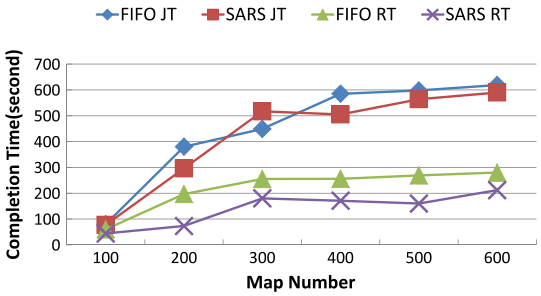



**Fig. 13.** Comparison between FIFO and SARS algorithms for MRBench.

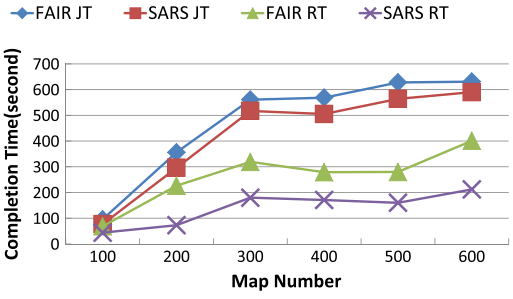



**Fig. 14.** Comparison between FAIR and SARS algorithms for MRBench.

It is clear from the above experiments that the SARS algorithm can sharply decrease the reduce task completion time. Our experimental results also illustrate that although the reduce processes are delayed to start, the whole completion time of a single job has not been extended. But in this way, the system can obtain a lot of idle slots. If these slots can be reallocated by reduce tasks from other jobs, the whole completion time will be decreased when more than one job are running together. How to schedule the idle reduce slots in a multiple-job environment will be resolved in our future works.

### 5.4. Average response time evaluation

The purpose of this experiment is to evaluate the average response time when multiple jobs are submitted by multiple users

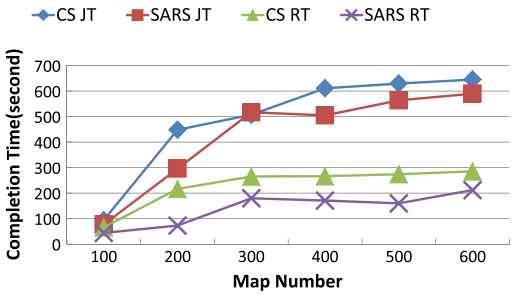



**Fig. 15.** Comparison between CS and SARS algorithms for MRBench.

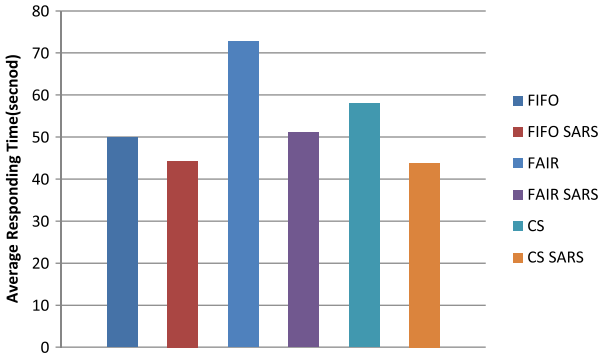



**Fig. 16.** The average response time of experimental jobs.

in the cluster. For appraising the running situation of multiple jobs in a complex cluster environment, this paper uses the Gridmix to submit jobs, which can analyze the Hadoop performance by simulating the actual load of a Hadoop cluster. At first, this experiment can generate a lot of input data and jobs according to the pre-set parameters. After these jobs are submitted from batch processing, they can be grouped into five categories:

(1) MapReduce jobs; (A: map tasks reserve 10% of the data, while reduce tasks reserve 40%; B: map tasks reserve 10% data, and reduce tasks reserve 40% data, output from A; C: map tasks reserve 10% data, and reduce tasks reserve 40% data, output from B.)
(2) The sort jobs for mass data, where key and value length are variable;
(3) The filter for large data sets; (Map tasks reserve 0.2% of the data, while reduce tasks reserve 5% of the rest.)
(4) The text sort through directly API invoking;
(5) The WordCount job including Combiners.

The data size and the number of the categories of these experiment jobs can be configured from Gridmix, and these jobs can be monitored and measured during their whole running process.

In consideration of the small scale of the experimental cluster, only 24 jobs are submitted in Gridmix. We compare the average response time of these algorithms before and after applying the SARS policy: FIFO, FairScheduler, and CapacityScheduler. The main evaluation indicator is the system response time with high workload.

Fig. 16 shows the results of this comparison, where the denotation "CS" represents the CapacityScheduler scheduling algorithm, and "FAIR" means the FairScheduler scheduling algorithm. From these results, we know that the SARS algorithm can advance the performance of existing schedulers. After applying the SARS policy to the traditional job scheduling algorithms, the average response time can be decreased by 11% to 29%.

## 6. Conclusion

The goal of the improved algorithm proposed in this paper is to decrease the completion time of reduce tasks in the MapReduce framework. This method can decide reduce task start time through running situation of the jobs, and to avoid waiting around of the reduce function. In this paper, the performance of this algorithm is estimated from the job completion time, reduce completion time, and the system average response time. The experimental results illustrate that when comparing with the FIFO, the reduce completion time is decreased sharply. It is also proved that the average response time is decreased by 11% to 29% when the SARS algorithm is applied to traditional job scheduling algorithms FIFO, FairScheduler, and CapacityScheduler.

However, there are also some difficulties in the experiments. The first is the speed of the network transmission. When multiple reduce tasks from one TaskTracker run at the same time, they may lead to the network I/O competition and lower the transmission speed during the copy stage. Moreover, this study is based on a homogeneous cluster environment. Since the SARS algorithm assumes that all of the map task execution times and output data are consistent, in a heterogeneous environment, the SARS algorithm needs to be further improved.

## Acknowledgments

The authors are grateful to the three anonymous reviewers for their criticism and comments which have helped to improve the presentation and quality of the paper. This work is supported by the Key Program of National Natural Science Foundation of China (Grant No. 61133005), and National Natural Science Foundation of China (Grant Nos. 61103047 and 61370095).

## References

[1] J. Dean, S. Ghemawat, MapReduce: simplified data processing on large clusters, in: Proc. 2004 Symposium on Operating System Design and Implementation, pp. 137–150.

[2] Xiang Gao, Qing Chen, Yurong Chen, Qingwei Sun, Yan Liu, Mingzhu Li, A dispatching-rule-based task scheduling policy for MapReduce with multi-type jobs in heterogeneous environments, in: 2012 Seventh ChinaGrid Annual Conference (ChinaGrid), pp. 17–24.

[3] Jiaqi Zhao, Lizhe Wang, Jie Tao, Jinjun Chen, Weiye Sun, Rajiv Ranjan, Joanna Kolodziej, Achim Streit, Dimitrios Georgakopoulos: a security framework in G-Hadoop for big data computing across distributed Cloud data centres, J. Comput. System Sci. 80 (5) (2014) 994–1007.

[4] Jiong Xie, FanJun Meng, HaiLong Wang, HongFang Pan, JinHong Cheng, Xiao Qin, Research on scheduling scheme for hadoop clusters, Proc. Comput. Sci. 18 (2013) 2468–2471.

[5] Zhuo Tang, Min Liu, Kenli Li, Yuming Xu, A MapReduce-enabled scientific workflow framework with optimization scheduling algorithm, in: 2012 13th International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT), pp. 599–604.

[6] Faraz Ahmad, Seyong Lee, Mithuna Thottethodi, T.N. Vijaykumar, MapReduce with communication overlap (MaRCO), J. Parallel Distrib. Comput. 73 (5) (2013) 608–620.

[7] Minghong Lin, Li Zhang, Adam Wierman, Jian Tan, Joint optimization of overlapping phases in MapReduce, Perform. Eval. 70 (10) (2013) 720–735.

[8] T. Sandholm, K. Lai, Dynamic proportional share scheduling in hadoop, in: Proceedings of the 15th Workshop on Job Scheduling Strategies for Parallel Processing, 2010, pp.110–131.

[9] M. Zaharia, D. Borthakur, J.S. Sarma, K. Elmeleegy, S. Shenker, I. Stoica, Delay scheduling: a simple technique for achieving locality and fairness in cluster scheduling, in: Proceedings of the 5th European Conference on Computer systems, 2010, pp. 265–278.

[10] Abhishek Verma, Ludmila Cherkasova, Roy H. Campbell, Orchestrating an ensemble of MapReduce jobs for minimizing their makespan, IEEE Trans. Depend. Secure Comput. (2013) 314–327.

[11] Kristi Morton, Magdalena Balazinska, Dan Grossman, Paratimer: a progress indicator for mapreduce DAGs, in: Proceedings of the 2010 International Conference on Management of Data, 2010, pp. 507–518. http://dx.doi.org/10.1145/1807167.1807223.

[12] Hameed Hussain, Saif Ur Rehman Malik, Abdul Hameed, Samee Ullah Khan, Gage Bickler, Nasro Min-Allah, Muhammad Bilal Qureshi, Limin Zhang, Yongji Wang, Nasir Ghani, Joanna Kolodziej, Albert Y. Zomaya, Cheng-Zhong Xu, Pavan Balaji, Abhinav Vishnu, Frédéric Pinel, Johnatan E. Pecero, Dzmitry Kliazovich, Pascal Bouvry, Hongxiang Li, Lizhe Wang, Dan Chen, Ammar Rayes, A survey on resource allocation in high performance distributed computing systems, Parallel Comput. 39 (11) (2013) 709–736.

[13] Yuan Luo, Beth Plale, Hierarchical MapReduce Programming Model and Scheduling Algorithms, in: Cluster Computing and the Grid, IEEE International Symposium on, pp. 769–774.

[14] Joanna Kolodziej, Samee Ullah Khan, Lizhe Wang, Aleksander Byrski, Nasro Min-Allah, Sajjad Ahmad Madani, Hierarchical genetic-based grid scheduling with energy optimization, Cluster Comput. 16 (3) (2013) 591–609.

[15] Hisham Mohamed, Stéphane Marchand-Maillet, MRO-MPI: MapReduce overlapping using MPI and an optimized data exchange policy, Parallel Comput. 39 (12) (2013) 851–866.

[16] Abhishek Verma, Ludmila Cherkasova, Roy H. Campbell, Two sides of a coin: optimizing the schedule of MapReduce jobs to minimize their makespan and improve cluster performance, in: 2012 IEEE 20th International Symposium on Modelling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS), pp. 11–18.

[17] Zhang Xiaohong, Ju Shui, Jiao Zhibin, A scheduling method based on deadlines in MapReduce, in: Proceedings of the 2011 International Conference on Electrical, Information Engineering and Mechatronics (EIEM), pp. 1585–1592.

[18] Zhuo Tang, Junqing Zhou, Kenli Li, Ruixuan Li, A MapReduce task scheduling algorithm for deadline constraints, Cluster Comput. 16 (4) (2013) 651–662.

[19] Peng Lu, Young Choon Lee, Chen Wang, Bing Bing Zhou, Junliang Chen, Albert Y. Zomaya, Workload characteristic oriented scheduler for MapReduce, in: 2012 IEEE 18th International Conference on Parallel and Distributed Systems (ICPADS), pp. 156–163.

[20] J. Polo, Y. Becerra, D. Carrera, M. Steinder, I. Whalley, J. Torres, E. Ayguade, Deadline-based MapReduce workload management, IEEE Trans. Netw. Serv. Manag. 10 (2) (2013) 231–244.

[21] J. Berliska, M. Drozdowski, Scheduling divisible MapReduce computations, J. Parallel Distrib. Comput. 71 (3) (2011) 450–459.

[22] J. Berlinska, M. Drozdowski, Heuristics for multi-round divisible loads scheduling with limited memory, Parallel Comput. 36 (4) (2010).

[23] Mohammad Hammoud, M. Suhail Rehman, Majd F. Sakr, Center-of-gravity reduce task scheduling to lower MapReduce network traffic, in: 2012 IEEE 5th International Conference on Cloud Computing (CLOUD), pp. 49–58.

[24] Lizhe Wang, Jie Tao, Rajiv Ranjan, Holger Marten, Achim Streit, Jingying Chen, Dan Chen, G-Hadoop: MapReduce across distributed data centers for data-intensive computing, Future Gener. Comput. Syst. 29 (3) (2013) 739–750.

[25] Saba Sehrish, Grant Mackey, Pengju Shang, Jun Wang, John Bent, Supporting HPC analytics applications with access patterns using data restructuring and data-centric scheduling techniques in MapReduce, IEEE Trans. Parallel Distrib. Syst. (2013) 158–169.

[26] Jiayin Li, Meikang Qiu, Zhong Ming, Gang Quan, Xiao Qin, Zonghua Gu, Online optimization for scheduling preemptable tasks on IaaS cloud systems, J. Parallel Distrib. Comput. 72 (5) (2012).

[27] Eunji Hwang, Kyong Hoon Kim, Minimizing cost of virtual machines for deadline-constrained MapReduce applications in the cloud, in: 2012 13th IEEE/ACM International Conference on Grid Computing (GRID), pp. 130–138.

[28] Zhifeng Xiao, Yang Xiao, Achieving accountable MapReduce in cloud computing, Future Gener. Comput. Syst. 30 (2014) 1–13.

[29] Nitesh Maheshwari, Radheshyam Nanduri, Vasudeva Varma, Dynamic energy efficient data placement and cluster reconfiguration algorithm for MapReduce framework, Future Gener. Comput. Syst. 28 (1) (2012) 119–127.

[30] Xiaoli Wang, Yuping Wang, An energy and data locality aware bi-level multiobjective task scheduling model based on MapReduce for cloud computing, in: 2012 IEEE/WIC/ACM International Joint Conferences on Web Intelligence (WI) and Intelligent Agent Technologies (IAT), pp. 648–655.

[31] Balaji Palanisamy, Aameek Singh, Ling Liu, Bryan Langston, Cura: a cost-optimized model for MapReduce in a cloud, in: 2013 IEEE International Symposium on Parallel and Distributed Processing (IPDPS), pp. 1275–1286.

**Zhuo Tang** received the Ph.D. in computer science from Huazhong University of Science and Technology, China, in 2008. He is currently an assistant professor of Computer science and Technology at Hunan University. His research interests include security model, parallel algorithms and resources scheduling for distributed computing systems, Grid and Cloud computing. He is a member of CCF.

**Lingang Jiang** is working towards the Master degree at the College of Information Science and Engineering, Hunan University of China. His research interests include modeling and scheduling for distributed computing systems, Parallel algorithms.

**Junqing Zhou** is working towards the Master degree at the College of Information Science and Engineering, Hunan University of China. His research interests include modeling and scheduling for distributed computing systems, Parallel algorithms.

**Kenli Li** received the Ph.D. in computer science from Huazhong University of Science and Technology, China, in 2003. He was a visiting scholar at University of Illinois at Champaign and Urbana from 2004 to 2005. Now He is a professor of Computer science and Technology at Hunan University, associate director of National Supercomputing Center in Changsha. His major research includes parallel computing, Grid and Cloud computing, and DNA computer. He has published more than 70 papers in international conferences and journals, such as IEEE TC, IEEE TPDS, JPDC, ICPP, CCGrid. He is an outstanding member of CCF.

**Keqin Li** received the B.S. degree in computer science from Tsinghua University, Beijing, in 1985 and the Ph.D. degree in computer science from the University of Houston in 1990. He is currently a full professor of computer science in the Department of Computer Science, State University of New York, New Paltz. His research interests are mainly in the design and analysis of algorithms, parallel and distributed computing, and computer networking, with particular interests in approximation algorithms, parallel algorithms, job scheduling, task dispatching, load balancing, performance evaluation, dynamic tree embedding, scalability analysis, parallel computing using optical interconnects, optical networks, and wireless networks. He has more than 200 research publications. He is a senior member of the IEEE, the IEEE Computer Society, and the ACM.