International Journal of Future Generation Communication and Networking
Vol. 8, No. 2 (2015), pp. 391-398
http://dx.doi.org/10.14257/ijfgcn.2015.8.2.32

An Efficient Job Scheduling for MapReduce Clusters

Jun Liu[1], Tianshu Wu[1], and Ming Wei Lin[1] and Shuyu Chen[2]

[1]College of Computer Science, Chongqing University, Chongqing, China
[2]College of Software Engineering, Chongqing University, Chongqing, China
liujuncqcs@163.com, netmobilab@cqu.edu.cn, wutianshu@cqu.edu.cn, linmwcs @163.com

Abstract

The job scheduling for Map Reduce clusters has received significant attention in recent years, because it plays an important role on Map Reduce clusters. Traditional job scheduling performs poorly in assigning a task to appropriate nodes, and can not predict the resource utilization of the unexecuted tasks. To address the problems, an efficient job scheduling for Map Reduce clusters is proposed in this paper. The job scheduling introduces dynamic priority scheduling and real-time prediction model. Dynamic priority scheduling introduces the minimum cost data locality algorithm with a weight to deal with different size jobs, and real-time prediction model can predict the resource utilization of unexecuted tasks by calculating the running tasks. The resource utilization contains CPU, memory, and network. Experimental results prove that the proposed job scheduling is able to perform well in Map Reduce clusters.

Keywords: *job scheduling, minimum cost data locality algorithm, and dynamic priority scheduling*

1. Introduction

Due to extremely easy to program, fast speed, scalability, and fault-tolerance achieved for a variety of applications, MapReduce [1-3] has been widely regarded as a promising alternative to large-scale data analysis such as graph processing, machine learning, and data mining. These applications which are submitted to MapReduce clusters are executed in the form of jobs, and each job contains a number of tasks. Every task will be assigned to a node, which is generally called slave node, by task scheduler in clusters. Task scheduler is one of the core technologies of MapReduce, it mainly controls the order of task executing and resource allocation. In addition, it can directly influence the performance of MapReduce clusters and the execution time of the different priority tasks. Therefore, an appropriate task scheduling is very important for MapReduce clusters.

MapReduce itself provides three main task scheduling algorithms, which are the First-in-first-out (FIFO), the capacity scheduling [4], and the fair scheduling [5]. First-in-first-out algorithm is the build-in scheduler in MapReduce clusters. The advantages of FIFO are simple and easy to implement, because it deals with the jobs in the way of first in first out, that is, the older job can be deal first, and the younger job can be handled later. However, it does not take fully into account that there are different sizes of jobs including small and large jobs in clusters, and does not consider the support of multiple users. To address the problems of FIFO, the fair scheduling is developed to deal with small and large jobs as fairly as possible in clusters. In order to achieve this goal, job priorities, pool weights, and delay scheduling is introduced. The scheduling of jobs is controlled by job priorities with a suitable weight, and weight is divided into a certain level, such as a weight of 1.0, a weight of 2.0, and 2x more weight. But the fair scheduling needs a lot of manually configuration, which can greatly influence the performance of the jobs. Moreover, the fair scheduling does not take the actual load of the *master node* into account, which plays the role on job scheduling and distributes job to a number of *slave nodes*. Capacity scheduling supports multiple job queues. However capacity scheduling limits the resources that a job can be used.

For above-mentioned reasons, it is quite clear that a valid job scheduling is a comprehensive study on the performance, the support of multiple users, and the effective utilization of resources. To reach this design principle, an efficient job scheduling, which aims to improve the performance of the clusters, and to meet different size jobs, is proposed. The major contributions of this paper can be summarized as follows:

(1).In order to improve the performance of the MapReduce clusters, real-time prediction model will be used to predict the unexecuted tasks. This model estimates the resource consumptions of the unexecuted tasks by calculating current running jobs.

(2).In order to satisfy the different size of jobs, the minimum cost data locality algorithm will be used to calculate the degree of data locality. The algorithm can efficiently avoid the problem that there are different size jobs in clusters, and it can also improve the efficiency of the job scheduling.

To evaluate the effectiveness of the proposed job scheduling, the job scheduling prototype has been implemented and various benchmarks have been conducted. The simulation results show that the proposed job scheduling significantly improves the performance of the MapReduce clusters.

The rest of this paper is organized as follows. Section 2 discusses related work. Section 3 presents the efficient job scheduling. Section 4 presents the experimental results. Section 5 concludes the paper.

2. Related Work

In order to design suitable job scheduling, a number of job scheduling algorithms have been proposed in recent years, this section briefly summarizes research work related to job scheduling in MapReduce clusters.

Zaharia *et al.*, [6] propose a job scheduling, which introduces the delay scheduling algorithm to improve the data locality. However, the algorithm does not take into account of the different size of jobs in cluster. Moreover, the algorithm performs well in small jobs, and performs poor in large jobs, because the delay scheduling algorithm can incur performance degradation in large jobs. In MapReduce, large jobs are divided into fixed number tasks, and there are many waiting tasks in queues due to the delay characteristics in the delay scheduling algorithm. So the execution time of large jobs will be increase. Jinshuang Yan et al. [10] also propose a job scheduling, which has advantages in the time cost during the initial phase of a job, and the task assignment because the push-model replaces the pull-model which is a task assignment mechanism. But, it can not perform well for large jobs, because it is designed for small jobs. Seo *et al*. [7] propose a new job scheduling, which introduces the perfecting technologies to improve the data locality and the performance of the clusters. But a lot of memory consumption and network throughout will be increased, because a number of unrelated data to the task is read to memory. In addition, a mass of data is transmitted through network due to the storage characteristics of HDFS [8-9].

Aprigio Bezerra *e.t al.* [11] propose a job scheduling, which selects tasks from a pending jobs queues by analyzing the available resources of the clusters to improve the performance of the clusters. Although analyzing the available resources of the clusters can appropriately submit a task to the cluster, it can not predict the resource consumptions of the pending jobs. Jisha S Manjaly [12] also propose a job scheduling, called *Task Tracker* aware scheduling algorithm, which aims to avoid the task failure caused by overloading in clusters, and in which users must configure the maximum load for every task by setting the threshold. However, the drawback of this job scheduling is the much configuration to the users. So it is very difficult to use.

Obviously, the above-mentioned job scheduling algorithms have the common drawbacks, which are not predicting the resource consumption of the unexecuted or pending tasks, and calculating the degree of the data locality, respectively.

3. An Efficient Job Scheduling

Based on the analysis of the above job scheduling in MapReduce clusters, an optimized job scheduling is presented in this section. The goal of the job scheduling is to assign resources to jobs fairly. Small and large jobs will be reasonably assigned to each node by analyzing the practical situation of resource utilization through the dynamic priority scheduling in MapReduce Clusters. Moreover, the job scheduling can predict the resource utilization of the jobs which have not been performed by analyzing the performed jobs.

3.1. Dynamic Priority Scheduling

The core of the job scheduling is the dynamic priority scheduling, which introduces the minimum cost data locality algorithm with a weight to deal with different size jobs. The weigh of a job can be defined as

$$W = Locality + \Pr iority \qquad (1)$$

where *priority* is the priority of a job, and *locality* is the degree of data locality. From the formula, we can see that the weight of jobs contains the data locality and the priority of jobs. The priority of jobs is the job execution order defined by users. Data locality is that the corresponding data of a job will be stored in the nodes where jobs are executed. In this paper, the date locality algorithm externs the host selection algorithm in Hadoop [13], and if the corresponding data of a job is divided into different nodes, the proposed job scheduling can calculate minimum cost and assign the job to appropriate nodes by minimum data locality algorithm.

Assume that a job contains *M* blocks. The *M* blocks are stored in different nodes, and the block numbers of each node are denoted as N_1, N_2…, and N_n respectively. The distances of each node to the task scheduling node (*Jobtracker) are denoted as* D_1, D_2*..., and* D_n respectively. T_i represents the time cost that a block is transferred to the node with the maximum number of blocks. The reason of this selection is that the first executed task is assigned to the node with the maximum number of blocks. T_i is relevant with the size of blocks, the number of blocks, and the actual network transmission speed. The size of blocks is denoted as $Block_{size}$ (64MB by default), the number of blocks is denoted as N_i, and the actual network transmission speed is denoted as *speed*. So T_i can be indicated as

$$T_i = \frac{D_i}{N_i \times speed \times Block_{size}} \qquad (2)$$

In this paper, assume that the data blocks of the executed job are distributed into *n* nodes, whose locality are denoted as $Locality_1$*,* $Locality_2$*, ..., and* $locality_i$ respectively. So the $locality_i$ is denoted as

$$Locality_i = \frac{1}{N_i} + (B_f) \times T_i \qquad (3)$$

Where B_f is the data blocks which is transmitted through the network. According to formula (2) and (3), $locality_i$ is equal to:

$$Locality_i = \frac{1}{N_i} + (B_f) \times \frac{D_i}{N_i \times speed \times Block_{size}} \quad (4)$$

The minimum cost data locality algorithm selects the smallest P nodes in {$Locality_1$, $Locality_2$, *and* $locality_i$}. User can predefine the value of P. The P nodes are sorted from smallest to largest according to *locality*. In MapReduce clusters, a job is divided into a fixed number of tasks, and each task is assigned to a node. Figure 1 illustrates an example of the minimum cost data locality algorithm. In this example, a job is divided into 2 tasks. There are 5 data blocks in $node_1$, two data blocks in $node_2$, and 3 blocks in $node_3$. The locality list is accessed in the order of $locality_1$, $locality_3$, and $locality_2$. Obviously, $task_1$ is assigned into $node_1$, and $task_2$ is assigned to $node_3$. The locality of $node_3$ is smaller than $node_2$ because the number of blocks in $node_3$ is larger than $node_2$. From the Figure 1, we can see that there is no task in the $node_2$. Assume that the distance between $node_1$ to $node_2$ is smaller than the distance between $node_2$ to $node_3$. Therefore the data blocks in $node_2$ are processed by the *task 1* in $node_1$.

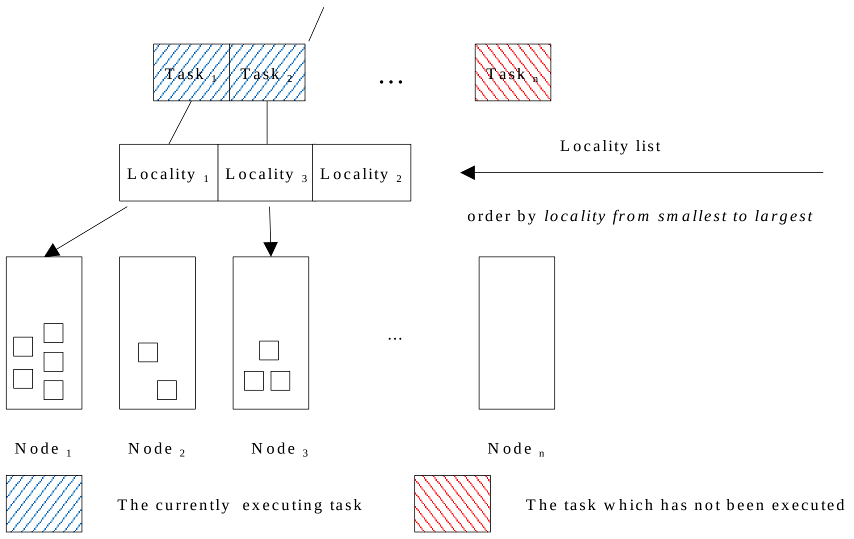

Task 1
Task 2
…
Task n
Locality list
Locality 1
Locality 3
Locality 2
order by locality from smallest to largest
…
Node 1
Node 2
Node 3
Node n
The currently executing task
The task which has not been executed

Figure 1. The Example of the Minimum Cost Data Locality Algorithm

3.2. The Real-Time Prediction Model of Jobs

In real-time prediction model, the resource utilization of the unexecuted tasks can be concluded through the executed tasks. These resources include CPU, memory, and network resources. Assume that a job is divided into ten tasks, which are denoted as $Task_1$, $Task_2$, …, and $Task_n$, whose lengths are denoted as L_1, L_2, …, and, L_n respectively, six of them are running, and the other four are waiting. The resource consumption of a task $Task_i$ with the length L_i in a node is derived as

$$C_c = C_{cpu} + C_{memory} + C_{network} \quad (5)$$

where C_{cpu} stands for the CPU consumption, C_{memroy} stands for memory consumption, and $C_{network}$ stands for the consumption of the network transmission. In addition, C_{memory} contains the number of memory bytes consumed by $Task_i$ it own, which is denoted as $C_{t.}$, the number of memory bytes consumed by storing local data, which is denoted as C_l, and the number of memory bytes consumed by storing network data, which is denoted as C_n. So C_{memroy} is derived as

$$C_{memory} = \frac{C_t + C_l + C_n - \alpha \times \log(L_l / C_l + C_n)}{T_{TotalTime}} \quad (6)$$

where L_l is the size of temporary data written to the local disk, because the task writes the memory data to the local disk when memory usage reaches a certain threshold, and *a is* regulator. When there is no data transmitted by network, the value of C_n is zero.

According to formula (5) and (6), C_c is equal to

$$C_c = C_{cpu} + \frac{C_t + C_l + C_n - \alpha \times \log(L_l / C_l + C_n)}{T_{TotalTime}} + C_{network} \quad (7)$$

where $T_{TotalTime}$ is the total running time of the job. When a running task T_r with the length L is completed, the resource consumption C_c of the task T_r is calculated. The real-time prediction mode will select an unexecuted task whose size is equal to L from the waiting list, and assign the task to the node where the task T_r located. If the size of task cannot be found that exactly matches the size, the closest match is used.

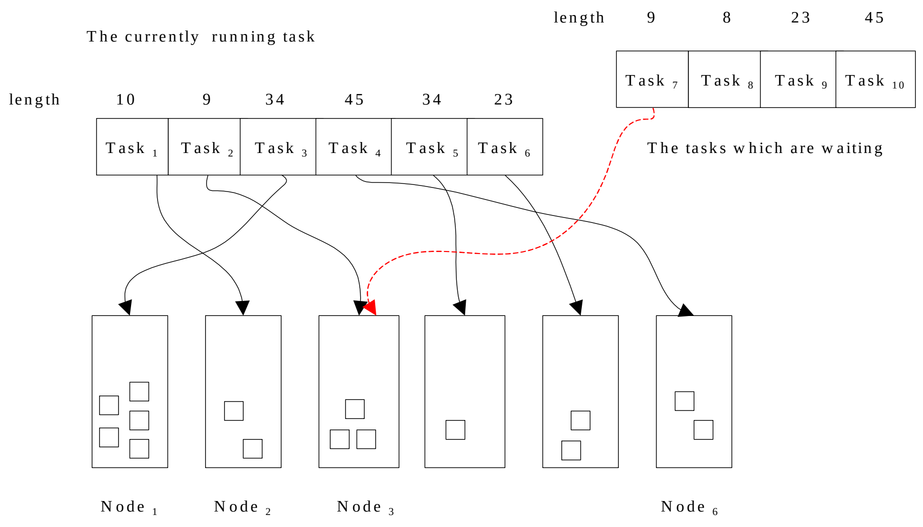


Figure 2. The Example of the Real-Time Prediction Mode

Figure 2 illustrates an example of the real-time prediction mode, support $Task_1$, $Task_2$, $Task_3$, $Task_4$, $Task_5$, and $Task_6$ are running, and $Task_7$, $Task_8$, $Task_9$, and $Task_{10}$ are waiting. As shown in Figure 2, $task_2$ is completed. The size of $task_2$ is 9. In waiting list, the size of $task_7$ is 9. So assigning $task_7$ to the $Node_3$ is the most efficient assignment method, because the resource including CPU, memory, and network on $Node_3$ can meet the requirements of the $task_7$.

4. Experimental Evaluations

4.1. Experimental Environment

In order to evaluate the effectiveness of the proposed job scheduling, the scheduling is compared with existing job scheduling algorithms, which are the First-in-first-out (FIFO) algorithm, and the fair scheduler. And to evaluate the performance of the proposed job scheduling, an experiment environment of a MapReduce cluster with hadoop 1.0.0 is established. The experiment cluster contains one *master* node and 9 *slave* nodes. These nodes are connected with a 100 Mb/s network. The *master* node is configured with 4-core 3.20 GHz Intel i7-960 processors, 16GB of memory and one 1TB 5400 RPM SATA disk. Each *slave* node is equipped with 4-core 3.10 GHz Intel i5-2400 processors, 16GB of memory and three 1TB 5400 RPM STAT disks. They all run CenterOS 6.4 with kernel 2.6.32-358.el6.x86_64 operating system. Each disk is formatted with the ext4 file system. The *master* node acts as *JobTracker*, *SecondaryNameNode*, and *NameNode*. Each *slave* node acts as *DataNode*, and *TaskTracker*.

WordCount [14] and *TestSort* [15] [16] benchmarks are performed in the experimental environment. The reason of selection the two benchmarks is that *WordCount* and *TestSort* program is often used as a baseline benchmark for MapReduce. The size of test data is classified into four types, which are 500MB, 1GB, 2GB, and 5GB. In order to make the test data stored into each *slave* node average, the block size of HDFS file system is set to the default value of 64MB, and the replication number of a block is set to the value of 3.

4.2. Experiment Result

Figure 3 and Figure 4 show the experiment results of the tree job scheduling algorithms in term of execution time. As shown in Figure 3 and Figure 4, the proposed job scheduling performs better than the FIFO and fair scheduler in term of execution time, because each task is assigned to the reasonable *slave* node through the dynamic priority scheduling and real-time prediction model. When the size of test data is 5GB, the advantage of the proposed scheduling is more apparent, because there are mount of waiting task in cluster. In this case, the proposed job scheduling can predict the resource usage of the waiting tasks, and assign a waiting job to the most suitable *slave* node. This assignment scheduler can use the cluster resources effectively. Moreover, the weight of a job can obviously reduce the data transmission of network, because a novel data locality is introduced. HIFO and fair scheduler spend a lot of time on the data transmission of network, which increases the total running time of the job.

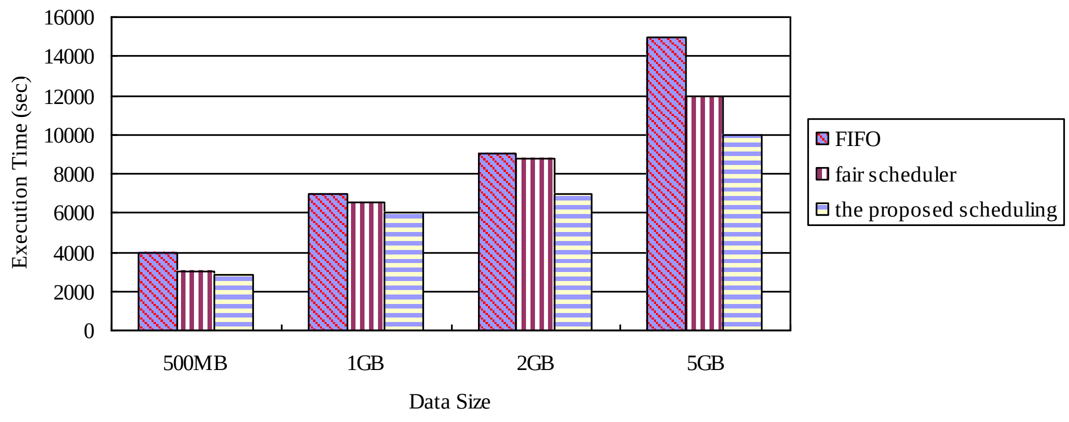


Figure 3. Word Count Job Execution Time

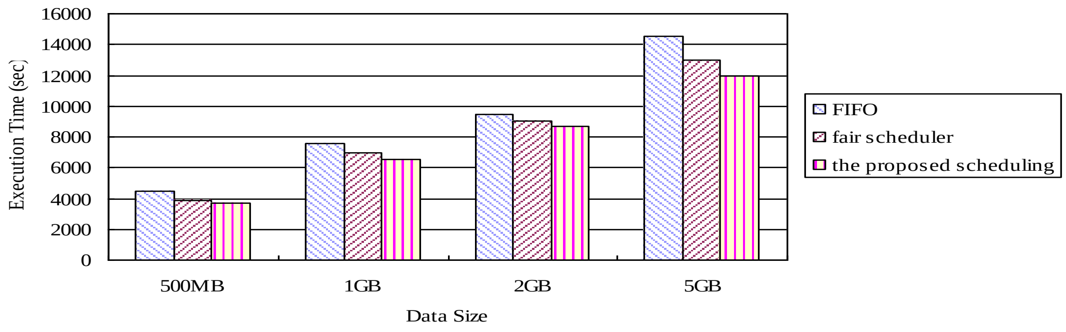


Figure 4. Test Sort Job Execution Time

From the Figure 3 and Figure 4, we can see that the execution time of the *TestSort* is larger than *WordCount*. The reason is that *TestSort* will carry shuffle data from one *slave* node to the other. The process will generate heavy disk I/O and network throughput. In addition, there is a mount of shuffle data in shuffle stage. In this case, a major bottleneck is network I/O bottlenecks. For *WordCount*, there are only small shuffle data in shuffle state.

5. Conclusion

This paper presents a novel job scheduling for MapReduce clusters. The objectives of the proposed scheduling are reducing the execution time of jobs, and taking full advantage of the resource of each node. The proposed job scheduling is advantageous in execution time and the resource utilization because it can calculate minimum cost and assign the job to appropriate nodes by minimum cost data locality algorithm. Moreover, the resource utilization of the unexecuted tasks can be concluded through the executed tasks. A series of experiments are conducted and encouraging results are obtained.

Acknowledgements

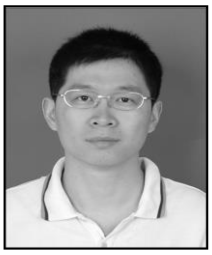
We are grateful to the editors and anonymous reviewers for their valuable comments on this paper. The work of this paper is supported by National Natural Science Foundation of China (Grant No. 61272399) and Research Fund for the Doctoral Program of Higher Education of China (Grant No. 20110191110038).

References

[1] J. Dean and S. Ghemawat, "Simplifying MapReduce data processing", Proceedings of the 4th IEEE International Conference on Utility and Cloud Computing", **(2011)** December, pp. 5-8, pp. 366-370, Melbourne, Australia.
[2] X. Kaiqi and H. Yuxiong, "Power-efficient resource allocation in MapReduce clusters", Proceedings of the IFIP/IEEE International Symposium on Integrated Network Management, (**2013)** May 27-31, pp. 603-608, Ghent, Belgium.
[3] Apache Software Foundation, Official Apache Hadoop Website. URL http://hadoop.apache.org/ Accessed date, **(2012)** July 1.
[4] Capacity Scheduler, Tech. rep., Retrieved, http://hadoop.apache.org/common/ docs/r0.20.2/capacity_scheduler.html, **(2012)** February.
[5] Fair Scheduler, Tech. rep., Retrieved: **(2012)** February. http://hadoop.apache.org/common/docs/r0.20.2/fair_scheduler.html
[6] M. Zaharia, D. Borthakur, J. S. Sarma, K. Elmeleegy, S. Shenker and I. Stoica, "Delay scheduling: a simple technique for achieving fairness," Proceedings of 16th Euro Sys Conference, **(2010)** March 1-5, pp. 265-278, Paris, France.
[7] S. Seo, I. Jang, K. Woo, I. Kim, J. S. Kim and S. Maeng, "HPMR: Perfecting and pre-shuffling in shared MapReduce computation environment," Proceedings of the IEEE International Conference on Cluster Computing and Workshops, **(2009)** August 31-September 4, New Orleans, United states.

[8] HDFS homepage. http://hadoop.apache.org/hdfs/
[9] S. Ghemawat, H. Gobioff, and S. Leung, “The Google File System,” Proceedings of the 19th ACM Symposium on Operating Systems Principles, vol. 37, no. 5, **(2003)** October 19-22, pp. 29-43. Lake George, United States.
[10] J. Yan, X. Yang, R. Gu, C. Yuan, and Y. Huang, "Performance Optimization for Short MapReduce Job Execution in Hadoop", Proceedings of the 2nd International Conference on Cloud and Green Computing and 2nd International Conference on Social Computing and Its Applications, Xiangtan, China, **(2012)** November 1-3, pp. 688-694.
[11] A. Bezerra, P. Hernández, and A. Espinosa, "Job Scheduling for Optimizing Data Locality in Hadoop Clusters", Proceedings of the 20th European MPI Users' Group Meeting, Madrid, Spain, **(2013)** September 15-18, pp. 271-276.
[12] J. S. Manjaly, V. S. Chooralil, "Task Tracker Aware Scheduling for Hadoop MapReduce", Proceedings of the 3th International Conference on Advances in Computing and Communications, **(2013)** August 29-31, pp. 278-281, Kochi, India.
[13] Hadoop homepage. http://hadoop.apache.org/.
[14] Word Count Program. Available in Hadoop source distribution: src/examples/org/apache/hadoop/examples/WordCount.
[15] Hadoop TeraSort program. Available in Hadoop source distribution since 0.19 version: src/examples/org/apache/hadoop/examples/terasort.
[16] TeraSort. http://sortbenchmark.org/.

Authors



Jun Liu, received his B.S. degree in Southwest University, P. R. China, at 2001, and M.S. degree in Chongqing University, P. R. China, at 2009. Currently he is a Ph.D. candidate in College of Computer Science, at Chongqing University. His current interests include big data analytics, flash memory, information security, and Linux Kernel.

ShuYu Chen, He received his Ph.D. degree in Chongqing University, P. R. China, at 2001. Currently, he is a professor of College of Software Engineering at Chongqing University. His research interests include embedded Linux system, distributed systems, cloud computing, etc. He has published over 120 journal and conference papers in related research areas during recent years.

Tianshu Wu, He received his B.S. degree in Chongqing University of Posts and Telecommunications, P. R. China, at 2011. He is currently a Ph.D. candidate in College of Computer Science, at Chongqing University. His current interests include cloud computing, large-scale data mining and fault detection.

MingWei Lin, received his B.S. degree in Chongqing University, P. R. China, at 2009. He is currently a Ph.D. candidate in Chongqing University. He is invited as the reviewer by *Journal of Systems and Software*, as well as *Computers and Electrical Engineering*. His current interests include large-scale data mining, flash memory, Linux Kernel, information security and wireless sensor network.