http://ansinet.com/itj

ITJ

ISSN 1812-5638

INFORMATION TECHNOLOGY JOURNAL

ANSInet
Asian Network for Scientific Information
308 Lasani Town, Sargodha Road, Faisalabad - Pakistan

Information Technology Journal 13 (3): 501-507, 2014
ISSN 1812-5638 / DOI: 10.3923/itj.2014.501.507

A Job Scheduling Policy based on the Job-classification and Dynamic Replica Mechanism

[1]Yan Qiang, [1]Yue Li, [2]Wei Wei, [1]Bo Pei, [1]Juanjuan Zhao and [1]Hui Zhang
[1]College of Computer Science and Technology, Taiyuan University of Technology, Taiyuan, 030024, China
[2]School of Computer Science and Engineering, Xi'an University of Technology, Xi'an, 710048, China

Abstract: In order to guarantee the stability of the system performance and the high Qos(quality of service) of users, a new method based on the HDFS (Hadoop Distributed File System) was proposed which including a job type classification method and a dynamic replica manage mechanism. The method uses the job type classification method to select the I/O intensive job, in order to achieve more accuracy of the classification taken the heterogeneity of the jobs into consideration. For the classified jobs, a dynamic replica manage mechanism was used to determine whether to increase or decrease the number of copies on the specific data node. For a test of a cluster with 1 namenode and 20 data nodes, the method has a high performance. The theoretical and experimental analyses in this paper prove that the proposed method has the ability to improve the performance of HDFS effectively.

Key words: Cloud computing, dynamic resources scheduling, replica placement policy

INTRODUCTION

Cluster computing systems, such as MapReduce, Hadoop and Dryad etc, have become a popular framework for data-intensive applications. Among various kinds of sizes in the cluster, throughput and job completion time are the main factors affecting the data-centers' costs and users' satisfaction. They were thought as the most important measures of the computation efficiency.

In recent years, many studies have been done on data locality. A distributed adaptive data replication algorithm DARE (Abad *et al*., 2011) that aids the scheduler to achieve better data locality was proposed. DARE improves data locality by more than 7 times with the FIFO scheduler in Hadoop and achieves more than 85% data locality for the FAIR scheduler with delay scheduling.

In order to improve the data locality of map tasks, a Next-k-node Scheduling (NKS) (Cheng *et al*., 2012) was proposed, implementing the NKS method in hadoop-0.20.2. Experiment results have shown that the NKS method reduced 78% of the map tasks processed without node locality; reduced 77% of the network load caused by the tasks and improved the performance of Hadoop MapReduce when comparing with the default task scheduling method in Hadoop.

When reassigning the slots, FAIR picks the most recently launched tasks to kill without considering the job character and data locality which increases the network traffic while rescheduling the killed Map/Reduce tasks. Focused on this problem an improved FAIR scheduling algorithm (Tao *et al*., 2011) was proposed, which take into account the job character and data locality while killing tasks to make slots for new users. In order to improve the scheduling efficiency, the dynamic processing slots scheduling (Kurazumi *et al*., 2012) was used for the I/O intensive jobs of Hadoop MapReduce focusing on I/O wait during execution of jobs.

An elastic replication management system for HDFS, which provides an active/standby storage model for HDFS (Cheng *et al*., 2012), was proposed. It utilizes a complex event processing engine to distinguish real-time data types and then dynamically increases extra replicas for hot data, cleans up these extra replicas when the data cool down and uses erasure codes for cold data.

A method of energy-conserving, hybrid, logical multi-zoned (hot zone and cold zone) variant of HDFS (Kaushik and Bhandarkar, 2010) was proposed to manage data-processing intensive, commodity Hadoop cluster. Analysis of the traces of a Yahoo! Hadoop cluster showed significant heterogeneity in the data's access patterns which can be used to guide energy-aware data placement policies. The trace-driven simulation results with three-month-long real-life HDFS traces from a Hadoop cluster at Yahoo! show a 26% energy consumption reduction by doing only Cold zone power management.

Corresponding Author: Hui Zhang, College of Computer Science and Technology, Taiyuan University of Technology, Taiyuan, 030024, China Tel: 18735128030

It is a common practice for data-intensive systems to place the data as close as possible when calculating, which referred to as the data locality problem. Due to the significant impaction on system throughput and job completion time, the data locality problem has become an important problem in data-intensive systems. HDFS adopts a uniform triplication policy (i.e., three replicas for each file) to improve the data locality. The policy can also ensure data availability and fault tolerance in the event of data and disk failures. This policy could also achieve load balancing by distributing work across the replicas. Though high reliability and performance the policy has in most cases, there are two significant problems.

First, in a large and busy HDFS cluster, the hot data (Chen *et al*., 2010) could be requested by many distributed clients concurrently. Replicating the hot datas only on three different nodes is not enough to avoid contention of the data nodes which store hotdatas. If the number of jobs which concurrently access to hot data exceeds that of replicas, some of these jobs may have to access to the data remotely and compete for the same replica. Second, the triplication policy comes with a high over head cost in the management for the cold data. Too many replicas may not improve availability significantly, but bring unnecessary expenditure instead. The management cost, including storage and network band width, will significantly increase with the increasing number of replica.

Therefore, the hot data should be assigned with a larger number of replicas to improve data locality of concurrent accesses. The additional copies may be used not only to not only improve availability, but also to provide load balancing and improve overall performance if replicas and data accessing requests are reasonably distributed. As a result, dynamic configuration of file replica is necessary (Berral *et al*., 2011).

METHODS

Here, we will give a generalized introduction of a method of job type classification and after that a dynamic replica policy was put forward.

Job type classification: Since the different types of job have different amount of requests on HDFS, the heterogeneity of a job type should be taken into consideration. In this part, we classify jobs into cluster according to the following method.

The execution time of a map task is the total time spent on fetching the output of the map tasks and writing results to HDFS. In the following we decouple the time consumption on I/O operations and CPU operations of a MapReduce application. The execution time for a map/reduce task is then defined as:

$$\text{Task ExecutionTime} = OT + CT + IOT \quad (1)$$

where, OT is the fixed overhead in running a task, CT and IOT are times taken in CPU and IO operations, respectively. OT is independent of data size which mainly includes JVM (Java virtual machine) initialization overhead and scheduling time. CPU-related operations mostly occur in the user-defined map and reduce function. Broadly, IO operations can be classified as follows:

- Input and output for a map/reduce task
- Reading and writing for sorting data in a map/reduce task
- Shuffle for a reduce task

CT and IOT are two parts, distinguishing from other types of task, to represent the characteristic of a task. The ratio between them is denoted by Computing Rate (CR) and I/O rate, respectively. The I/O rate of a task is the total amount of input and output of a task divided by task execution time. Since, the Hadoop framework uses cache mechanism and temporary files for sorting, the accurate total amount of input and output of a task is difficult to be counted. Thus, in this study, we adopt CR to represent the characteristic of a task which is defined as:

$$CR = CT / \text{Task ExecutionTime} = CT / (OT + CT + IOT) \quad (2)$$

If a task's CR reaches to 1, the task is regarded CPU intensive, or I/O intensive if CR is close to 0.

The LPL was validated by running 19 bench marks, including machine learning jobs (Lu *et al*., 2012), web search jobs and some typical MapReduce benchmark jobs etc. According to their results, an I/O intensive job can be defined as both of its map CR and reduce CR is less than 0.2. Now, reduce tasks in 16 jobs of 17 benchmarks (94%) are able to be reflected correctly.

Dynamic replica policy: In this part, we present a dynamic replica management based on HDFS, including a dynamic adjustment and selection mechanism of replica. After that we implement a experiment on the method replica controller based on HDFS. The overall architecture of the system is shown in Fig. 1.

The following presents the main module and function of the replica controller:

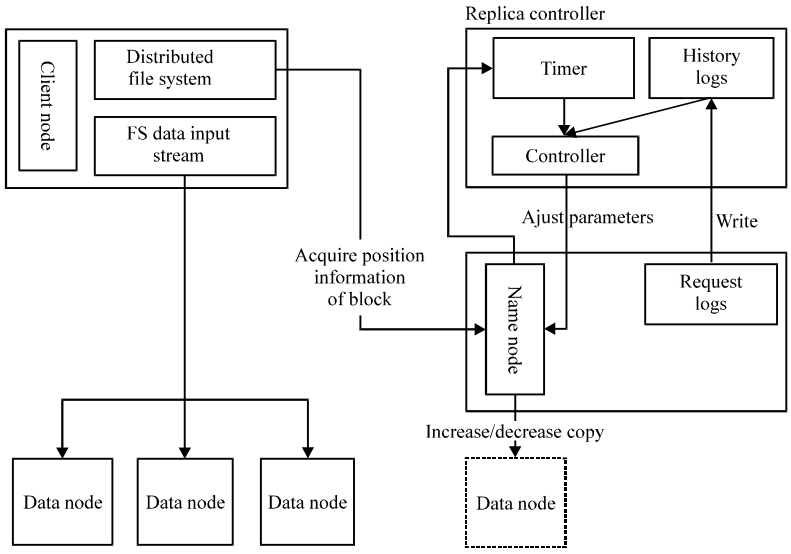

Fig. 1: System architecture

- **History Logs:** Once name node receives file read requests from clients, the request logs will be first stored in buffer first, then be sent to the "history Logs" of replica system in every t sec
- **Timer:** The main function is to prevent the situation that a file read request from the client surge in a short time and then a sharp decline, leading to corresponding changes of the number of copies in a short period of time. These changes may increase the burden on system. Its cycle equals β_0
- **Controller:** It has two main functions. The first is to modify the BlockMap in Namenode to adjust numbers of replica. The second is to select the suitable nodes by analysis of history logs information, then add or delete replica to this selected nodes

Dynamic adjustment mechanism of replica: In order to eliminate the performance bottleneck of HDFS and achieve dynamic adjustment of replica. We made some small changes on the basis of the original HDFS. Let variable which represents the number of replica in fsimage as the min replica number (minReplica) and add two variables numReplica and connectCounter to the attribution of HDFS. NumReplica represents current numbers of replica, its initial value equals minReplica. Replica system will adjust the number of replica dynamically according to value of numReplica. ConnectCounter represents the number of users that read the same file in the same time.

Pseudocode of file read and dynamic adjustment process is shown as Table 1.

In this pseudocode, $\rho_{max} = 0.8 \times L_{max} \times$num Replica, $\rho_{min} = st_0 L_{max}$ is the max link number of each replica and st represents the standard replica number in fsimage.

Dynamic selection mechanism of replica: Here, a copy selection method based on the stage of history information is proposed. This method will describe that which data node should be assigned.

In general, an arbitrary undirected or bidirectional graph G(V, E) is taken into consideration. V is node set $V_1 \cap V_2 \ldots V_h \ldots \cap V_k = \Phi$, V_i is sub set of V and represents data node in different region. E is link set between two nodes and $E \subset V \times V$. The link between any two nodes n_i and n_j is given a weight value of d(i, j), it is the delay between them. In this mechanism, assign value X to each node X_i, represent whether the replica of file X is stored on node H_i. If $X_i = 1$, indicates that there are copies of the file on the node n_i. V'_h represents node set which has copies of file in region V_h.

Select node which needs to be added to file replica as follows:

First, in order to determine adding file replica in which domain, replica controller will divide records into $R = r_1 \cup r_2 \cup \ldots \cup r_h \ldots r_k$ according to history information, r_k represents number of requests sent by client in region V_k.

Second, replica controller will select region $\bar{V}$ of node that need to add replica:

$$\bar{V} = \min(r_h / V_h) \quad (3)$$

After determine the region, replica controller need to determine which node is in this selected region. Since

Table 1: Pseudocode of file read and dynamic adjustment process

Algorithm 1:	
Steps	Description
1	Begin:
2	numReplica = minReplica;
3	Read requests of file from clients;
4	connectCounter++;//Namenode
5	If (counectCounter>upper threshold ρ_{max}){
6	Send message "addInfo";//From Namenode to System controller
7	Start the timer;
8	Ack;//From Controller to Namenode
9	Redefine the value of ρ_{max};//Namenode
10	Assign freedatanode and add new file replica on this datanode;//controller
11	numReplica++;}
12	Read data;
13	Send request logs to controller;
14	connectCounter--;
15	if (counectCounter<lower threshold ρ_{min}){
16	Send message "addInfo";//From Namenode to System controller
17	Start the timer;
18	Ack;//From Controller to Namenode
19	Redefine the value of ρ_{min};//Namenode
20	Assign datanode and delete file replica on this datanode;//controller
21	numReplica--;}
22	End

replica controller has history information, it can record the location of users who send request as V_r, $V_r = V_{r1} \cup V_{r2} \cup \ldots V_{rh} \ldots \cup V_{rk}$, V_{rh} is node set of all the requests in V_h. $G_{rh}(V_{rh}, E)$ is graph constituted by V_{rh} and link between nodes. And distribution of all copies in HDFS file system can be informed by reading the name of the node in the BlockMap. By doing this, we can get V_R, $V_R = V_{R1} \cup V_{R2} \cup V_{R3} \ldots \cup V_{Rh} \ldots V_{Rk}$, V_{Rh}, V_R, is set of nodes which own copies of file. $G_{Rh}(V_{Rh}, E)$ is the coverage of V_{Rh} In summary:

$$\bar{G}' = G_{rh} - (G_{rh} \cap G_{Rh}) \quad (4)$$

The selection process of deleting file replica is similar to that of adding nodes. What need to do is change the value of $\bar{V}$ to $\max(r_h/V_h)$.

EXPERIMENTS

We evaluated the proposed method of job classification and dynamic replica management in a private cluster with 1namenode and 20 data nodes. The name nodes are run on the server with Ubuntu 10.0.4, two Intel Xeon E5520 CPUs, 2.26 GHz, 12 GB memory and 2TB SATA disk. The data nodes are run on a personal computer with Ubuntu 10.0.4, Intel Xeon E5420 CPU, 2.50 GHz, 8 GB memory and 300 GB SATA disk. The Java version is 1.6.0_24. These nodes locate in three different racks with Gigabit Ethernet network connecting.

Experiments of job type classification: The aim of the first experiment is to evaluate our analysis on generalization of workload characteristics. The input data is 512 MB for each job. Since the default size of block is 64 MB, the size of input data leads eight map tasks for each job and we manually set eight reduce tasks.

We have verified the similarity in the characteristic of tasks in an individual benchmark. For simplicity, 5 representative applications among 19 benchmarks is presented. Those 5 benchmarks are, respectively Word Count, Grep, Sort, Terasort and Crypto. The selection is based on their relativeness with CR. Namely while the latter three can represent I/O intensive, the first two are relatively more CPU intensive. We separately ran each of these 5 benchmarks in isolation to identify the computing rate and the execution time of each (map/reduce) task. The result shows that there is no significant variation for tasks belonging to the same job in terms of both CR and execution time. The variation of reduce tasks tends to be greater than that of map tasks, which is because that the input data of reduce tasks and outputted by map tasks are probably not partitioned evenly. However, the variation of execution time for map tasks and reduce tasks in the same job still remains in a very similar level. Therefore, we can use the characteristic of a task to represent the characteristic of the rest of tasks.

According to the research of Hadoop benchmark provided by Intel, some characteristics of benchmark program workload are shown in Table 2.

From the comparison of Fig. 2 and Table 1, the proposed method according to computing rate can be used for job classification. The results are basically in line with Intel's research.

Experiments of dynamic replica management: TeraSort is a standard map/reduce sort, except for a custom

Table 2: Characteristics of benchmark program workload

Workload	Resource characteristics
Word count	CPU intensive, light workload of network and I/O
Sort	I/O intensive, medium utilization of CPU
TreaSort	CPU intensive in map and shuffle, medium I/O I/O intensive in reduce, medium utilization of CPU
Nutch Indexing	CPU intensive in map I/O intensive in reduce, medium utilization of CPU
Page rank	Basically CPU intensive
Bayesian classification	Four jobs are basically I/O intensive, CPU utilization of the first job is high
K-means clustering	Computing of center node: CPU intensive Clustering: I/O intensive
Enhanced DFSIO	I/O intensive

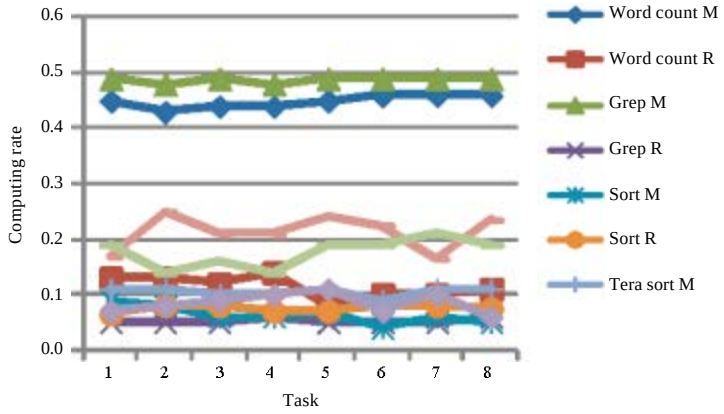

Fig. 2: Characteristic analysis for 6 typical benchmarks

partitioner that ensures that all of the keys in reduce N are after all of the keys in reduce. This is a requirement of the contest so that the output of the sort is totally ordered; even if it is divided up by reduce. Experimental data is generated by TeraGen. It generates input data for the sort that is byte for byte equivalent to the C version that was released in March of 2009, including specific keys and values. It divides the desired number of rows by the desired number of tasks and assigns ranges of rows to each map. The map jumps the random number generator to the correct value for the first row and generates the following rows.

TeraSort uses Hadoop's MapReduce mechanism to achieve Sort. With the perfect combination of Hadoop mechanism, TeraSort has been a superior sort program. So it can be used for testing Hadoop in cluster due to its high test value. From experiment 1, TeraSort can be classified as I/O intensive job in reduce phase. The amount of experimental data is 10 GB. The amount of map task can be adjusted by "mapred.min.split.size" and the amount of reduce task can be adjusted by "mapred.reduce.tasks". To simulate reality in different time periods of different users simultaneously access to the same data, the number of map task and reduce task can be adjusted.

First, we maintain the amount of map task equals 16 and change the amount of reduce task. Moreover, make a comparison of execution time between HDFS default copy strategy and the dynamic replica adjustment method. The execution time of map and reduce under HDFS default copy strategy and under the dynamic replica adjustment method are shown in Fig. 3 and 4, respectively.

From the analysis of Fig. 3 and 4, we can draw such conclude that: when the number of reduce task is less than 15, the total time and reduce time are inversely proportional to the amount of reduce task. When the number of reduce task is greater than 15, total time and reduce time remain substantially constant. The amount of reduce task should be close to that of slave nodes and slightly larger than that of nodes. There is no apparent relationship between execution time of Map and the amounts of reduce task.

Compared to HDFS original copy strategy, dynamic replica policy has a significant performance improvement, in particular when reduce task number is greater than 15. But there is no significant decline of execution time when rapid growth of the number of users occurs. The most suitable number of reduce task should be 0.95 or 1.75×(number of datanode×mapred.tasktracker.tasks.maximum).

If the number of tasks is 0.95 times as many as that of nodes all reduce tasks can be launched at the same time after the end of the map task output transmission. If the number of tasks is 1.75 times as many as that of nodes,

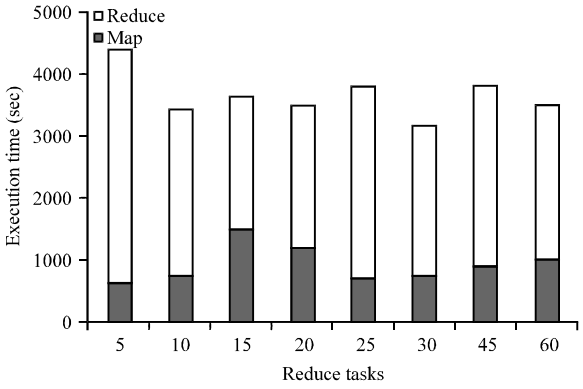

Fig. 3: Execution time of map and reduce under HDFS default copy strategy

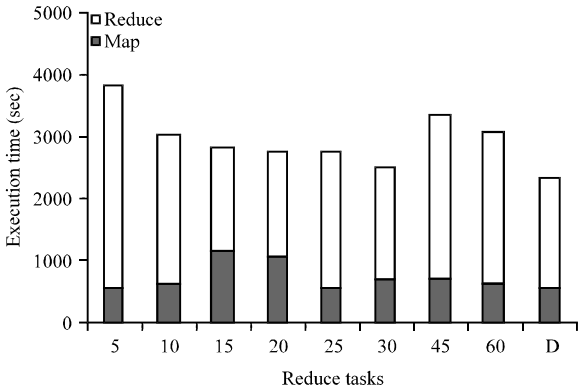

Fig. 4: Execution time of map and reduce under the dynamic replica adjustment method

then fast node will launch the second batch reduce tasks after finishing of the first batch reduce tasks, it is more favorable load balancing. As can be seen from Fig. 3 and 4, the proposed method works well when there are 15 (nearly 0.95 times), 25 and 30 (nearly 1.75 times), program's execution time has an apparently decline. In other words, the performance of the Hadoop can be significantly improved with reasonable adjustment of the number of Job Tasks.

Moreover, a dynamic parameter adjustment method was achieved by script. In the initial stage, set 5 reduce tasks, set 20 reduce tasks from time t1, set 40 reduce tasks from time t2, set 60 reduce tasks from time t3, generate a random number between 10-60 from time t4. The contrast of the proposed dynamic replica adjustment method and HDFS default replica policy is shown in Fig. 4, last row.

CONCLUSION AND FUTURE WORK

The replica technique is widely used in the storage system, which can ensure system stability, enhance the rate of user access to files, as well as improve the security of data and other characteristics. HDFS, which uses the form of a copy of the file block to store larger files, is a common distributed file system. It has a default replica mechanism, yet is static. When many users request for the same particular file at the same time, this file will be the so-called hot file, as a result, system performance will have a sudden drop. Users' QoS will have a corresponding decline. Therefore, it is necessary to use a dynamic replica scheduling mechanism to guarantee system performance and Users' QoS. In this paper, a job type classification method and a dynamic replica manage mechanism based on HDFS are proposed. I/O intensive jobs will be picked up first and then uses the dynamic replica manage mechanism to determine whether to increase or decrease the number of copies on appropriate datanode. Experiment result shows that the proposed method can improve HDFS performance effectively.

There are still some limitations in this proposed method. Its basic idea is sacrifice storage space to reduce users' access delay time. However, this mechanism will not be activated until large amounts of file requests occur. And the number of file replica will have a corresponding decline when the number of user request decreases. Furthermore, the prices of the storage resources are relative lower in the real cloud environment. Therefore, the proposed method is acceptable in real cloud environment.

ACKNOWLEDGMENTS

This study was supported by the National Natural Science Foundation of China (Grant No. 61202163, 61240035); Natural Science Foundation of Shanxi Province (Grant No. 2012011015-1). This work was also supported in part by the US National Science Foundation (NSF) under Grant No. CNS-1016320 and CCF-0829993. This work is also supported by Scientific Research Program Funded by Shaanxi Provincial Education Department (Program No.2013JK1139).

REFERENCES

Abad, C.L., Y. Lu and R.H. Campbell, 2011. DARE: Adaptive data replication for efficient cluster scheduling. Proceedings of the IEEE International Conference on Cluster Computing, September 26-30, 2011, Austin, TX., USA., pp: 159-168.

Berral, J.L., R. Gavalda and J. Torres, 2011. Adaptive scheduling on power-aware managed data-centers using machine learning. Proceedings of the 12th International Conference on Grid Computing, September 21-23, 2011, Lyon, pp: 66-73.

Chen, Q., D. Zhang, M. Guo, Q. Deng and S. Guo, 2010. SAMR: A self-adaptive mapreduce scheduling algorithm in heterogeneous environment. Proceedings of the IEEE 10th International Conference on Computer and Information Technology, June 29-July 1, 2010, Bradford, UK., pp: 2736-2743.

Cheng, Z., Z. Luan, Y. Meng, Y. Xu and D. Qian *et al*., 2012. ERMS: An elastic replication management system for HDFS. Proceedings of the International Conference on Cluster Computing Workshops, September 24-28, 2012, Beijing, China, pp: 32-40.

Kaushik, R.T. and M. Bhandarkar, 2010. Greenhdfs: Towards an energy-conserving, storage-efficient, hybrid hadoop compute cluster. Proceedings of the USENIX Annual Technical Conference, June 23-25, 2010, Boston, MA., USA., pp: 1-5.

Kurazumi, S., T. Tsumura, S. Saito and H. Matsuo, 2012. Dynamic processing slots scheduling for I/O intensive jobs of Hadoop MapReduce. Proceedings of the 3rd International Conference on Networking and Computing, December 5-7, 2012, Okinawa, Japan, pp: 288-292.

Lu, P., Y.C. Lee, C. Wang, B.B. Zhou, J. Chen and A.Y. Zomaya, 2012. Workload characteristic oriented scheduler for mapreduce. Proceedings of the IEEE 18th International Conference on Parallel and Distributed Systems, December 17-19, 2012, Singapore, pp: 156-163.

Tao, Y., Q. Zhang, L. Shi and P. Chen, 2011. Job scheduling optimization for multi-user mapreduce clusters. Proceedings of the 4th International Symposium on Parallel Architectures, Algorithms and Programming, December 9-11, 2011, Tianjin, China, pp: 213-217.