MTSD: A task scheduling algorithm for MapReduce base on deadline constraints

Zhuo Tang[1], Junqing Zhou[1], Kenli Li[1], Ruixuan Li[2]
[1]*School of Information Science and Engineering, Hunan University, Changsha 410082,Hunan, China.*
[2]*School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, Hubei, China.*
hust_tz@126.com, zjq@hnu.edu.cn, jt_lkl@hnu.cn, rxli@hust.edu.cn

***Abstract*—The previous works about MapReduce task scheduling with deadline constraints neither take the diffenences of Map and Reduce task, nor the cluster's heterogeneity into account. This paper proposes an extensional MapReduce Task Scheduling algorithm for Deadline constraints in Hadoop platform: MTSD. It allows user specify a job's deadline and tries to make the job be finished before the deadline. Through measuring the node's computing capacity, a node classification algorithm is proposed in MTSD. This algorithm classifies the nodes into several levels in heterogeneous clusters. Under this algorithm, we firstly illuminate a novel data distribution model which distributes data according to the node's capacity level respectively. The experiments show that the data locality is improved about 57%. Secondly, we calculate the task's average completion time which is based on the node level. It improves the precision of task's remaining time evaluation. Finally, MTSD provides a mechanism to decide which job's task should be scheduled by calculating the Map and Reduce task slot requirements.**

Keywords: MapReduce, scheduling algorithm, data locality, deadline constraints, Hadoop.

I. INTRODUCTION

Nowadays the requirements for massive data processing are increasing, such as machine learning [1], scientific analysis [2], astrophysics [3] and bioinformatics [4]. MapReduce [5] is a distributed programming model for expressing distributed computation on massive amounts of data and an execution framework for large-scale data processing on clusters of commodity servers. One of the most important advantages of MapReduce is its convenience. Programmer can process massive data without knowing the details of distributed implementation. User can process large scale of data by only providing the Map and Reduce interface.

The initial MapReduce model was designed for off-line data processing. However, it is now popularly applied in heterogeneous, sharing and multi-user environments. Nowadays, the MapReduce scheduling algorithms mainly include FIFO, LATE [6], FairScheduler [7] and CapacityScheduler [8]. The features about data locality, user priority, fault-tolerant and fairness are all considered by these algorithms. However, few algorithms have considered the user's job deadline constraints, such as in the elastic cloud computing environment.

The works in [9, 10] propose the solutions for MapReduce job scheduling on deadline constraints problem. In [9], the author builds a task execution model by evaluating the tasks' data processing unit time and data transferring unit time. And it uses the model to calculate that how many Map and Reduce tasks should be scheduled in order to satisfy the deadline constraints. However, the scheduling algorithm is based on homogeneous cluster. Polo et al. [10] propose a task scheduler to predict the performance of concurrent MapReduce jobs dynamically and adjust the resource allocation of the jobs. Nevertheless, it fails to consider the differences between the Map and Reduce tasks.

The data locality in MapReduce means that a Map task is executed on the node which contains its input data. Literatures [11, 12, 13] work on improving MapReduce's

978-0-7695-4676-6/12 $26.00 © 2012 IEEE
DOI 10.1109/IPDPSW.2012.250

data locality. In [11], the author improves data locality through building a relationship between application and nodes to place data reasonable. Matei Zaharia et al. [12] propose the delay scheduling algorithm to address the conflict between locality and fairness. NKS algorithm [13] is proposed to improve the data locality of Map tasks, and it is based on homogeneous environment.

To meet the users' job deadline requirement in the cloud environment we present the MTSD algorithm. The MTSD algorithm takes the data locality and cluster heterogeneity into account. The contributions of this paper are as follows: (1) A node classification algorithm is proposed to improve the Map task's data locality. (2) We present a novel remaining task execution time model which is based on node classification algorithm.

II. RELATED WORK

At present, the works about MapReduce scheduling algorithms focus on data locality, sharing, fairness and fault-tolerant ability. Dynamic Proportional Scheduler [14] 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. Data locality is a key performance factor of task's completion time in Hadoop. Matei Zaharia et al. propose the delay scheduling algorithm [12] to address the conflict between data locality and fairness. However, the method takes fairness withered as the cost and it isn't fit for the jobs which have large size or few slots per node. In [11], the authors improve data locality through building a relationship between application and nodes to place data reasonable. Xiaohong Zhang et al. propose the NKS algorithm [13] to improve the data locality of map tasks. However, it is based on the homogeneous environment.

There are some works on MapReduce job scheduling algorithm take deadline constraints into account. Time estimation and optimization for Hadoop jobs has been explored by [15, 16]. In [15], the authors focus on minimizing the total completion time of a set of MapReduce jobs. In [16], it estimates the progress of queries that run as MapReduce DAGs. Kamal Kc et al. [9] propose a task execution time model by evaluating the tasks' data processing unit time and data transferring unit time, and then it uses the model to compute how many Map and Reduce tasks should be scheduled to satisfy the deadline constraints. But the scheduling algorithm is base on homogeneous cluster. Polo et al. propose a task scheduler to predict the performance of concurrent MapReduce jobs dynamically and adjust resource allocation for the jobs [10]. However, it didn't consider the differences between the Map and Reduce tasks. Both of the Map and Reduce task's execution time are uncorrelated, so it's not accurate to compute average task execution time by taking Map and Reduce tasks together.

In this paper, we propose MTSD algorithm for job's deadline constraints; meanwhile, it improves the map task's data locality and has a more accurate task time evaluation method by nodes classification algorithm. The rest of this paper is organized as follows: In section III, we propose the MapReduce Task Scheduler for Deadline (MTSD) algorithm to meet the deadline constraints, and node classification algorithm to improve the Map task's locality. In section IV, we show the experiment results to evaluate the performance of our algorithm. We conclude the paper in section V.

III. MTSD: DEADLINE CONSTRAINTS BASED MAPREDUCE SCHEDULING ALGORITHM

A. MapReduce computing framework and data locality

As a distributed computing framework on commercial computer, one of the MapReduce's most significant advantages is that it provides an abstraction that hides many system-level details from programmer. It processes data by dividing the progress into two phases: Map and Reduce. Figure 1 shows that the Map function is applied to each input key-value pair and generates an arbitrary number of intermediate key-value pairs. The Reduce function is applied to all values that associated with the same intermediate key and generates output key-value pairs as the final result. Hadoop is an open source implementation of MapReduce.

In the MapReduce framework, Map or Reduce codes can be moved among the cluster nodes and the data can be transferred from a node to another. If the code and data on the same node, we call this data locality. The cost of

migrating code is extremely lower than migrating data. So the ideal situation is moving the code, not data. For the sake of this purpose, it needs a reasonable data distribution strategy. The default data distribution strategy in Hadoop is random. In this paper, we show a new data distribution model to improve data locality.

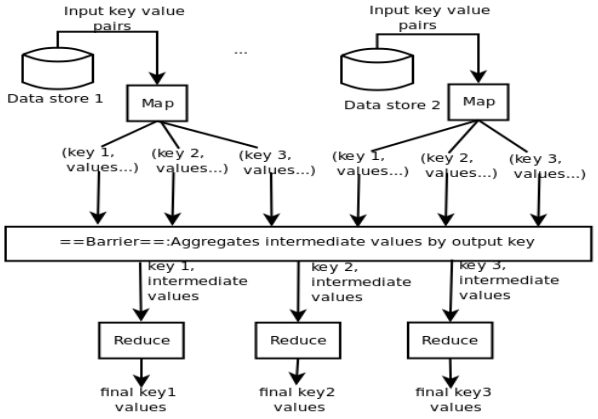


Figure 1. The MapReduce computing framework

B. *Node classification algorithm*

In a heterogeneous environment, cluster usually contains nodes with different computing capacity. It means that the speed of the node processes data. We classify nodes according to their computing capacity. There are two purposes of node classification with their computing capacity: one is to optimize the data distribution in order to improve the data locality; the other is to improve the evaluation accuracy of the task remaining time in heterogeneous environment.

Assure that there are K levels of nodes in cluster, L_p ($1 \le p \le K$) means the level factor of the p-th level, and let L_p as:

$$L_p = \begin{cases} L_{p-1} * \delta & 1<p\le K \\ 1 & p=1 \end{cases} \quad (1)$$

Where δ is a constant number, and named classification factor.

We quantize the node's computing capacity simply by running a group of specific tasks. On a given cluster, we set each node with one Map slot and one Reduce slot. And all the map tasks are feed by the same input data. Then the benchmark jobs run on the cluster. We record the completion time of each task on where it ran. We set δ as a constant value larger than 1. Pseudo code for node classification algorithm is shown in algorithm 1.

Algorithm 1 node classification algorithm

```
N={1,2, …,n};              //node collection
CLASS[] = NULL;            //level collection
level = 1;                 //the level indicator
for i from 1 to n
      Compute_task_on(node);
      T[i].time = get_execute_time(node);
      T[i].node = i ;
end for
Sort T[] by time in descending order;
node = T[1].node ;
CLASS[level] ← CLASS[level] U node;
for j from 2 to T.length-1
      if get_execute_time(node)/T[j].time > δ
         then     // larger than δ , set T[j].node to next level
            node = T[j].node
            level++
            CLASS[level] ← CLASS[level] U T[j].node
      else      // set T[j].node to current level
            CLASS[level] ← CLASS[level] U T[j].node
       end if
end for
return CLASS
```

Note that different benchmark jobs will make a different classification result. For example, a node may be in level i on job1 while be in level j on job2. To resolve this problem, we test a series of benchmark jobs and use a statistics method to decide which level a node belongs to.

The node classification algorithm enables us to get the nodes' level by their computing capacity. The data distribution strategy is that the size of each node's data is in proportion to the node's level. For example, there are three levels of cluster nodes L1, L2 and L3; and the size of data is 60G. Their computation ability is 1, 2, 3 grade, so the data distribution on nodes is 10G, 20G and 30G respectively. The data in the same type is distributed by random distribution principle. In the same class of nodes, it selects a node to store data randomly. This data distribution strategy can avoid the migrating of data from low level nodes to high level nodes.

C. *The remaining task execution time model*

In order to finish task before the deadline, we need to build an accurate remaining time model. Evaluation the

remaining time of a task is mainly on the average competition time that the same type of tasks on some type of nodes.

Definition 1: J = (M,R,A,D) means a MapReduce job. M, R, A and D denotes Map task set, Reduce task set, job arrived time, and job deadline constraints respectively.

CM(J,p) denotes the completed job J's Map task set which run on the p-th level; and CR(J,p) means the already completed Reduce task set which ran on the p-th level; TM_m denotes the task M_m's ($M_m \in CM(J,p)$) completion time; TR_r denotes the task R_r's ($R_r \in CR(J,p)$) completion time. So,

$$\text{Average\_TM}(J,p) = \frac{\sum_{M_m \in CM(J,p)} TM_m}{|CM(J,p)|} \quad (1)$$

is the average completion time of job J's Map tasks, which run on the p-th level. And

$$\text{Average\_TR}(J,p) = \frac{\sum_{R_r \in CR(J,p)} TR_r}{|CR(J,p)|} \quad (2)$$

is the average completion time of job J's Map tasks', which run on the p-th level.

We use UM(J) and UR(J) to denote the job J's Map and Reduce task's waiting set respectively. $MM_m(J,p)$ denotes the completion time of job J's Map task which runs on the node in p-th level. So the completion time of a running task equal to the running time that have spent and the remaining execution time. That is:

$$MM_m(J,p) = RTM_m + CTM_m \quad (3)$$

Where CTM_m means the m-th Map task's running time that have spent, RTM_m means the m-th Map task's remaining execution time. So the m-th Map task's remaining execution time of job J is:

$$RTM_m = MM_m(J,p) - CTM_m \quad (4)$$

The completion time of a certain job's tasks, which run on the nodes belonging to the same capacity level, will tend to be the same. We know that MMm(J,p) is approximately equal to Average_TM(J,p). So equation (2) can be revised as:

$$RTM_m = \text{Average\_TM}(J,p) - CTM_m \quad (5)$$

In the same way, we can get the running Reduce task's remaining execution time:

$$RTR_r = \text{Average\_TR}(J,p) - CTR_r \quad (6)$$

where CTR_r denotes the running time that have spent.

Now we can calculate the sum of the remaining Map and Reduce task's execution time of job J which running on the p-th level nodes:

$$SM(J,p) = \sum_{1 \le i \le m} RTM_m \quad (7)$$

$$SR(J,p) = \sum_{1 \le i \le r} RTR_r \quad (8)$$

D. Deadline constraints based task scheduling algorithm

1) Map and Reduce's two stage scheduling

In MapReduce, the job's execution progress includes Map and Reduce stage. So, the job's completion time contains Map execution time and Reduce execution time.

In view of the differences between Map and Reduce's code, we divide the scheduling progress into two stages, namely Map stage and Reduce stage. Previous researches usually simplify the Map and Reduce as the same type of scheduling problem. It may do simplify the problem, but it is improper for the reason that the execution of Map and Reduce's code is different. In the aspect of the task's scheduling time prediction, the execution time of Map and Reduce is not correlative; their execution time depends on the input data and function of their own. Therefore, in this paper the scheduling algorithm sets two deadlines: map-deadline and reduce-deadline. And reduce-deadline is just the users' job deadline.

In order to get map-deadline, we need to know the Map task's time proportion on the task's execution time. In a cluster with limited resources, Map slot and Reduce slot number is decided. For an arbitrary submitted job with deadline constraints, the scheduler has to schedule reasonable with the remaining resources in order to assure that all jobs can be finished before the deadline constraints.

Definition 2: P_m and P_r are the proportion of Map and Reduce task's execution time respectively, and $P_m + P_r = 1$.

This paper uses empirical data to quantize the value of Pm and Pr. We use a data sample from user's input data and run the code on the data, and then we get the map and reduce task's execution time Tm and Tr. So:

$$P_m = \frac{T_m}{T_m + T_r} \quad (9)$$

and

$$P_r = \frac{T_r}{T_m + T_r} \tag{10}$$

Now we can calculate the map-deadline by Pm:

$$D_m = A + (D - A) * P_m \tag{11}$$

where A and D means the job's arrived time and deadline respectively.

According to map-deadline, we can acquire the current map task's slot number it needs; and with reduce-deadline, we can get the current reduce task's slot number it needs.

We estimate the time needed for the remaining task on the lowest level node. By this way, we can find the emergency degree of current job and the minimum Map slot number of job J needs:

$$\delta_J^m = \frac{\sum_{1 \le p \le K} SM(J,p) * L_p^{\ p} + UM(J) * MM(J,1)}{|D_m - CurrentTime|} \tag{12}$$

Similarly, the minimum Reduce slot number required of job J is:

$$\delta_J^r = \frac{\sum_{1 \le p \le K} SR(J,p) * L_p^{\ p} + UR(J) * MR(J,1)}{|D - CurrentTime|} \tag{13}$$

2) Deadline constraints based task scheduling algorithm

The scheduling strategy of MSTD is based on δ_J^m and δ_J^r. The minimum Map and Reduce slot number required of job J can be denoted as δ_J^m and δ_J^r respectively. The symbol δ_J^m reflects that δ_J^m Map tasks should be scheduled at present in order to meet job J's map-deadline, as well as to meet the reduce-deadline (job deadline) δ_J^r Reduce tasks should be scheduled. In the scheduling process, we take δ_J^m and δ_J^r as the basic criteria of priority allocation.

But there are some special cases should be considered: (1) At the beginning of the job be submitted, there is no data available, so the scheduler can't estimate the required slots or the completion time of tasks. It is not possible to allocate the priority for this job at that time. In this case, the job's precedence is over than the others. (2) In some scenarios, jobs may have already missed their deadline. The strategy we use is the same as the previous case: set such jobs' tasks with the highest priority. Algorithm 2 proposes the MTSD scheduling method. The input parameter t which containing the free map slots and reduce slots, represents a request to ask for waiting tasks. And the return value is a collection T which means the tasks assignment.

Algorithm 2 MTSD scheduling algorithm

```
Collection assignTask(TaskTracker t)
M ←t.freeMapSlots          //M is the map slot collection
R ←t.freeReduceSlots       //R is the reduce slot collection
J ← uncompleted jobs in system    //the job collection
JQ ← NULL                  //job queue
ComputeSlotRequirement(J)  //compute δ_J^m and δ_J^r
ComputePriority(eJ)        // according t δ_J^m , δ_J^r and time
JQ ← sortJobByPriority(J)  //sort jobs in decending order
for each slot m ∈ M do
   while JQ not empty  do
      job ← pop(JQ)
      if job exists waiting map task
         then  T ←T ⁱ  < m , obtainMapTask(job) >
               break
      end if
   end while
end for
JQ ← sortJobByPriority(J)  //sort jobs in decending order
for each slot r ∈ R do
   while JQ not empty  do
      job ← pop(JQ)
      if job exists waiting reduce task
         then  T ←T ⁱ  < r , obtainReduceTask(job) >
               break
      end if
   end while
end for
return T
```

IV. EXPERIMENT RESULTS

In this section, we evaluate and compare the performance of our scheme with traditional scheduling algorithm. We have carried out a comprehensive set of experiments in order to evaluate the effectiveness of MTSD scheduling algorithm. We evaluate the MTSD algorithm from data locality and the performance of scheduling. We use three typical MapReduce programs as the running example, WordCount, Join and Girdmix. In the experiments, we use two node levels and let $\delta = 1.2$. Each node has four map slots and four reduce slots. The hardware configuration in our experiments is showed in Table I.

TABLE I. THE HARDWARE CONFIGURATION

Level	CPU	Memory	Amount
1	4-core, 3.07 GHZ	4G	8
2	4-core, 2.7s GHZ	4G	10

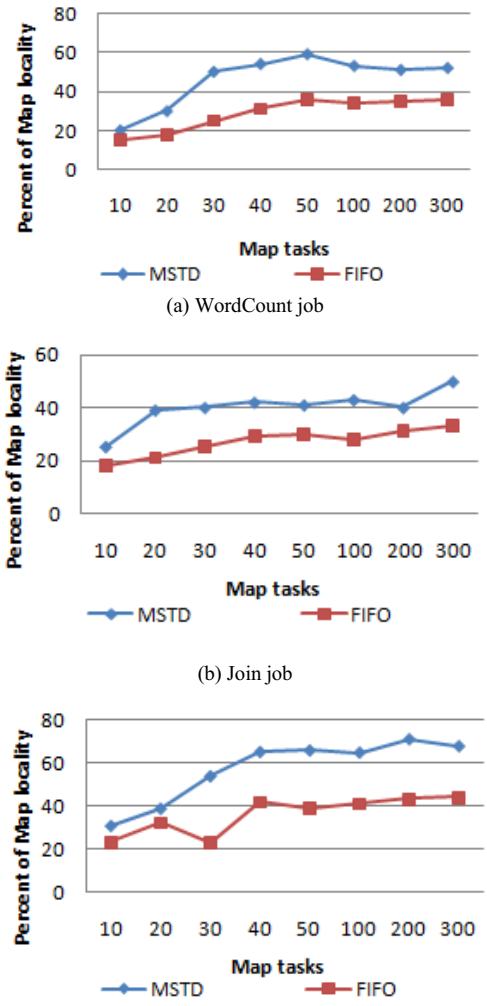


Figure 2. The Map tasks' data locality

A. *Data locality*

According node classification algorithm, the data distribution on the node is based on the node classification. In this way, the data migration from node to node can modify the data locality. We run WordCount, Join and Gridmix program to measure the proportion of the data locality. The experiments are divided into two groups: a group with the node classification algorithm and a group without node classification algorithm which is traditional algorithm. The results are shown in Figure 2 as follow.

From Figure 2 (a) to Figure 2 (c), we can see that the proportion of data locality of nodes with node classification algorithm is higher than the nodes without node classification algorithm. The proportion of data locality can improve about 57%. So we can draw the conclusion that node classification algorithm can improve the data locality.

B. *Job scheduling*

In order to evaluate the performance of MTSD algorithm, we use three jobs: J1, J2 and J3 stand for Gridmix, Join and WordCount respectively. The three job's assignments respectively in three time points: S1, S2, S3. Each job has its deadline. In Figure 3 (a), (b) and (c), the symbols M1, M2 and M3 means three jobs' map-deadline; R1, R2 and R3 means the three jobs' reduce-deadline. The horizontal indicates the time, and the unit is second.

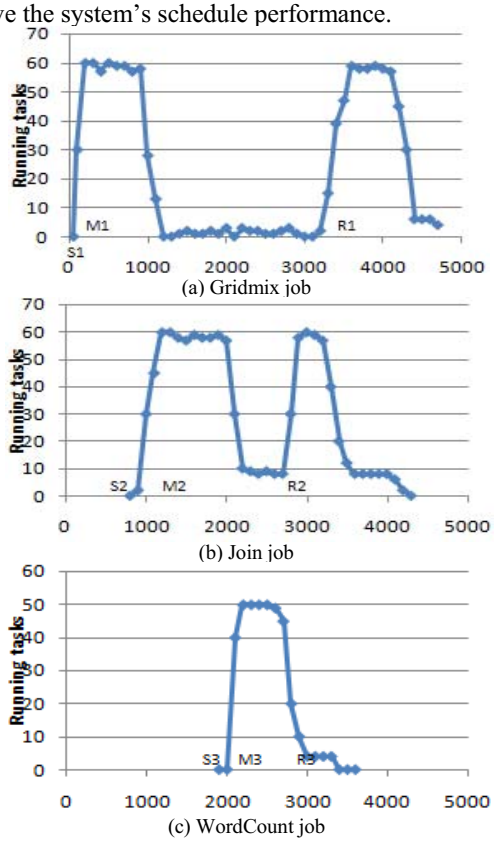
At the time S1, in Figure 3 (a), only J1 is running, and the current free slot number is 60, MTSD algorithm will assign all slots to J1 for map task. At S2 moment, the job J2 arrives, M2-S2 < M1-S1, which means that J2 is more urgent than J1, so J2 will have a higher priority than J1. However, at S3, the user submits J3, and M3-S3 < M2-S2, obviously, J3 has a higher priority than J2, so j3 is given privileged access of slots. From S3 to M3, the system assign 50 slots to J3, the rest of the 10 slots assign to J2. When J3, J2 finished, MTSD will assigned most of the slots to j1 in order to assure that j1 can finish the task before m1. The reduce-deadline scheduling is similar to map-deadline scheduling. From Figure 3 we can see that the closer the task to deadline, the higher priority the job gets.

From Figure 3, MTSD scheduling algorithm can satisfy the job's Deadline constraints, especially in multi-jobs environment. MTSD schedules the tasks with their urgent level. But, in some extreme cases, scheduler can't meet requirements. And if the running tasks' deadlines don't set reasonable, in some cases all the work can't finish their tasks before deadline.

V. CONCLUSION

User constraints such as deadlines are important requirements which are not considered by existing cloud-based data processing environments such as Hadoop. The MTSD proposed in this paper focuses on user's deadline constraints problem. In this algorithm, Node classification algorithm is proposed to divide the nodes into different type

with their computation ability. With the classification results, we get the data distribution principle for the input data. And the experiments show that the MTSD algorithm improve the data locality. The experiments show that 90% tasks can meet the deadline constraints. We propose two stages scheduling to accurate the scheduling progress, and improve the system's schedule performance.



(a) Gridmix job

(b) Join job

(c) WordCount job

Figure 3. The progress of job scheduling

ACKNOWLEDGMENTS

This work is supported by the National Natural Science Foundation of China (61103047), National Post doctor Science Foundation of China (20100480936).

REFERENCES

[1] C. Chu, S. kim, Y. Lin, Y. Yu, G. Bradski, A. Ng, and K. olukotun, Map-reduce for Machine Learning on Multicore, http://www.cs.standford.edu/peop;e/ang//papers/nips06-mapreducemulticoure.pdf. 2006.

[2] J. Ekanayake, S. Pallickara, and G. Fox, "MapReduce for data intensive scientific analyses," In Proceedings of the 2008 IEEE Fourth International Conference on eEcience, 2008, pp.277-284, doi: 10.1109/eScience.2008.59.

[3] G. Mackey, S. Sehrish, J.Bent, J.Lopez, S. Habib, and J. wang, "Introducing map-reduce to high end computing," In Proceedings of the 2008 3rd patascale data storage whokshop, 2008, pp.1-6, doi: 10.1109/PDSW.2008.4811889.

[4] A. Matsunaga, M. Tsugawa, and J. Fortes, "CloudBLAST: combining MapReduce and virtualization on distributed resources for bioinformatics applications," In Proceedings of 4th IEEE international conference on eScience, 2008, pp.222-229, doi: 10.1109/eScience.2008.62.

[5] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," In Communications of the ACM, Vol.51, Issue 1, 2008, pp.107-113, doi: 10.1145/1327452.1327492.

[6] M. Zaharia, A. Konwinski, A. D. Joseph, R. Katz, and I. Stoica, "Improving MapReduce Performance in Heterogeneous Environments," In Proceedings of the 8th USENIX conference on Operating Systems design and implementation, 2008, pp.29-42.

[7] M. Zaharia, D. Borthakur, J. S. Sarma, K. Elmeleegy, S. Shenker, and I. Stoica, "Job Scheduling for Multi-User MapReduce Clusters," Technical Report of University of California, Berkeley, 2009.

[8] http://hadoop.apache.org/mapreduce/docs/r0.21.0/capacity_scheduler.html, 2011.

[9] K. Kc, K. Anyanwu, "Scheduling Hadoop Jobs to Meet Deadlines," IEEE Second International Conference on Cloud Computing Technology and Science, 2010, pp.388-392, doi: 10.1109/CloudCom.2010.97.

[10] J. Polo, D. Carrera, Y. Becerra, J. Torres, E. Ayguade, M. Steinder, and I. Whalley. "Performance-Driven Task Co-Scheduling for MapReduce Environments," In IEEE proceedings of Network Operations and Management Symposium, 2010, pp.373-380, doi: 10.1109/NOMS.2010.5488494.

[11] J. Xie, S. Yin, X. J. Ruan, Z. Y. Ding, and Y. Tian, "Improving MapReduce Performance through Data Placement in Heterogeneous Hadoop Clusters", In IEEE International Symposium on Parallel & Distributed Processing, Workshops and PhdForum, 2010, pp.1-9, doi: 10.1109/IPDPSW.2010.5470880.

[12] M. Zaharia, D. Borthakur, J. S. Sarma, K. Elmeleegy, S. Shenker, and 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.

[13] X. H. Zhang, Z. Y. Zhong, S. Z. Feng, B. B. Tu, and J. P. Fan, "Improving Data Locality of MapReduce by Scheduling in Homogeneous Computing Environments," In IEEE 9th International Symposium on Parallel and Distributed Processing with Applications, 2011, pp.120126, doi: 10.1109/ISPA.2011.14.

[14] Thomas Sandholm and Kevin Lai, "Dynamic proportional share schedul ing in hadoop," In JSSPP '10: 15th Workshop on Job Scheduling Strategies for Parallel Processing, 2010.

[15] Ashraf Aboulnaga, Ziyu Wang, and Zi Ye Zhang, "Packing the most onto your cloud," In CloudDB '09: Proceeding of the first international workshop on Cloud data management, pages 25–28, New York, NY,USA, 2009. ACM.

[16] Kristi Morton, Magdalena Balazinska, and Dan Grossman, "Paratimer: a progress indicator for mapreduce DAGs," In SIGMOD '10: Proceedings of the 2010 international conference on Management of data, pp.507–518, New York, NY, USA, 2010. ACM.