Multi-threading based Map Reduce tasks scheduling

Conference Paper · April 2014
DOI: 10.1109/IACS.2014.6841943

CITATIONS
11

READS
214

4 authors:

Althebyan Qutaibah
Al Ain University of Science and Technology
26 PUBLICATIONS **162** CITATIONS
SEE PROFILE

Omar Ahmad Alqudah
Jordan University of Science and Technology
6 PUBLICATIONS **55** CITATIONS
SEE PROFILE

Yaser Jararweh
Jordan University of Science and Technology
214 PUBLICATIONS **1,641** CITATIONS
SEE PROFILE

Qussai Yaseen
Jordan University of Science and Technology
33 PUBLICATIONS **192** CITATIONS
SEE PROFILE

2014 5th International Conference on Information and Communication Systems (ICICS)

Multi-Threading Based Map Reduce Tasks Scheduling

Qutaibah Althebyan , Omar ALQudah, Yaser Jararweh
Software Engineering & Computer Science Departments
Jordan University of Science and Technology
Irbid, Jordan
Qaalthebyan@just.edu.jo, Yaser.amd@gmail.com,
qomar86@yahoo.com

Qussai Yaseen
Computer Science Department
Yarmouk University
Irbid, Jordan
{qyaseen@yu.edu.jo}

***Abstract*— Map Reduce is a parallel and a distributed computing framework used to process datasets that have large scale nature on a cluster. Due to the nature of data that needs to be handled in the Map Reduce problem which involves huge amount of data, many problems came up that are of great importance. Scheduling tasks is considered one of these major problems that face Map Reduce frameworks. In this paper, we tackled this problem and proposed a new scheduling algorithm that is based on a multi-threading principle. In our proposed algorithm, we divided the cluster into multi blocks where each one of them is scheduled by a special thread. Two major factors are used to test our algorithm; the simulation time and the energy consumption. Our proposed scheduler is then compared with existing schedulers and the results showed the superiority and the preference of our proposed scheduler over the existing schedulers.**

Keywords—Map Reduce, Schedulers; Cloud Computing; Clustering; Hadoop; Scalability.

I. INTRODUCTION

New IT technologies lead to an explosion in the amount of data; in 2012 the world has produced about 2.5 Exabyte's of data daily [1]. A recent study estimates a 7 Zettabyte of data to be generated in 2014 [2]. These data are generated due to the explosion of using electronic devices such as computers, sensors networks and smart phones as well as the use of social communication sites in several daily activities. This huge amount of generated data needs to be handled using novel and efficient data management systems. So, the big data terms comes to live. The big data term is currently used to represent such huge and complex data sets that no traditional data processing systems can handle efficiently. Big Data according to McKinsey [3] refers to "datasets whose size are beyond the ability of typical database software tools to capture, store, manage and analyze". Big data needs new technologies that will be able to extract value from those datasets; such processed data might be used in other fields such as artificial intelligence, data mining, etc. According to IBM [4] Big Data consists of three attributes: variety, volume and velocity. Variety means that the produced data are different in type; it can be digits, texts, audios, videos, log files, or any other type. All these accumulate huge amounts of data. The second attribute is the volume where the data that will be analyzed are too huge; they might reach hundreds or thousands of terabytes. The third attribute is the velocity where processing and analyzing data must be done in a fast manner to extract value of data in appropriate time. The above characteristics drive for developing new methodologies to deal with such huge amounts of data. So, comes to existence the term "big data management".

There are prominent technologies that have wide spread in big data operations. Examples of these include but are not limited to, cloud computing, distributed systems, data warehouse, Hadoop, Map Reduce, etc.

These technologies lead to producing huge amounts of data. Such data needs to be handled and processed in a special manner. Usually they are distributed among large scale clusters in order to be utilized. In fact, handling these clusters lead for a need of high optimized schedulers with a constraint that they should guarantee high level of availability, high cluster utilization, data locality and scalability.

Map Reduce is one of these frameworks that are produced to handle the problem of big data management. It is a software framework introduced by Google for processing large amount of data in a parallel manner [5]. Map Reduce framework suffers from many drawbacks that severely harm the data management efficiency and performance. Such drawbacks are related to the Map Reduce task scheduling techniques as will be discussed in the next section. Hence, the problems in Map Reduce scheduling techniques need some amendments especially, in the task scheduling in order to get more cluster utilizations, as well as other factors that affect the performance of the whole system. So, a need for a scalable scheduler to face these kinds of problems is needed.

So, the contribution of this paper can be summarized as follows. It aims to enhance overall system performance through our proposed scheduler. In fact, our proposed technique improves the overall system performance. It speeds up system by proposing a new scheduling algorithm. It improves cluster utilization by dividing the cluster into blocks. It also improves system scheduling by using multi-threading scheduling. Moreover, it improves data locality by using fair share method for each block. And finally it reduces the amount of energy used for cluster operations.

978-1-4799-3023-4/14/$31.00 ©2014 IEEE

The rest of the paper is organized as follows. Section II discussed some related work. Section III explained our proposed scheduling algorithm (MTL). In section IV we showed our experimental results. Finally, we concluded our work and draw some future highlights in section V.

II. BACKGROUND AND RELATED WORK

As noted in the introduction section, there are prominent technologies that have wide spread in big data operations. Examples of these include but not limited to, distributed systems, cloud computing, data warehouse, Hadoop, Map Reduce, etc. Distributed systems are those systems that are used to distribute tasks on multi computers that are connected through networks. These tasks will be processed in parallel to achieve better performance while reducing cost. Whereas data warehouse is a special database designed for reporting and storing vast amount of structured data. Warehouse uses different technologies such as ETL (Extract, Transform, and Load) to upload data from operational stores, business intelligence. Cloud computing is a parallel distributed system with scalable resources. It provides services via networks. Cloud computing offers good environment for techniques that need scalable resources and distributed systems such as Map Reduce framework [5]. There are many examples for cloud computing providers, namely, Amazon EC2, Aneka, and Microsoft Azure. Hadoop is an open source software produced by apache to handle huge datasets through distributed systems. Hadoop also uses Map Reduce framework [6]. It is usually decomposed into two main components: Hadoop distributed file systems "HDFS" and Map Reduce Framework. Map Reduce is a software framework introduced by Google for processing large amounts of data in a parallel manner [7]. It is mainly based on massive parallel computing infrastructure that exploits the massive computing infrastructure available to tackle the big data major issues. Map Reduce framework provides a set of features such as: user defined functions, automatic parallelization and distribution, fault tolerance and high availability by data replicating. Map-Reduce framework follows master/slave technique where Job tracker is responsible for scheduling, monitoring and re-executing failure tasks on slave machines. Task tracker is responsible for processing tasks that have been assigned by job tracker.

As illustrated in the previous paragraphs, all these technologies produce huge amounts of data. This data is a result of several jobs of different diversity which usually share large scale computing clusters. This in terns, needs to be processed in a specialized manner. A major step in this specialized processing of the data is scheduling. Unless the scheduling is not optimized, dealing with this big data will not be of that much good results that will enhance the system utilization as a whole, such as high availability, high cluster utilization, fairness, data locality (computation near to its input data), scalability, etc., which makes it not easy to achieve an efficient scheduling model. In fact, there are many techniques that are currently used for scheduling tasks in the Map Reduce framework. Namely [8]:

- First Input First Output "FIFO". This scheduler usually processes the first job submitted then the next respectively, and so on. In this criteria; the jobs are processed by the order of jobs as first comes first severed.
- Fair share scheduler: it gives every job a fair share of the cluster over a respected time slot. This time slot is predefined in order to prevent any greedy jobs from resources reserving [9].
- Matchmaking scheduler: It gives every slave node a chance to process local tasks. The main idea in this algorithm is to allow nodes to process local tasks from another job in the queue. Hence, the scheduler does not consider the jobs order to be important. To explain, when a scheduler does not find local task in first job, the scheduler keeps searching in next job to find local map task for that node. If the node does not find more local tasks, at this heartbeat, no any non-local task will be assigned which gives more fair. In the second cycle, if a node still does not find more local tasks to free slots and to avoid starvation, the scheduler will assign only one non local map task to the node that has free slots in every heartbeat interval [10].
- Capacity scheduler: it makes partitions for the resources and divides them into multi pools with dedicated job queue for every pool. The authors of this scheduler observed that the order of executing job has an impact on all execution times for all jobs. So, they try to concentrate more on ordering jobs to achieve best system performance. Balanced pool uses Johnson's algorithm that has optimal scheduling for two stages problems such as Map and Reduce scheduling [11].
- Dynamic priority scheduler: it uses parallel scheduling, that is, it uses one of the previous methods with a priority condition. This condition differs from algorithm to another, but most of priority algorithms use job deadline, pricing, or other thresholds [12].

Every scheduler from the above schedulers has disadvantages. For example, in FIFO scheduler, small jobs have a problem in waiting large job processing. FIFO also does not respect data locality for jobs that are needed in Map-Reduce scheduling framework. Fair share scheduler demands more time for job scheduling in context switch between jobs. Capacity scheduler does not respect data locality. And finally Dynamic priority scheduler interests to achieve special goals. So, this scheduler does not respect data locality too.

Many other schedulers exist in the literature. Most of these schedulers suffer from the same problems the above mentioned schedulers suffer from. Hence, we limited ourselves in this paper in listing the above schedulers.

III. PROPOSED APPROACH: MULTI –THREADING LOCALITY SCHEDULER (MTL)

Our goal is to improve the overall performance of the Map-Reduce system and to solve all problems that are explained in the previous sections. Hence, a multi-level management approach will be used for the cluster. In particular, we will use a multi –threading approach. In our proposed approach multi-threading system will be used which is responsible for

scheduling cluster. For each thread, there is a special block to search in it for data locality. In this scenario, a good performance can be achieved, especially with large clusters that are exposed by intensive jobs which vary in size.

In our proposed algorithm, we divide the cluster into N blocks. Each block contains a number of commodity machines to process and store input data. Each block is scheduled by a special thread that schedules the jobs in the wait queue. Once a new job arrives to the cluster, the map-reduce scheduler contacts the namenode to determine the rack that includes the largest proportion of data locality tasks for this job.

When any job needs to be processed, the threads start searching in their blocks node for local map task, where each thread takes information about current task and starts searching in their blocks. Once a thread finds a local data for this task, it immediately notifies other threads to stop searching for this task and starts searching for next tasks. If all threads could not be able to find more local task, the threads starts in assigning just one non-local map task for each node in the block for this heartbeat.

So, our method differs from the previous methods in dealing with synchronizations blocks scheduling, through using multi –threading scheduling. Our proposed approach has a significant advantage of reducing scheduling time, especially, if we have large clusters with large number of tasks. Also, our proposed algorithm has a mark plus on the pervious methods in solving data locality problem by multi searching using multi-threading.

In our algorithm, we assume the following

- Numbers of nodes are divided into multi racks; each rack is divided into N blocks.
- Denote T to represent the number of threads. Since we determine one thread for each block then (N=T) where N= number of blocks.

So, we can summarize our algorithm to work as follows:

- Global thread contact Namenode to retrieve information about job to determine which rack is needed.
- At this moment, the wait queue will be ready
- For each job in the waiting queue, each task will be processed in two levels:
    - Process all local map tasks: all threads start searching on the same assigned task. Each thread searches in its block nodes. Once the first thread finds local data for task, notify all other threads to stop searching for this task and moves to the next task. In case this task is not found by any thread, the task stills without processing and go to level 2 for processing non local map tasks.
    - Process only one non local map task for each node that has free slots in heartbeat interval.

Fig. 1 represents pseudo code for our proposed algorithm, where: *B*: Number of Blocks and *N*: Number of Nodes.

A. A Working Example Scenario

Suppose that there is a cluster with 9 nodes. This cluster is divided into three blocks where each block contains 3 nodes. Fig. 2 illustrates our method for this example:

Each job in the job queue consists of three tasks; each task has a square with a special color that identifies data of map task. Each node has to map two map slots which are represented as a frame with a special color. The mission of the proposed algorithm is to match each square task in the job queue with same color from the frames tasks in map slots to achieve high data locality. At the beginning of each block, there exists a thread (responsible for scheduling tasks) to map slots for their blocks only.

From Fig. 3, each thread takes information about current task needed data (in this example color of task) and starts searching in their nodes.

Input: User parameter (Job Queue; N; B)
Output: Scheduled Queue

```
1.  Partition_ Blocks( N, B);
2.  Assign_ Threads(B);
3.  For each ( Job in Job Queue)
4.     For each ( thread in Threads)
5.        Search_ Locality(block , Job);

6.  Search_ Locality(block , Job){
7.  If (Job has unassigned Local Task)
8.     For each ( node in block)
9.        If (node has local data)
10.          Assign node→Task;
11.          Notify other threads to stop search for this task;
12. Else
13.    Assign node→Task;// assign only one un local task
}
```

Fig. 1. Multi-Threading Locality Algorithm

Once a thread finds locality for that task in a node, the thread will assigns this task to a node that contains the data, then the thread notifies the other threads to stop searching on that's task. In Fig. 2, thread one finds data locality for task one (blue square) in first node from first block, then all threads move to second task (red square) and start searching locally, second task is found by thread one (red square), and third task (orange square) is found by thread three. The process continues like this until all tasks will be taken care of.

Hence, fulfilling the job is processed in three movements, due to number of tasks in that job. This example illustrates our proposed algorithm with few numbers of nodes and tasks. Although multi- threading system has management overhead, but we proved with real experiments with large number of nodes and tasks, the scalability of our proposed algorithm, the management overhead has little effects, with noticeable performance improvement.

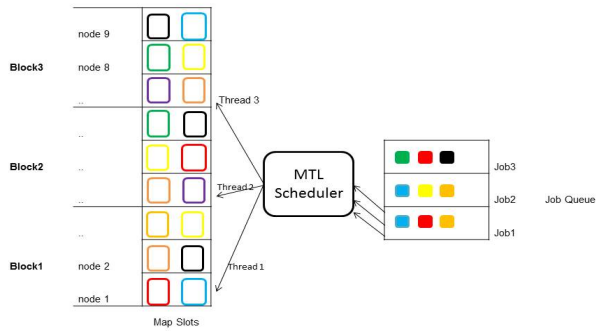

Fig. 2. Working scenario for our Proposed algorithm

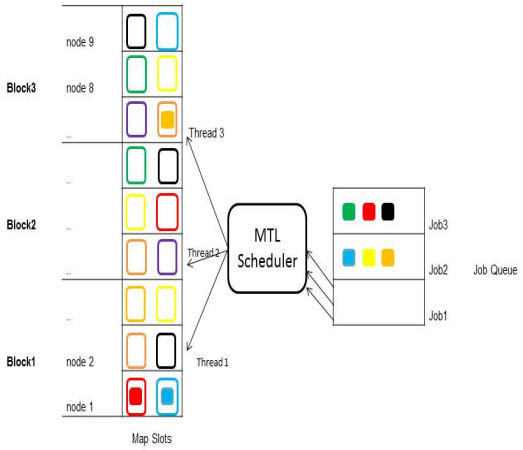

Fig. 3. Proposed algorithm example after first job

IV. EXPERIMENTAL RESULTS

In order to prove the superiority of our proposed algorithm over existing algorithms we run a simulation for our proposed map- reduce framework. Our simulation uses map phase only, where the scheduler is used to schedule map phase without any change in the reduce phase. For this simulation, we use CloudSim simulator that runs on Net Beans [13]. For our Map Reduce system we need to use some of the CloudSim characteristics such as scalability to prove goodness and improvements achieved by our proposed algorithm. In this simulation, every host is connected to special storage using the same id. The host specification is fully described in Table 1 where every host has one virtual machine. The hosts in our proposed scheduler are divided among number of blocks.

In this simulation, the tasks are divided among a number of jobs where each task has several properties as it explained in Table 2.

TABLE I. HOST PROPERTIES

Host ID	ID (0- number of hosts)
Storage Capacity	1 TB
MIPS for CPU	1024
Memory Capacity	2 GB
Network Bandwidth	10 Mbps
Virtual Machine Scheduler	Space shared

TABLE II. TASK PROPERTIES

Task ID	ID (0-number of tasks)
Job ID	Job ID which task belongs to
Task Length	Random # between 1-500
Number of CPU	1
Input File Size	64MB
Output File Size	64MB
Number of File Per Task	1

The following parameters will be used in our simulation. The following is a brief description of each:

No_ Block: is special for our algorithm. It determines number of threads that searches for data locality.

No_ Host: is the number of nodes that will be used for this experiment.

No_ File: is the number of files that contain data for tasks and will spread among nodes.

No_ Job: is the number of jobs that come from different users.

No_ Task: is the number of tasks that need to be executed. The number of tasks is distributed among the jobs randomly.

In our simulation, we need to show the superiority of our proposed scheduler MTL against other existing schedulers. So, we run our scheduler MTL against other schedulers, specifically, we chose FIFO scheduler, Matchmaking scheduler and the delay scheduler. In fact we conducted comparisons

among them in terms of two factors which are the simulation time factor and the energy consumption of nodes factor.

Different values for the parameters have been conducted. These values vary from small values of nodes and tasks to large numbers. All the results as will be shown shortly proved the scalability of our proposed scheduler MTL. This proves the superiority of our MTL scheduler over all other existing schedulers. Only samples of our results will be shown that proves the scalability of our MTL scheduler.

A. 10000 tasks Experiment:

We noticed from different experiments that data locality cannot be considered as a distinguishing factor because with small numbers of tasks most of the algorithms will have 100% data locality. Although this percentage of data locality will not be 100% for most of them, still it will be 100% for the matchmaking algorithm and the delay algorithm. So, we use simulation time and energy consumption as our main comparison criteria to compare between algorithms.

The following two figures fig. 4 and figure fig. 5 show the behavior of all algorithms with 10000 tasks in terms of simulation time and energy consumption respectively.

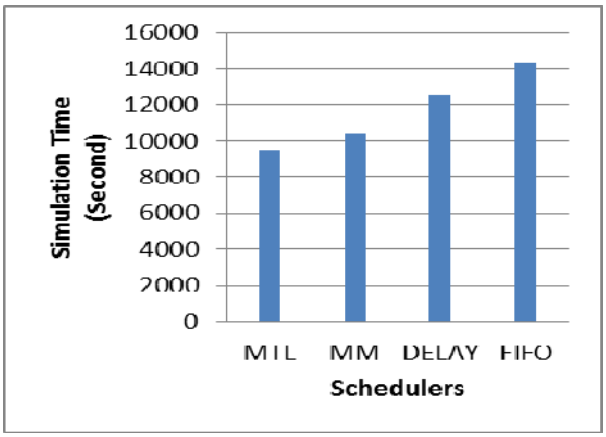

Fig. 4. Simulation Time for 10000 tasks

As can be noticed in fig. 4 and fig. 5, our MTL scheduler achieves the best results in terms of simulation time and energy consumption. Although the difference is slight between our MTL scheduler and the MM scheduler, but while increasing the number of tasks, the gap between our MTL scheduler and all other schedulers will increase. This proves the scalability of our scheduler and the superiority of it in terms of both simulation time and energy consumption.

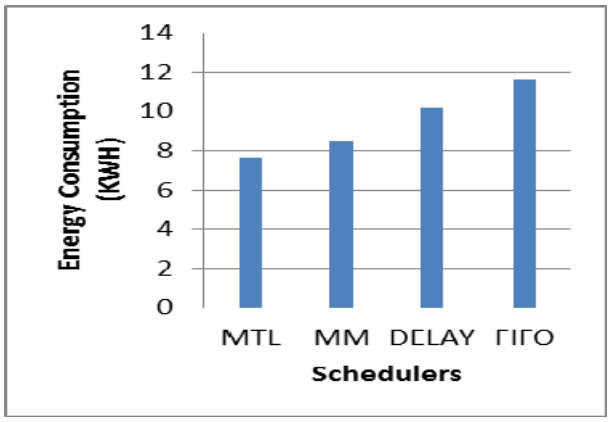

Fig. 5. Energy Consumption for 10000 tasks

B. 100000 tasks experiment:

The following two figures fig. 6 and figure fig. 7 show the behavior of all algorithms with 100000 tasks in terms of simulation time and energy consumption respectively.

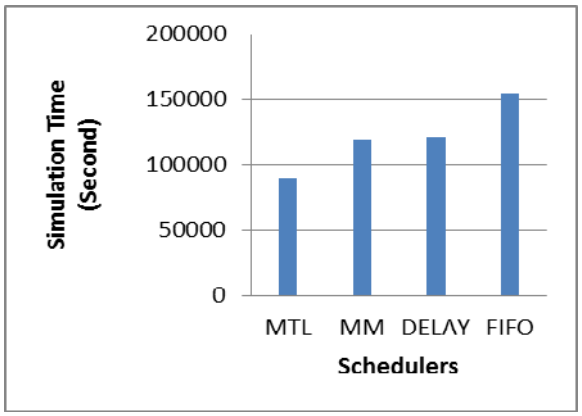

Fig. 6. Simulation Time for 100000 tasks

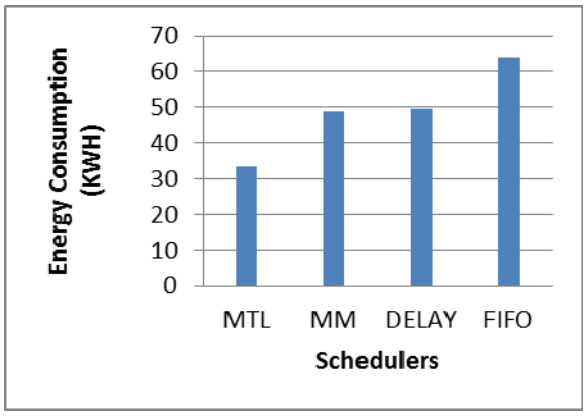

Fig. 7. Energy Consumption for 100000 tasks

In the 100000 tasks experiment, we use huge number of tasks to simulate big data behavior. Our MTL algorithm achieves the best values, which gives it a promising future in big data processing. In fact the100000 tasks experiment clearly proves that our MTL is scalable. The MTL achieves good results in simulation time which in terns is reflected on the energy consumption. The above Figures fig. 6 and fig. 7 show the superiority of our proposed algorithm, where MTL takes about 90000s to process its map tasks. Whereas all other methods take over 120000s their process it maps tasks, which is a huge save of simulation time.

To give a clear image of the superiority of our MTL scheduler, the following table summarizes the percentages of how much our proposed algorithm MTL outperforms all other schedulers.

TABLE III. MTL ALGORITHM IMPROVEMENT

Number of Tasks	Matchmaking	Delay	FIFO
10000	24%	27%	45%
30000	24%	27%	46%
50000	31%	32%	47%
100000	35%	36%	50%

We can notice from the above table that our proposed algorithm gives better results than Matchmaking of about 24 %, about 27% from the Delay, and about 45% from the FIFO in 10000 tasks.

We can also notice that the improvements' percentages have increased with increasing the number of tasks. The improvements will rise to achieve about 35% from Matchmaking, about 36% from Delay, and about 50% from FIFO in 100000 tasks.

V. CONCLUSIONS AND FUTURE WORK

In this paper, we proposed a new scheduler for meeting the explosion in the amount of data that generated from different users every time. The new algorithm is used for job scheduling in map –reduce system. The proposed algorithm is called "Multi – Threading Locality" MTL.

Our proposed algorithm is based on multi –threading principles, where the infrastructure is divided into blocks, every block is scheduled by a special thread, and this scheduling is achieved in a synchronous time. When a job comes to the system, each thread takes information about the task and start searching for locality in their blocks.

To test our proposed algorithm, we built a map –reduce system using CloudSim simulator in java language. The conducted experiments showed that our proposed algorithm achieved better results comparing with existing algorithms such as FIFO, Delay, and Matchmaking in terms of simulation time, and energy consumption. The experiments also showed that our proposed algorithm enhance about 29% from Matchmaking on average, about 31% from Delay and about 47 % from FIFO algorithm.

The experiments proved that our proposed algorithm has speed behavior and scalability infrastructure, to keep pace with the rapid growth of data.

As future directions, we intend to leverage from visualization principle by applying our proposed algorithm on multi virtual machines on physical nodes, and then compare such results with results produced from applying our algorithm on physical nodes. We also intend to implement the proposed algorithm "Multi-Threading Locality" in real Hadoop cluster with real big data such as Facebook social network. This will prove the speed and scalability that is produced by our proposed algorithm.

REFERENCES

[1] A.Mcafee, E.Brynjolfsson, 'Big Data: The Management Revolution', Harvard Business Review, 2012.

[2] R. Villars, C. Olofson, M.Eastwood, 'Big Data: What It Is and Why You Should Care', White paper, 2011.

[3] J.Manyika, M.Chui, B.Brown, J.Bughin, R.Dobbs, C.Roxburgh, A.H.Byers, 'Big Data: the next frontier for innovation, competition, and productivity', 2011.

[4] C.Statchuk, D.Rope, 'Enhancing Enterprise Systems with Big Data', IBM corporation, 2013.

[5] J.Schad, J.Dittrich, 'Flying Yellow Elephant: Predictable and Efficient MapReduce in the Cloud', Information Systems Group, Saarland University, 2010.

[6] The apache software foundation, (2012) Hadoop Apache, http://hadoop.apache.org/.

[7] J.Dean, S.Ghemawat, 'MapReduce: Simplified Data Processing on Large Clusters', OSDI 04: In the Proceedings of the 6th Symposium on Operating Systems Design and Implementation, 2004.

[8] B.T.Rao, L.S.S.Reddy, 'Survey on Improved Scheduling in Hadoop MapReduce in Cloud Environments', International Journal of Computer Applications, 34 (9) :0975 – 8887, 2011.

[9] M.Zaharia, D.Borthakur, J.S.Sarma, k.Elmeleegy, S.shenker, l.Stoica, (2009) ' Job Scheduling for Multi-User MapReduce Clusters', Technical Report No. UCB/EECS-2009-55.

[10] C.He, Y.Lu, D.Swanson, 'Matchmaking: A New MapReduce Scheduling Technique', Department of Computer Science and Engineering, University of Nebraska-Lincoln, Lincoln, US, 2011.

[11] A.Verma, L.Cherkasova, R.H.Campbell, (),'Two Sides of a Coin: Optimizing the Schedule of MapReduce Jobs to Minimize Their Makespan and Improve Cluster Performance'.

[12] J.Polo, D.Nadal, D.Carrera, Y.Becerra, V.Beltran, J.Torres and E.Ayguad´e, (2009) 'Adaptive Task Scheduling for MultiJob MapReduce Environments', XX Jornadas de Paralelismo, Coruña, Spain, 96-101.

[13] Y.Jararweh, Z.Alshara, M.Jarrah , M.Kharbutli, M.N.Alsaleh, 'TeachCloud: A Cloud Computing Educational Toolkit', International Journal of Cloud Computing (IJCC), 2 (2/3).