Enhance Performance of Mapreduce Job on Hadoop Framework using Setup and Cleanup

Priyam Jain
Department of Computer Science and Engineering
Bhopal Institute of Technology and Science
Bhopal, India

Satyaranjan Patra, PhD
Asst. Professor
Department of Computer Science and Engineering
Bhopal Institute of Technology and Science
Bhopal, India

Pankaj Richhariya
Prof. and Head
Department of Computer Science and Engineering
Bhopal Institute of Technology and Science
Bhopal, India

ABSTRACT
MapReduce is an effective programming model for large-scale data-intensive computing applications. Hadoop is an open-source implementation of MapReduce which has been widely used. The communication overhead from the big data sets' transmission affects the performance of Hadoop greatly. In consideration of data locality, Hadoop schedules tasks to the nodes near the data locations preferentially to decrease data transmission overhead, which works well in homogeneous and dedicated MapReduce environments. However, due to practical considerations about cost and resource utilization, it is common to maintain heterogeneous clusters or share resources by multiple users. Unfortunately, it's difficult to take advantage of data locality in these heterogeneous or shared environments [1]. To improve the performance of MapReduce in heterogeneous or shared environments, a data prefetching mechanism is proposed, In this paper, we can fetch the data to corresponding compute nodes in advance. It is proved that the proposal of this paper reduces data transmission overhead effectively with theoretical analysis. We also work on applying similar prefetching mechanisms to other phases in MapReduce, and researching on predicting the execution nodes of tasks in cluster computing to improve performance and the result are clearly shows that proposed system will takes a less execution time as compared to existing mapreduce job.

Keywords
Big data, Hadoop, Mapreduce, performance, prefetching mechanism, setup & cleanup

1. INTRODUCTION
MapReduce is a relatively young framework - both a programming model and an associated run- time system - for large-scale data processing. Hadoop [2] is the most popular open-source implementation of a MapReduce framework that follows the design laid out in the original paper. A combination of features contributes to Hadoop's increasing popularity, including fault tolerance ,data-local scheduling, ability to operate in a heterogeneous environment, handling of straggler tasks, as well as a modular and customizable architecture.

The MapReduce programming model [3] consists of a map(k1; v1) function and a reduce(k2; list(v2)) function. Users can implement their own processing logic by specifying a customized map() and reduce() function written in a general-purpose language like Java or Python. The map(k1; v1) function is invoked for every key-value pair hk1; v1i in the input data to output zero or more key-value pairs of the form hk2; v2i (see Figure 1). The reduce(k2; list(v2)) function is invoked for every unique key k2 and corresponding values list(v2) in the map output. reduce(k2; list(v2)) outputs zero or more key-value pairs of the form hk3; v3i. The MapReduce programming model also allows other functions such as (i) partition(k2), for controlling how the map output key-value pairs are partitioned among the reduce tasks, and (ii) combine(k2; list(v2)), for performing partial aggregation on the map side. The keys k1, k2, and k3 as well as the values v1, v2, and v3 can be of different and arbitrary types.

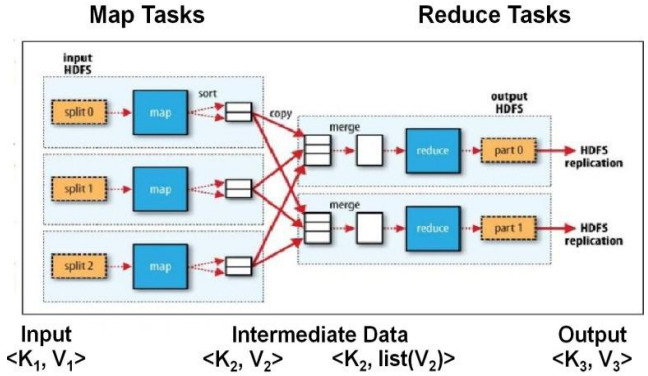


Figure 1: Execution of a MapReduce job.

A Hadoop MapReduce cluster employs a master-slave architecture where one master node (called JobTracker) manages a number of slave nodes (called TaskTrackers). Figure 1 shows how a MapReduce job is executed on the cluster. Hadoop launches a MapReduce job by first splitting (logically) the input dataset into data splits. Each data split is then scheduled to one TaskTracker node and is processed by a map task. A Task Scheduler is responsible for scheduling the execution of map tasks while taking data locality into account. Each TaskTracker has a predefined number of task execution slots for running map (reduce) tasks. If the job will execute more map (reduce) tasks than there are slots, then the map (reduce) tasks will run in multiple waves. When map tasks complete, the run-time system groups all intermediate key-value pairs using an external sort-merge algorithm. The intermediate data is then shuffled (i.e., transferred) to the TaskTrackers scheduled to run the reduce tasks. Finally, the reduce tasks will process the intermediate data to produce the results of the job.

The Map task execution is divided into five phases:

1. Read: Reading the input split from HDFS and creating the input key-value pairs (records).
2. Map: Executing the user-defined map function to generate the map-output data.
3. Collect: Partitioning and collecting the intermediate (map-output) data into a buffer before
4. spilling.
5. Spill: Sorting, using the combine function if any, performing compression if specified, and finally writing to local disk to create file spills.
6. Merge: Merging the file spills into a single map output file. Merging might be performed in multiple rounds.

The Reduce Task is divided into four phases:

1. Shuffle: Transferring the intermediate data from the mapper nodes to a reducer's node and
2. decompressing if needed. Partial merging may also occur during this phase.
3. Merge: Merging the sorted fragments from the different mappers to form the input to the reduce function.
4. Reduce: Executing the user-defined reduce function to produce the final output data.
5. Write: Compressing, if specified, and writing the final output to HDFS.

We model all task phases in order to accurately model the execution of a MapReduce job. We represent the execution of an arbitrary MapReduce job using a job profile, which is a concise statistical summary of MapReduce job execution. A job profile consists of dataflow and cost estimates for a MapReduce job j : dataflow estimates represent information regarding the number of bytes and key-value pairs processed during j's execution, while cost estimates represent resource usage and execution time.

2. LITERATURE REVIEW

One way to balancing load, Hadoop using HDFS distributed big size data to multiple nodes based on local disk storage capacity in clusters [4]. The data location is efficient in homogeneous environment where all nodes have identical both computing speed and disk capacity. In this environment computes same workload on all nodes representing that no data needs to be moved from one node to another node. All nodes are independent as well as can not share data between two nodes in cluster of homogeneous environment. In heterogeneous Environment or clusters have set of nodes where each node computing speed capacities and local disk capacity may be significantly different. If all nodes have different size workload then a faster computing (high performance) nodes can complete processing local data faster than slow computing (low- performance)nodes. Faster node finished processing data then result residing into its local disk and handle unprocessed data of remote slow node. When move or transfer unprocessed data from low performance (remote) node to high performance node is huge then overhead of data transmission is occurring. If wants Progress the MapReduce performance in heterogeneous environment then reduce the amount of data moved between low performances nodes to high performance nodes.

Improve The MapReduce performance in Various Environments:

A. Data Placement in Heterogeneous Hadoop Clusters

B. Heterogeneous Network Environments and Resource Utilization

C. Smart Speculative Execution Strategy

D. Longest Approximate Time to End.

A. Improve MapReduce Performance through Data Placement in Heterogeneous Hadoop Clusters [5].

We want improve the performance then minimize data movement between slow and fast nodes achieved by data placement scheme that distribute and store data across multiple heterogeneous nodes based on their computing speed.

1) Data placement in Heterogeneous- Two algorithms are implemented and incorporated into Hadoop HDFS. The first algorithm is to initially distribute file into heterogeneous nodes in a cluster. When all file fragments of an input files are distributed to the computing nodes. The second algorithm is used to reorganize file fragments to solve the data skew problem. There two cases in which file fragments must be reorganized. First, new computing nodes are added to an existing cluster to have the cluster expanded. When, new data is appended to an existing input file. In both cases, file fragments distributed by the initial data placement algorithm can be disrupted.

B. Improving MapReduce Performance in Heterogeneous Network Environments and Resource Utilization [6]

1) Resource stealing- When number of map and reduce slots are carefully chosen to gain optimal resource usage. Resource utilization is inefficient when there are not some enough tasks to fill all task slots as the reserved resources for idle slots are just wasted. Then Resource stealing, which enables running tasks to steal the residual resources and return them when new tasks are assigned. There is use of wasted resources to improve overall resource utilization and reduce job execution. First-come-Most(FCM) , Shortest-Time-Left-Most(STLM) , Longest –Time-Left-Most(LTLM) these are resource allocation policies.

2) Benefit Aware Speculative Execution –This mechanism predicts the benefit of launching new speculative tasks and greatly eliminates unnecessary runs of speculative tasks. Speculative execution in Hadoop was observed to be inefficient, which is caused by the excessive runs of useless speculative tasks. Benefit Aware Speculative Execution manages speculative tasks in a benefit-aware manner and expected to improve the efficiency.

C. Improving MapReduce Performance using Smart Speculative Execution Strategy [7]

Multiple speculative execution strategies are improving the performance in Heterogeneous as well as Homogeneous. But there are some Pitfalls degrade the performance. When existing strategies cannot work well, then they develop a new strategy, MCP (Maximum Cost Performance), which improves the effectiveness of speculative execution significantly.

When a machine takes an unusually long time to complete a task (the so-called straggler machine), it will delay the job execution time (the time from job initialized to job retired)

and degrade the cluster throughput (the number of jobs completed per second in the cluster) significantly. This problem handled speculative execution. A new speculative execution strategy named MCP for Maximum Cost Performance. We consider the cost to be the computing resources occupied by tasks, while the performance to be the shortening of job execution time and the increase of the cluster throughput. MCP aims at selecting straggler tasks accurately and promptly and backing them up on proper worker nodes.MCP is quite scalable, which performs very well in both small clusters and large clusters.

D. Improving MapReduce Performance in Heterogeneous Environments [8]

1) LATE, for Longest Approximate Time to End. If nodes in cluster ran at reliable speeds and no cost to initiation a speculative task on an ideal node then LATE policy would be best. The LATE algorithm has various applications. First, it is dynamic to node heterogeneity, because it will relaunch only the low performance tasks and only a small number of fragment parts of large file with slow performing as tasks. LATE prioritizes amongst the slow tasks based on how much they injure job response time. LATE also capture the number of slow performance tasks to limit argument for shared resources. In contrast, Hadoop's native scheduler has a fixed threshold [8].

3. RELATED WORK

To alleviate the performance degradation caused by data transmission, some related work is done. Data Prefetching is an effective approach to diminishing the data transmission overhead. To avoid directly modifying the native Hadoop, a bi-directional processing approach is proposed in HPMR [9]: computing fetches and processes data from the beginning of the input split data while the prefetching fetches data from the end of the input split data. Obviously, the computing has to fetch data by itself before meeting the data fetched by the prefetching, which discounts the benefits of data prefetching. While the proposal in this paper fetches data from the beginning of input data to reduce the overhead of data transmission at the maximum.

Some researchers focus on optimizing task scheduling algorithms or data replication policies to improve data locality in MapReduce [10]. These proposals only improve the probability of data locality in MapReduce and may increase the complexity of achieving load balance. The LATE scheduling algorithm is proposed for MapReduce in heterogeneous environments [11]. M. Zaharia et al., have proposed a delay scheduling algorithm, which addresses the conflict between locality and fairness in shared MapReduce cluster [13]. In MTSD [12], computing nodes are classified by computing capability and a modified task scheduling algorithm is studied. X. Zhang et al. have studied scheduling with consideration about data locality in homogeneous cluster [14]. DARE is a distributed adaptive data replication algorithm that is sensitive to the heterogeneity of computing nodes, and the more powerful nodes get more data replications [15].

4. PROBLEM DEFINITION

When an analysis is being conducted on Big Data it is of utmost importance that the data being dealt with is accurate and does not have any abnormalities. There are numerous factors that affect the performance of Hadoop such as hardware and software when handling huge amounts of data. Both the main components of Hadoop, that is, HDFS and MapReduce play a major role in its performance Hadoop and the results that are generated.

HDFS: The number of reading and writing operations performed on the nodes also affects the performance of Hadoop. The performance of HDFS also depends on whether the work is being performed on big or small dataset.

MapReduce: Tuning the number of map tasks and reduce tasks for a particular job in the workload is another way that performance can be optimized. If the mappers are running only for a few seconds then fewer mappers can be used for longer periods. Also performance depends on the number of reducers used which should be slightly less than the number of reduce slots in the cluster to improve performance. This allows the reducers to finish in one wave and fully utilizes the cluster during the reduce phase. MapReduce job performance can also be affected by the number of nodes in the Hadoop cluster and the available resources of all the nodes to run map and reduce tasks.

Shuffle tweaks: The MapReduce shuffle also helps to alter performance as it maintains a balance between the map and reduce functions. If adequate amount of memory is allocated to map and reduce functions then the shuffle can also be allocated enough memory to operate thereby improving performance. Therefore, a trade off needs to be carried out when allocating memory to tasks in MapReduce.

5. PROPOSED WORK

For analysis performance enhancement for MapReduce job we need:-

1. Dataset

In order to evaluate performance comparision between mapreduce job we need a dataset, a big or huge dataset through which we can evaluate performance.

2. Hadoop

Hadoop should be configure first because all the mapreduce job will work on hadoop framework, because hadoop comes with HDFS (hadoop distributed file system) which is used to stored such huge or large datasets and Mapreduce which is used to process this huge dataset.

3. MapReduce Job

MapReduce job will be developed on some IDE through which we can develop various mapreduce job jar file which is used to run on hadoop environment to compare performance.

6. PROPOSED METHODOLOGY:

Our Steps or Algorithm Steps will follow:

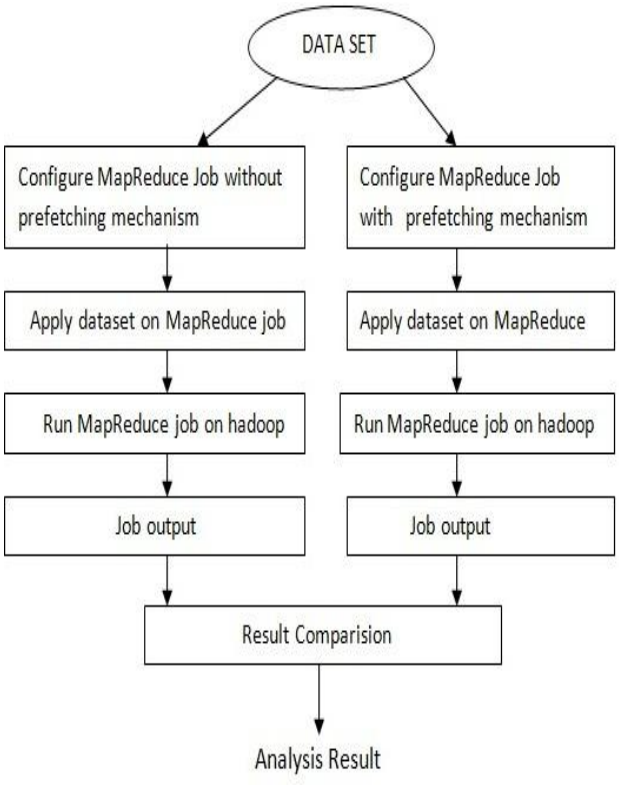
Step 1: first we collect dataset and apply these dataset into various mapreduce job.

Step 2: now we develop mapreduce job without prefetching mechanism or with prefetching mechanism on which we can apply the same datasets.

Step 3: Configure hadoop on which we can run the mapreduce job jar file.

Step 4: The dataset should be store in HDFS and mapreduce takes input from HDFS and perform mapreduce task and stored the mapreduce output in HDFS.

Step 5: In this step we are analysing the time taken or performance between various mapreduce job and check without prefetching mechanism is better or with prefetching mechanism.



Analysis Steps

7. EXPERIMENTAL & RESULT ANALYSIS

All the experiments were performed using an i5-2410M CPU @ 2.30 GHz processor and 4 GB of RAM running ubuntu 14. After thar we can install java which is a prerequest for hadoop, and than after we are configuring hadoop on ubuntu . After we can developed a mapreduce job for performing operation to find a mazimum length word in a overall file, in that we can create two mapreduce job , existing.jar which is based on pre-fetching mechanism to perform a operation in which prefetching is done only on the reducer phase and second one is proposed.jar which is based on map-side prefetchig mechanism as well reducer side to, so we can developed same prefetching mechanism on map side to enhance the overall performance.

After developing we can launch the existing.jar file on hadoop environment shown in figure 2.

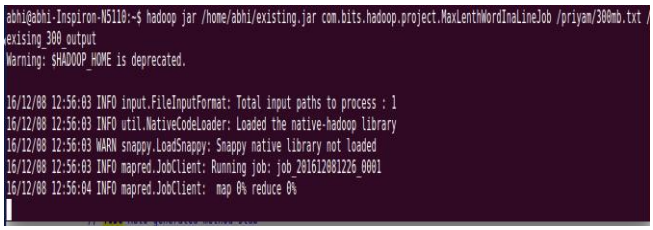
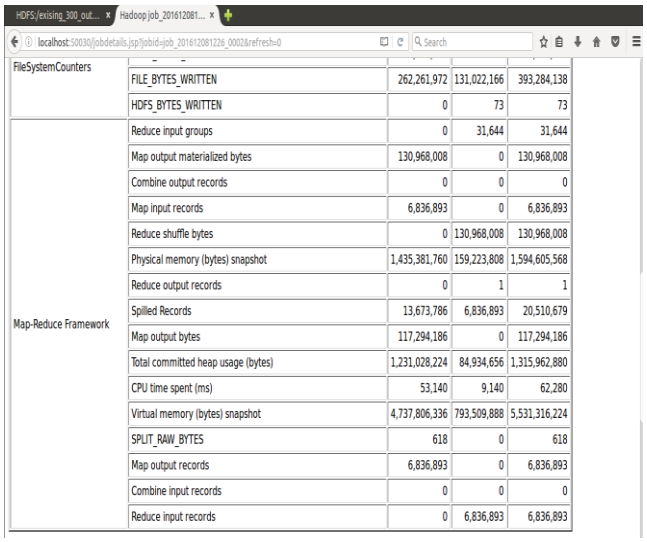


Figure 2. launching existing mapreduce job on 300 mb dataset

After execution of existing mapreduce job the final output is shown in output directory and the other performance fields such as shuffle bytes taken and time taken for execution, the execution time taken are shown in figure 3.



FileSystemCounters	FILE_BYTES_WRITTEN	262,261,972	131,022,166	393,284,138
	HDFS_BYTES_WRITTEN	0	73	73
Map-Reduce Framework	Reduce input groups	0	31,644	31,644
	Map output materialized bytes	130,968,008	0	130,968,008
	Combine output records	0	0	0
	Map input records	6,836,893	0	6,836,893
	Reduce shuffle bytes	0	130,968,008	130,968,008
	Physical memory (bytes) snapshot	1,435,381,760	159,223,808	1,594,605,568
	Reduce output records	0	1	1
	Spilled Records	13,673,786	6,836,893	20,510,679
	Map output bytes	117,294,186	0	117,294,186
	Total committed heap usage (bytes)	1,231,028,224	84,934,656	1,315,962,880
	CPU time spent (ms)	53,140	9,140	62,280
	Virtual memory (bytes) snapshot	4,737,806,336	793,509,888	5,531,316,224
	SPLIT_RAW_BYTES	618	0	618
	Map output records	6,836,893	0	6,836,893
	Combine input records	0	0	0
	Reduce input records	0	6,836,893	6,836,893

Figure 3. Time taken by existing mapreduce job

After existing mapreduce job execution is done than we launch a porposed.jar mapreduce job on hadoop shown in figure 4.

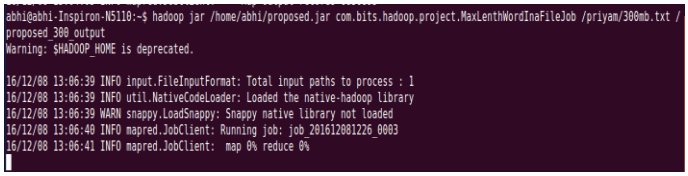
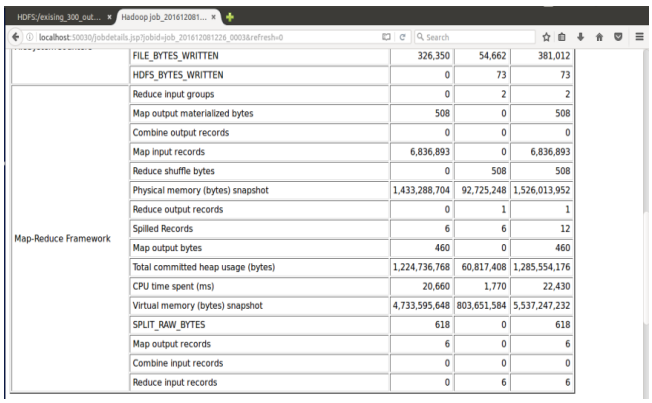


Figure 4.launching proposed mapreduce job on 300 mb dataset

After completing the execution of proposed.jar mapreduce job on 300 mb file the total time taken by proposed job are shown in figure 5.



	FILE_BYTES_WRITTEN	326,350	54,662	381,012
	HDFS_BYTES_WRITTEN	0	73	73
Map-Reduce Framework	Reduce input groups	0	2	2
	Map output materialized bytes	508	0	508
	Combine output records	0	0	0
	Map input records	6,836,893	0	6,836,893
	Reduce shuffle bytes	0	508	508
	Physical memory (bytes) snapshot	1,433,288,704	92,725,248	1,526,013,952
	Reduce output records	0	1	1
	Spilled Records	6	6	12
	Map output bytes	460	0	460
	Total committed heap usage (bytes)	1,224,736,768	60,817,408	1,285,554,176
	CPU time spent (ms)	20,660	1,770	22,430
	Virtual memory (bytes) snapshot	4,733,595,648	803,651,584	5,537,247,232
	SPLIT_RAW_BYTES	618	0	618
	Map output records	6	0	6
	Combine input records	0	0	0
	Reduce input records	0	6	6

Figure5. Time taken by proposed mapreduce job

After performing operation on 300 MB file , it is clearly that proposed mapreduce job taken less time as compared to existing mapreduce job. And after performing operation on 300 MB file we can perfrom same operation on different size file such as 300MB, 500MB, 1GB and total time taken in execution of both existing and proposed mapreduce job are shown in table which is mention in figure 6.

DATASET SIZE	EXISTING SYSTEM EXECUTION TIME (IN SEC)	PROPOSED SYSTEM EXECUTION TIME (IN SEC)
300 MB	62.28	22.43
500 MB	146.41	58.00
1 GB	265.91	81.44

Figure 6. Execution time taken by existing and proposed system on different dataset size

The tabular result which is shown in figure 6 are represented on graph shown in figure 7, on which it is clearly show that proposed mapreduce job are taking less execution time as compared to existing mapreduce job.

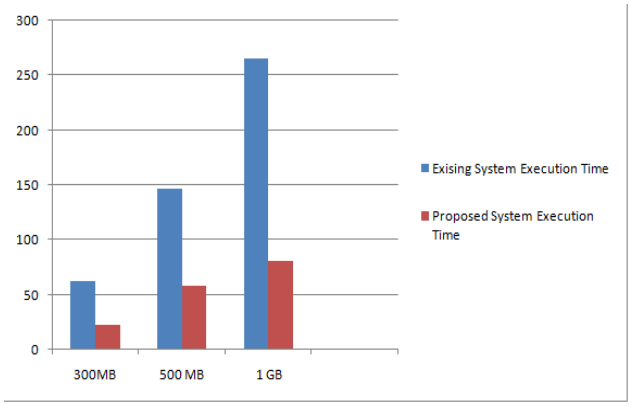


Figure 7. Graph representation of execution time taken

8. CONCLUSION

Hadoop MapReduce is now a popular choice for performing large-scale data analytics. we describes a detailed set of mathematical performance models for describing the execution of a MapReduce job on Hadoop. In this paper, we can fetch the data to corresponding compute nodes in advance. It is proved that the proposal of this paper reduces data transmission overhead effectively with theoretical analysis. We also work on applying similar prefetching mechanisms to other phases in MapReduce, which is clearly shown in table and figure 7 that prefetching mechanism on map phase will enhanced the performace of overall hadoop mapreduce framework.

9. REFERENCES

[1] Swathi Prabhu, Anisha P Rodrigues, Guru Prasad M S & Nagesh H R, "Performance Enhancement of Hadoop MapReduce Framework for Analyzing BigData", IEEE 2015, 978-1-4799-608S-9/1S

[2]Hadoop Wiki Website, Apache, http://wiki.apache.org/hadoop

[3] Improving MapReduce Performance Using Data Prefetching mechanism in heterogeneous or Shared Environments Tao gu,Chuang Zuo,Qun Liao , Yulu Yang and Tao Li, International Journal of grid and distributed computing (2013).

[4] "Improve the MapReduce Performance through complexity and performance based on data placement in Heterogeneous Hadoop Cluster " Rajashekhar M. Arasanal, Daanish U. Rumani Department of Computer Science University of Illinois at Urbana-Champaign.

[5] J. Xie, S. Yin, X. Ruan, Z. Ding, Y. Tian, J. Majors, A. Manzanares and X. Qin, "Improving MapReduce Performance through Data Placement in Heterogeneous Hadoop Clusters", IEEE International Symposium on Parallel & Distributed Processing, Workshops and PhD Forum (IPDPSW), (2010) April 19-23: Arlanta, USA.

[6] Improving MapReduce Performance in Heterogeneous Network Environments and Resource Utilization, Zhenhua Guo, Geoffrey Fox IEEE (2012)

[7] Improving MapReduce Performance Using Smart Speculative Execution Strategy Qi Chen, Cheng Liu, and Zhen Xiao, Senior Member, IEEE 0018-9340/13/$26.00 © 2013 IEEE

[8] S. Khalil, S. A. Salem, S. Nassar and E. M. Saad, "Mapreduce Performance in Heterogeneous Environments: A Review", International Journal of Scientific & Engineering Research, vol. 4, no. 4, (2013).

[9] S. Seo, I. Jang, K. Woo, I. Kim, J. S. Kim and S. Maeng, "HPMR: Prefetching and Pre-shuffling in Shared MapReduce Computation Environment", IEEE International Conference on Cluster Computing and Workshops, (2009) August 31-September 4: New Orleans, USA.

[10] S. Khalil, S. A. Salem, S. Nassar and E. M. Saad, "Mapreduce Performance in Heterogeneous Environments: A Review", International Journal of Scientific & Engineering Research, vol. 4, no. 4, (2013).

[11] J. Xie, S. Yin, X. Ruan, Z. Ding, Y. Tian, J. Majors, A. Manzanares and X. Qin, "Improving MapReduce Performance through Data Placement in Heterogeneous Hadoop Clusters", IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW), (2010) April 19-23: Arlanta, USA.

[12] Z. Tang, J. Q. Zhou, K. L. Li and R. X. Li, "MTSD: A task scheduling algorithm for MapReduce base on deadline constraints", IEEE International Symposium on Parallel and Distributed Processing Workshops and PhD Forum (IPDPSW), (2012) May 21-25: Shanghai, China.

[13] M. Zaharia, D. Borthakur, J. Sen Sarma, K. Elmeleegy, S. Shenker and I. Stoica, "Delay Scheduling: A Simple Technique for Achieving Locality and Fairness in Cluster Scheduling", Proceedings of the 5th European conference on Computer systems, (2010) April 13-16: Paris, France.

[14] X. Zhang, Z. Zhong, S. Feng and B. Tu, "Improving Data Locality of MapReduce by Scheduling in Homogeneous Computing Environments", IEEE 9th International Symposium on Parallel and Distributed Processing with Applications (ISPA), (2011) May 26-28: Busan, Korea.

[15] C. Abad, Y. Lu and R. Campbell, "DARE: Adaptive Data Replication for Efficient Cluster Scheduling", IEEE International Conference on Cluster Computing (CLUSTER), (2011) September 26-30: Austin, USA.