INTERNATIONAL JOURNAL OF RESEARCH CULTURE SOCIETY ISSN: 2456-6683 Volume - 2, Issue - 4, Apr – 2018
UGC Approved Monthly, Peer-Reviewed, Refereed, Indexed Journal Impact Factor: 3.449 Publication Date: 30/04/2018

ENHANCE PERFORMANCE OF HADOOP BY TUNING MAPREDUCE JOB PARAMETERS

Raj Kumar Badode[1], Ajay Phulre[2]
[1]Student of CSE, [2]Prof. & Head of CSE Dept.
[1, 2] Shri Balaji Institute of Tech. & Management, Betul
Email -[1] rajbadode.k@gmail.com [2] aphulre@gmail.com

Abstract: Hadoop has become a key component of big data, and gained more and more support. Since users have recognized the enormous potential of Hadoop, some of them are working to develop and optimize the existing technologies
to supplement Hadoop when using it. Hadoop MapReduce is a popular framework for distributed storage and processing of large datasets and is used for big data analytics. It has various configuration parameters which play an important role in deciding the performance i.e., the execution time of a given big data processing job. Default values of these parameters do not result in good performance and therefore it is important to tune them. In this paper we proposed that there are more than 100 configuration parameters in mapreduce framework. In this we preposed a prefetch mechanism approach by tuning the mapreduce framework and check the performance and compare it with bydefault mapreduce configured parameters.

Key Words: *Big data, Hadoop, Mapreduce, performance, prefetching mechanism, tuning.*

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.

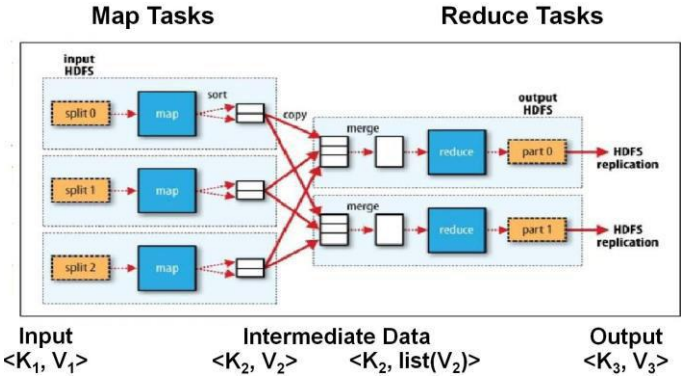


Figure 1: Execution of a MapReduce job.

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 spilling.
4. Spill: Sorting, using the combine function if any, performing compression if specified, and finally writing to local disk to create file spills.
5. 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 decompressing if needed. Partial merging may also occur during this phase.
2. Merge: Merging the sorted fragments from the different mappers to form the input to the reduce function.
3. Reduce: Executing the user-defined reduce function to produce the final output data.
4. 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:

According to [1], Processing and analyzing bigdata is a tuff task for now a days. There is inherent difficulty in tuning the parameters due to two important reasons - first, the parameter search space is large and second, there are cross-parameter interactions. We are in the era of big data and huge volumes of data are generated in various domains like social media, financial markets, transportation etc. Quick analysis of such huge quantities of unstructured data is a key requirement for achieving success in many of these domains. Performing distributed sorting, extracting hidden patterns and unknown correlations and other useful information is critical for making better decisions. To efficiently analyze large volumes of data, there is a need for parallel and distributed processing/programming methodologies.

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[10]
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[12] based on their computing speed.
1) Data placement in Heterogeneous- Two algorithms are implemented and incorporated into Hadoop HDFS.[11] 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] [8]

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[13]. 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] [9]

Multiple speculative execution strategies are improving the performance in Heterogeneous as well as Homogeneous[14]. 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.[15]

3. 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.

4. 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.

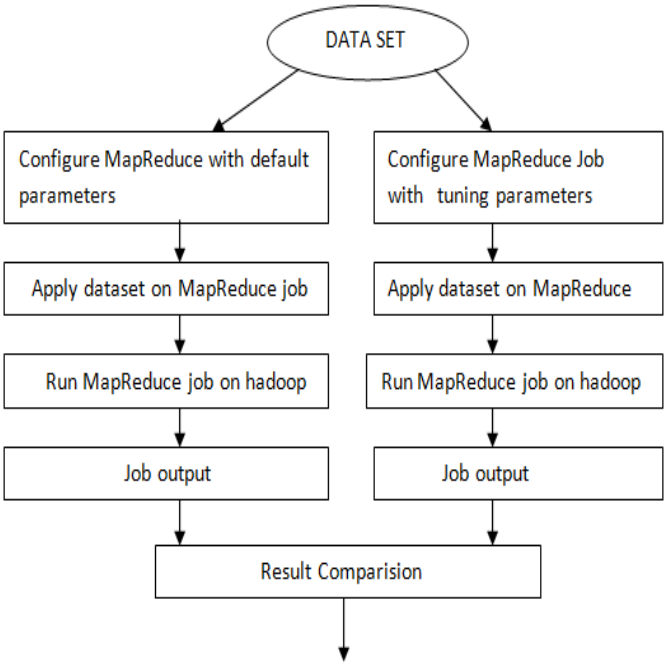
3. MapReduce Job

   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.

5. 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 with default parameters or with tuning parameters 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.



ANALYSIS STEPS

6. 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 word count application in a overall file , the file size is around 560MB, In these we can run mapreduce job on a default parameters configuration and than runs the same application on a tuned mapreduce configured framework.

After developing we can launch the existing.jar file on default configuration parameters shown in figure 2.

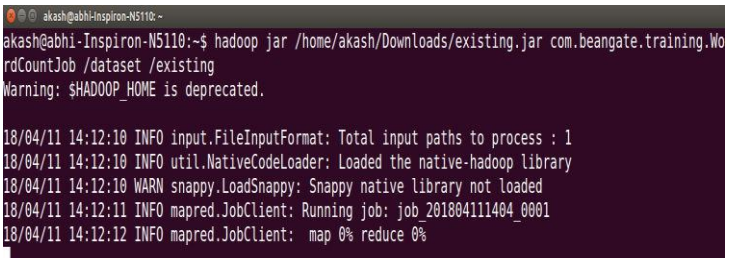
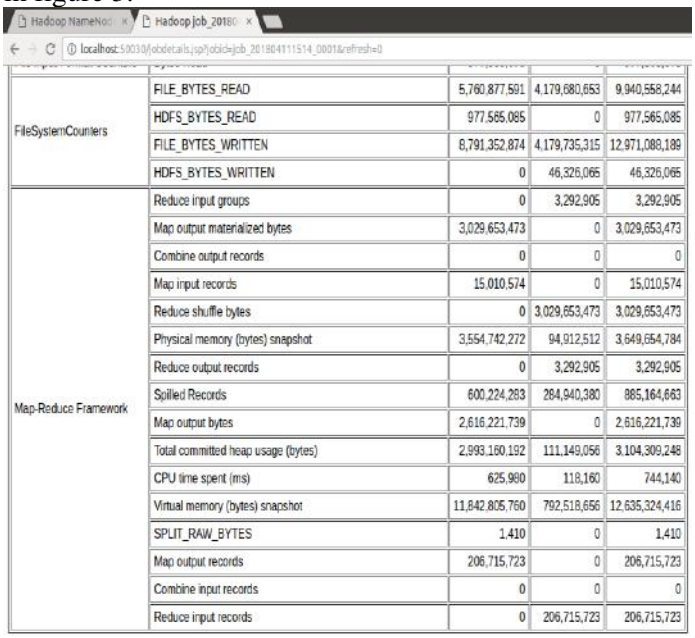


Figure 2. launching job on default configuration

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.



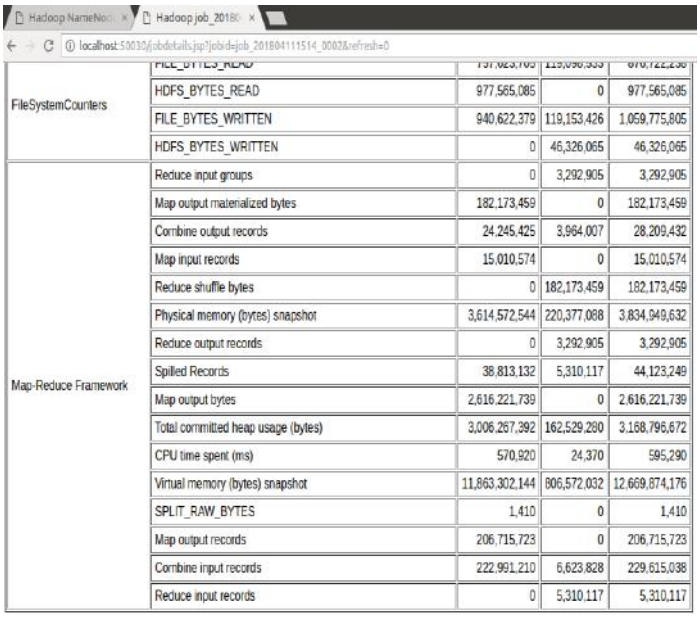
FileSystemCounters	FILE_BYTES_READ	5,760,877,591	4,179,680,653	9,940,558,244
	HDFS_BYTES_READ	977,565,085	0	977,565,085
	FILE_BYTES_WRITTEN	8,791,352,874	4,179,735,315	12,971,088,189
	HDFS_BYTES_WRITTEN	0	46,326,065	46,326,065
Map-Reduce Framework	Reduce input groups	0	3,292,905	3,292,905
	Map output materialized bytes	3,029,653,473	0	3,029,653,473
	Combine output records	0	0	0
	Map input records	15,010,574	0	15,010,574
	Reduce shuffle bytes	0	3,029,653,473	3,029,653,473
	Physical memory (bytes) snapshot	3,554,742,272	94,912,512	3,649,654,784
	Reduce output records	0	3,292,905	3,292,905
	Spilled Records	600,224,263	284,940,380	885,164,663
	Map output bytes	2,616,221,739	0	2,616,221,739
	Total committed heap usage (bytes)	2,993,160,192	111,149,056	3,104,309,248
	CPU time spent (ms)	625,980	118,160	744,140
	Virtual memory (bytes) snapshot	11,842,805,760	792,518,656	12,635,324,416
	SPLIT_RAW_BYTES	1,410	0	1,410
	Map output records	206,715,723	0	206,715,723
	Combine input records	0	0	0
	Reduce input records	0	206,715,723	206,715,723

Figure 3. time taken by existing mapreduce job

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

Figure 4.launching job on tuned mapreduce configuration

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



FileSystemCounters	HDFS_BYTES_READ	977,565,085	0	977,565,085
	FILE_BYTES_WRITTEN	940,622,379	119,153,426	1,059,775,805
	HDFS_BYTES_WRITTEN	0	46,326,065	46,326,065
Map-Reduce Framework	Reduce input groups	0	3,292,905	3,292,905
	Map output materialized bytes	182,173,459	0	182,173,459
	Combine output records	24,245,425	3,964,007	28,209,432
	Map input records	15,010,574	0	15,010,574
	Reduce shuffle bytes	0	182,173,459	182,173,459
	Physical memory (bytes) snapshot	3,614,572,544	220,377,088	3,834,949,632
	Reduce output records	0	3,292,905	3,292,905
	Spilled Records	38,813,132	5,310,117	44,123,249
	Map output bytes	2,616,221,739	0	2,616,221,739
	Total committed heap usage (bytes)	3,006,267,392	162,529,280	3,168,796,672
	CPU time spent (ms)	570,920	24,370	595,290
	Virtual memory (bytes) snapshot	11,863,302,144	806,572,032	12,669,874,176
	SPLIT_RAW_BYTES	1,410	0	1,410
	Map output records	206,715,723	0	206,715,723
	Combine input records	222,991,210	6,623,828	229,615,038
	Reduce input records	0	5,310,117	5,310,117

Figure5. time taken by proposed mapreduce job

Time taken by (in sec)	Existing.jar	Proposed.jar
DATASET	744.14	595.29

Figure 6. Execution time taken by existing and proposed system

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.

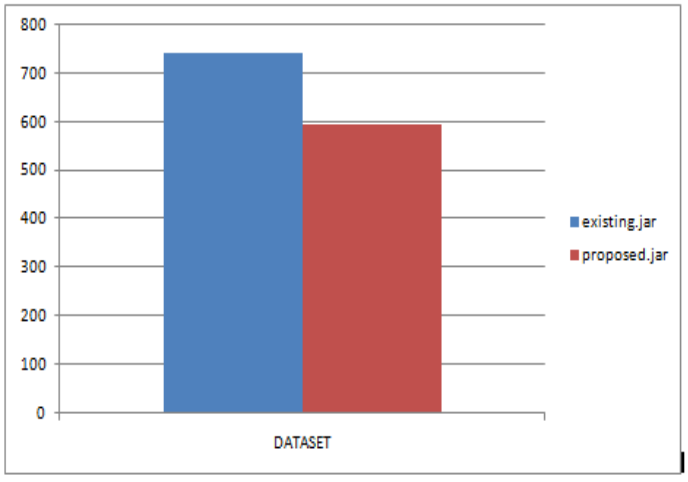


Figure 7. Graph representation of execution time taken

The proposed techniqus shows better execution result as compared to existing default configuration because in proposed we can perform all the operation on map side so the reducer input records is less thats by the network cost and the reduce load is optimize in proposed. Table 2 shows the number of records process by mapper and reducer.

Record process by	Mapper Input	Mapper output	Reducer input	Reducer output
Existing technique	15010574	206715723	206715723	3292905
Proposed technique	15010574	206715723	5310117	3292905

Table 2. Number of records process by mapper & reducer

7. 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 preposed a prefetch mechanism approach by tuning the mapreduce framework and check the performance and compare it with bydefault mapreduce configured parameters.

REFERENCES:

1. Ankita Jain, Monika Choudhary, " Analyzing & Optimizing Hadoop Performance" in Big Data Analytics and Computational Intelligence (ICBDAC), in IEEE 2017.
2. 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
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.