International Journal of Pure and Applied Mathematics
Volume 118 No. 18 2018, 2545-2551
ISSN: 1311-8080 (printed version); ISSN: 1314-3395 (on-line version)
url: http://www.ijpam.eu
Special Issue

ijpam.eu

Improving Late Scheduler and Result Analysis in Heterogeneous Environment

Priyanka Kumar
Department of Computer Science and Engineering,
Amrita School of Engineering, Coimbatore,
Amrita Vishwa Vidyapeetham, India.
e-mail: k_priyanka@cb.amrita.edu

Veerakumar S, Srenithi M
Department of Computer Science and Engineering,
Amrita School of Engineering, Coimbatore,
Amrita Vishwa Vidyapeetham, India.
e-mail: cb.en.p2cse16021@cb.students.amrita.edu,
cb.en.p2cse16018@cb.students.amrita.edu

***Abstract*- The word Heterogeneity signifies diversity and Heterogeneity of Hadoop cluster exists as nodes have different configuration and they are distributed in various geographical sites. In distributed environments, homogeneity and local access of data are not full filled. Therefore, performance reduction will happen in default Hadoop schedulers where the cluster nodes are heterogeneous. Because of highly robust nature of LATE Scheduler, it can be used to improve response time in heterogeneous environment by varying the number of Mappers and Reducers for each node. In this paper we have used variable number of Reducers and Mappers to improve the performance of Hadoop heterogeneous cluster.**

Keywords—speculative execution; improved LATE; variable Mappers; variable Reducers, heterogeneous.

I. INTRODUCTION

Cloud Computing has some special characteristics like tractability, manageability and scalability. The advantage of cloud gives rise to emerging areas like Big Data more popular in recent years and the above said features makes processing of Big Data more powerful, but the limitations over storage makes the processing challenging on cloud[1].

Then in literature, MapReduce feature of Hadoop came into the picture which will take care of failures by default making it fault tolerant. If a node fails, Hadoop again runs its tasks on a different machine. In cloud cluster if available node performs slow compared to the other nodes which are named as Straggler nodes, then nodes are identified by their progress score and Hadoop runs a speculative copy on straggler on different node to process faster [5]. Straggler in Hadoop arises due to fault in hardware or misconfiguration in software. Hence without speculation, nodes performance are poor. Speculative execution of straggler nodes increases performance [3].

Hence, the question is how speculative execution can be done to maximize performance [17]? Hadoop Scheduler performs speculative execution based on few heuristics determined by the kernel by comparing each of its execution time with the average time [5]. It works for homogeneous case, which can lead to severe performance degradation when it's heterogeneous [17].

Homogeneity assumptions of Hadoop scheduling leads to more speculative execution in heterogeneous cases, resulting in decrease in the behavior of distributed systems [16]. In extreme cases, 80% of tasks were speculatively executed but in reality, speculation is a complex issue for several reasons.

(1) Speculative jobs are not free – they compete for certain resources, like network, with other running tasks.
(2) Selection of a node to return the slow task.
(3) In heterogeneous cases, it is hard to identify the slow running nodes. Also before it affects the response time of the system, speculation has to be executed.

A. Speculative Execution in Scheduling

A Hadoop scheduler provides the required rendition for their users and providers. Schedulers assign available resources to the processes that request for. But, there are some problems in Hadoop which affects scheduler's rendition, heterogeneity, jobs and resources [1]. Most of the Hadoop schedulers do not consider these which leads to poor rendition of scheduler.

Installation of Hadoop system on an intranet is a common practice. This leads to change in jobs and resources significantly in a given time. Investigation results in Hadoop native scheduling algorithms such as Fair, FIFO, and Capacity Schedulers does not give the ideal rendition in average completion time [1].

This paper intends an improved LATE which decreases execution time nearly half of the other scheduling algorithms. LATE works by running the fast nodes, and preventing thrashing by covering the speculative tasks [6]. We show that LATE improves the time to response of MapReduce jobs by a factor of two in large clusters by using variable number of mappers and reducers.

B. Hadoop Introduction and Architecture

Hadoop is an open source Programming framework for efficient distributed execution. It deliberately hides the details of parallel processing like data distribution to processing nodes, restarting failed sub jobs, and consolidation of results after the computation in nodes. Hadoop allows people to write parallel programs which focus on the computation problem, in place of parallelization issues.

- **Hadoop Distributed File System (HDFS):** A file system that store large amount of data with high speed access to data on clusters [7]
- **Name Node:** The Name node takes care of the mapping of namespaces in the Hadoop system and also the mapping of files to the Data node. The client which wants to access the data node first has to obtain the address of the files from the name node closest to the client. Here replication factor by default is three. Hence the Name node gives three replicas of the files requested by the client. Then the client writes the data on those nodes sequentially, and the snapshot of the system is stored as an image in the file system forming a checkpoint for later verification [7]. It saves all modifications to the log file and also it is distributed to the other servers for security purposes.
- **Data Nodes:** Data node starts execution and contacts to the name node and performs connection establishment [17]. The reason is to verify the address of the namespaces and version of the data node. If it does not match, the name node the data node automatically shuts down due to misconfiguration. After the handshake the Data Node connects with the Name Node [11].
- **Map-Reduce Architecture (Hadoop Map Reduce):** a JAVA based software framework for programming implementation. Map Reduce framework splits the supplied job into various blocks of chunks which Map the tasks process parallel. The corresponding outputs of the map tasks are internally sorted by the MapReduce framework itself temporarily and are supplied to the reducer nodes for further processing. All the internal results are stored in HDFS.
- **Hadoop Systems Scalability and Heterogeneity:**

  Hadoop heterogeneity assumptions are divided into three categories: workload, cluster and users.

  Workload: The tasks normally submitted to the Hadoop environment are heterogeneous because those tasks vary in their number, data, arrival rate, execution and the computation time [1]. This makes all the jobs different from one another; hence the workload varies from one node to another.

  Clusters: The computing clusters vary in their data storage and capability for data processing.

  Users: Each Hadoop user would be assigned a different priority to access the data and different requirements, also the workload varies from one user to another. The resources in Hadoop generally vary based on time having load throughout the day time with less workload at evening. When the system is not scalable with respect to resources, there may be overloading or underutilization of resources resulting in poor performance. But Hadoop scalable feature makes it best to scale up or down according to the current data requirements by considering the number and complexity of each nodes and also each job [1]. Hence as a result, the optimized performance from Hadoop can be achieved from selecting a proper scalable scheduler to run the jobs under the non-homogenous situations.

II. Native Scheduling in Hadoop

During normal execution data nodes send heartbeat messages to the name node. Three seconds is the default heart beat interval. If the data node fails to send heart beat message to name node, then name node considers that data node has failed. The name node allocates new replica creation of blocks over Data Nodes. These all processes has done with Job tracker and Task tracker [17] [11].

*Job Tracker***:** Each cluster has only one Job Tracker which is used for submitting and Tracking Map Reduce tasks in Hadoop [17]. So Map Reduce is a failure service and hence if it fails all running jobs is halted. The slaves are configured to the node location of the Job Tracker and perform tasks as instructed by the Job Tracker.

Task Tracker: Each slave node has one Task Tracker which tracks task instances and notifies the Job Tracker regarding the status. It is the Ultimate functionality of the task tracker to control the execution of map task and reduce task inside the Hadoop clusters environment. The group of Task tracker sends heartbeats messages to the job tracker periodically. When there is no message from the Task tracker, the master node

concludes that the particular slave node has failed or crashed and makes further arrangements to rerun the supplied task on other nodes [12].

A. Native Scheduling algorithms

*FIFO***:** It is the Hadoop's default scheduler. Based on the jobs arrival time they are ordered in the queue, by not considering the system's heterogeneity. The investigation results of different schedulers depicts schedulers like FAIR, FIFO can cause severe rendition degradation chiefly in multiuser environments [1].

Fair Sharing: To overcome the drawbacks of FIFO algorithm, Fair sharing is developed. While small jobs are dealt and heterogeneity of users. For each user, this scheduler defines an idle pool and the slots of the pool among other users slots [1].

Capacity scheduler: Capacity scheduler is developed by Yahoo, is used when multiple organizations want to share the large cluster with minimum capacity and sharing huge capacity among the users. Instead of pools unlike in fair sharing, several queues are created; each queue is configured with MapReduce slots. The queue has priority with FIFO and achieved in capacity scheduler. High priority jobs must access resources, compared to the low priority jobs. Scheduled tasks are analyzed by memory consumption of each task; the scheduler has the ability to control and allocate memory based on the available resources [13]. Based on the needs resources are shared, available resources are partitioned within MapReduce cluster among many Organizations.

B. Performance Metrics

The rendition of a Hadoop system is measured as follows [15]:

- *Average Completion Time:* It *is* the mean time to complete the jobs that are scheduled.
- *Dissatisfaction:* It measures scheduling algorithm's success in full filling the maximum user's requirement [1].
- *Fairness:* It measures fairness of algorithm in providing users with resources.
- *Locality:* The fraction of tasks which are locally running over the resource that embeds their data in it. Large clusters are handled by Hadoop system. Accordingly, big data is handled by mappers and reducers can have considerable cost of communication on neglecting of locality of data.
- *Scheduling Time:* It is the aggregate time consumed for incoming job scheduling. This metric calculates each of Hadoop scheduler overhead [1].

In native scheduling policy if there exists an empty slot in a node, Hadoop scheduler chooses a task as given below:-
(1) Highest priority is given to failed tasks. It also detects bug when a task fails and stop the job [9].
(2) Data local tasks to node cluster are chosen initially by considering the non-running tasks, [9].

III. Assumptions to be held in Hadoop's scheduler

Hadoop's inbuilt scheduler makes several implicit assumptions [17]:

1. All the Hadoop nodes process the jobs at the same amount of time.
2. The submitted jobs progresses through the scheduler at a constant rate.
3. Instead of having an idle slot, it is better to run the speculative copy of another node which doesn't increase the computation cost [8].
4. Progress Score for each task is an indicator of its total work processed till that instance.
5. Slow processing Nodes classified as Straggler nodes based on the Progress score value
6. Tasks in the same division do roughly an equal amount of work [8].

An assumption 1 and 2 breaks down in a distributed data center due to heterogeneity. Assumptions 3, 4 and 5 breaks down in a homogeneous and may cause Hadoop to perform poorly. Assumption 6 is inherent in the Map Reduce scheduler. Tasks in Map Reduce should be small, otherwise a single large task slows down the entire job.

Data locality is a decision argument for the Map Reducer performance. Here, the algorithm that was developed to improve data locality management in a heterogeneous Hadoop cluster has following features.

A. Heterogeneity

The first 2 assumptions are about homogeneous. Hadoop assumes that any slow node is faulty. In a non-virtualized data center, there may be multiple layers of hardware. In a virtualized data center where many virtual machines run on each physical host, may cause heterogeneity. For resolving this in heterogeneous environments we have different approaches to increase the scheduler performance. The below mentioned algorithms tries to improve the execution time of the process in a heterogeneous environment. Also this improved feature of algorithms are divides into two following categories [17]:

- Fault Tolerance Algorithms.
- Data Locality Algorithm.

B. Placement of Data in Existing Heterogeneous Clusters

Data placement strategy is useful for a homogeneous environment which having same amount of computing and disk capacity [17]. In heterogeneous Hadoop nodes, a faster node finishes execution before the slow performing nodes. The approach used here is after the faster node finishes execution on the local disk, it is also assigned the additional task of processing unprocessed data on the slow nodes also. Hence the above transfer mechanism increases the network overload of the Hadoop scheduler if the data transfer is huge [2]. Hence to avoid this data transfer between the nodes have to be minimized for optimal execution under heterogeneous environment.

Hence, the above problem can be solved in Hadoop by the application of Data Placement algorithms where the data on the Hadoop is distributed and stored on multiple nodes inside the cluster in accordance with their computing power [2]. This data placement algorithm can be implemented in different ways. One way is to distribute the data according to the computing power of each node. Second way is to reduce skewness issue by reorganizing the entire data.

C. Initial Placement of Data

This algorithm works by dividing the whole input job file into even -sized partitions by the large data distribution server. The round robin scheduling approach is followed for data distribution by the server based on their computing powers [2]. If computing ratio is less, indicates the high speed node which can execute more number of partitions, while a larger value indicates the slow processing node which can only execute lesser number of partitions [2].

D. Redistribution of Hadoop Data

The partitioned input job file by the Data placement algorithm may be redistributed under Hadoop environment due to the following reasons.

1. The new partitioned data may be added to the end of the existing file.
2. While doing division, the data blocks are removed from the input job file.
3. The existing Hadoop cluster is flooded with large amount of new data making the computation complex. [2]

E. Data Locality Task Scheduling Method for Heterogeneous domain

Here, work is built over the method to upgrade locality of data to Map reduce over homogeneous environments. Procedure assumes that processing tasks of all nodes have alike speed when using node to issue the request. Job's input data is stored on the node and procedure reserves the node's task [4].

The goal is to construct a tradeoff among the delay time and communication time when a task is scheduled to a node to reach the execution time of optimal task. On receiving node request, the procedure first schedules the task on the node which requests [4]. Then this method calculates the delay time and communication time of the chosen job. If the delay time is lesser than the communication time, the procedure keeps the node tasks which having old data input else, the task schedules to the node that requests.

F. Fault Tolerance Algorithms

A major advantage of Map reduce is automatically organizes the failures and hides from the users with complexity of the fault tolerance. Hadoop rendition is related to scheduler task, which thinks that homogenous nature cluster nodes and linearly progresses. Hadoop's native scheduler causes severe rendition degradation in heterogeneous domain because the underlying assumptions no longer exist [14].

IV. LATE Scheduler and its Limitations

LATE stands for Longest Approximate Time to End. . LATE estimates the task's finish time based on the progress score given by Hadoop [6] [17].

Finish Time= (1-Progress Score) / Progress rate **[6]**

When the node inside the Hadoop cluster has an empty slot for processing, Hadoop scheduler selects those nodes based on three categories: First, failed tasks are chosen by assigning them highest priority. Second, the non-running nodes are considered for re-execution, by assigning high probability for the map tasks. Third, the speculative nodes are given priority. The Execution status of a node is calculated by progress score which ranges from 0 to 1 for assigning speculative tasks. For Map, Progress score is simply the part of input data which is already read by the node. But for Reduce, it has three phases inside it where each having ⅓ probability.

Fixed threshold is assigned prior, hence those nodes having their average progress score falling below this limit is labeled as a straggler node. LATE always speculatively runs the task which will finish latest in the future.

Normally, the execution of all the nodes is at constant pace to increase the scheduler performance, Speculation have to be carried out on the fast execution nodes and not on slow speed nodes. It is done not based on threshold values for slow nodes. This idea increases the performance ratio compared to the older approach of speculating slow nodes falling below the threshold limit [6].

The next method to increase the performance of heterogeneous cluster is to speculate multiple copies for a node, but this also has a limitation of resource wastage. Finally to implement, the following facts are considered regarding speculative tasks cost resources,

1. The total number of Tasks speculated under a Hadoop cluster which is denoted by Speculative Cap
2. To determine the straggler nodes, slower task threshold is used.

A. LATE Algorithm [1]

In late algorithm, when a new task is assigned to a node and when they are lesser than running speculative tasks then [3],

- Incoming request is disregarded if the progress of node is below Slow Node Threshold.
- Rank the currently processing tasks that aren't being calculated time left speculation [17].
- Top-ranked task copy is launched with progress rate under Slow Task Threshold as Hadoop's scheduler [17].

B. Shortcomings of LATE Algorithm [1]

There exists some disadvantages which reduces the performance of heterogeneous Hadoop cluster,

- PR computation is done statistically in LATE.
- For Map and Reduce tasks, LATE does not have a different computing technique.
- Many Static parameters are supplied as inputs.

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>

<!-- Put site-specific property overrides in this file. -->

<configuration>
<property>
<name>mapred.job.tracker</name>
<value>localhost:54311</value>
<value>mapred.map.tasks=6</value>
<description>The host and port that the MapReduce job tracker runs
at. If "local", then jobs are run in-process as a single map and reduce task.</description>
</property>
</configuration>
```

V. Our Contribution: Improved LATE Scheduler Algorithm

Improved LATE algorithm are proposed to increase the efficiency and performance of the LATE scheduler in heterogeneous environments.

- The algorithm, searches for slow straggler nodes in both Map and Reduce.
- Backup copy of the tasks is collected.
- Progress score and Progress Rate is calculated.
- Updating information on each straggler node for speculative execution.
- Mappers can be increased by assigning variable number of jobs to each node.
- Number of Reducers can be increased by the formula,

Reducer count=0.95*(nodes*mapred.tasks.maximum)

Here we improve the performance of heterogeneous cluster nodes by increasing variable number of Reducers and variable number of Mappers. In LATE, the number of reducers are varied programmatically which reduces the execution time (ms). And also number of Mappers are varied programmatically to improve the execution time of heterogeneous cluster nodes in Hadoop. As shown in figure 2 number of reducers and number of mappers are varied programmatically.

VI. Our Contribution: Results Execution and Analysis for Improved Late Algorithm

The Hadoop scheduler code is modified by inclusion of Improved LATE Algorithm. Multiple combinations of Mappers and Reducers are tested for the efficiency of the algorithm and the values are tabulated. The execution time is found to be decrease with variable number of Reducers and also assigning variable number of jobs to the Mapper nodes. The required numbers of Mapper jobs are set by modifying the mapred.conf file and the number of Reducers can be calculated from the above formula and given as input at execution time to the scheduler. The maximum capacity of Mappers and Reducers is limited to the nodes count in the Hadoop environment.

Fig. 1. Assigning Variable Number of Mapper Tasks in mapred.conf file

```
public static void main(String[] args) throws Exception {
  Configuration conf = new Configuration();

  Job job = Job.getInstance(conf);

  job.setNumReduceTasks(2);
  job.setJarByClass(WordCount.class);
  job.setMapperClass(TokenizerMapper.class);
  job.setCombinerClass(IntSumReducer.class);
  job.setReducerClass(IntSumReducer.class);
  job.setOutputKeyClass(Text.class);
  job.setOutputValueClass(IntWritable.class);
  FileInputFormat.addInputPath(job, new Path(args[0]));
  FileOutputFormat.setOutputPath(job, new Path(args[1]));
  System.exit(job.waitForCompletion(true) ? 0 : 1);
 }
}
```

```
17/04/29 03:22:16 INFO mapred.JobClient:   Map-Reduce Framework
17/04/29 03:22:16 INFO mapred.JobClient:     Spilled Records=20
17/04/29 03:22:16 INFO mapred.JobClient:     Map output materialized bytes=173
17/04/29 03:22:16 INFO mapred.JobClient:     Reduce input records=10
17/04/29 03:22:16 INFO mapred.JobClient:     Virtual memory (bytes) snapshot=13235822592
17/04/29 03:22:16 INFO mapred.JobClient:     Map input records=12
17/04/29 03:22:16 INFO mapred.JobClient:     SPLIT_RAW_BYTES=120
17/04/29 03:22:16 INFO mapred.JobClient:     Map output bytes=135
17/04/29 03:22:16 INFO mapred.JobClient:     Reduce shuffle bytes=173
17/04/29 03:22:16 INFO mapred.JobClient:     Physical memory (bytes) snapshot=799682560
17/04/29 03:22:16 INFO mapred.JobClient:     Reduce input groups=10
17/04/29 03:22:16 INFO mapred.JobClient:     Combine output records=10
17/04/29 03:22:16 INFO mapred.JobClient:     Reduce output records=10
17/04/29 03:22:16 INFO mapred.JobClient:     Map output records=12
17/04/29 03:22:16 INFO mapred.JobClient:     Combine input records=12
17/04/29 03:22:16 INFO mapred.JobClient:     CPU time spent (ms)=9780
17/04/29 03:22:16 INFO mapred.JobClient:     Total committed heap usage (bytes)=551550976
```

Fig. 2. Experimental run with Mapper jobs= 20 and Reducer count=2

```
public static void main(String[] args) throws Exception {
  Configuration conf = new Configuration();

  Job job = Job.getInstance(conf);

  job.setNumReduceTasks(2);
  job.setJarByClass(WordCount.class);
  job.setMapperClass(TokenizerMapper.class);
  job.setCombinerClass(IntSumReducer.class);
  job.setReducerClass(IntSumReducer.class);
  job.setOutputKeyClass(Text.class);
  job.setOutputValueClass(IntWritable.class);
  FileInputFormat.addInputPath(job, new Path(args[0]));
  FileOutputFormat.setOutputPath(job, new Path(args[1]));
  System.exit(job.waitForCompletion(true) ? 0 : 1);
 }
}
```

```
17/04/29 03:22:16 INFO mapred.JobClient:   Map-Reduce Framework
17/04/29 03:22:16 INFO mapred.JobClient:     Spilled Records=20
17/04/29 03:22:16 INFO mapred.JobClient:     Map output materialized bytes=173
17/04/29 03:22:16 INFO mapred.JobClient:     Reduce input records=10
17/04/29 03:22:16 INFO mapred.JobClient:     Virtual memory (bytes) snapshot=13235822592
17/04/29 03:22:16 INFO mapred.JobClient:     Map input records=12
17/04/29 03:22:16 INFO mapred.JobClient:     SPLIT_RAW_BYTES=120
17/04/29 03:22:16 INFO mapred.JobClient:     Map output bytes=135
17/04/29 03:22:16 INFO mapred.JobClient:     Reduce shuffle bytes=173
17/04/29 03:22:16 INFO mapred.JobClient:     Physical memory (bytes) snapshot=799682560
17/04/29 03:22:16 INFO mapred.JobClient:     Reduce input groups=10
17/04/29 03:22:16 INFO mapred.JobClient:     Combine output records=10
17/04/29 03:22:16 INFO mapred.JobClient:     Reduce output records=10
17/04/29 03:22:16 INFO mapred.JobClient:     Map output records=12
17/04/29 03:22:16 INFO mapred.JobClient:     Combine input records=12
17/04/29 03:22:16 INFO mapred.JobClient:     CPU time spent (ms)=9780
17/04/29 03:22:16 INFO mapred.JobClient:     Total committed heap usage (bytes)=551550976
```

Fig. 3. Experimental run with Mapper jobs= 10 and Reducer count=7

Number of mappers	Number of Reducers	Execution time(ms)
10	7	2170
10	6	3560
10	5	5250
20	4	7060
20	3	7990
20	2	9780

Table 1. Estimated Execution Time for Improved LATE scheduler

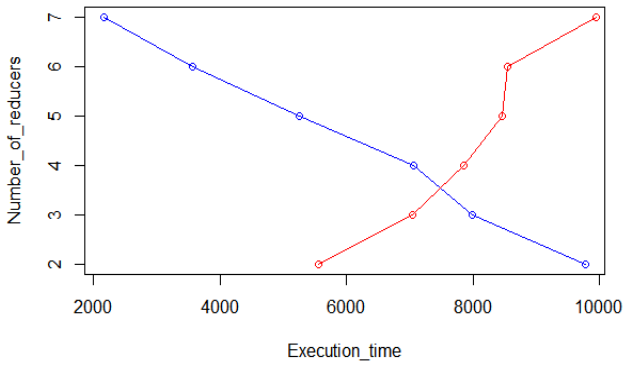

Fig. 5. Graph for Comparison of Improved LATE (Blue) and LATE Scheduler (Red).

From figure 5 it can be inferred from the graph that the execution Time for Improved LATE scheduler decreases for increasing values of **number of Reducers(R)** whereas for LATE scheduler the **execution time (T)** slightly increases for larger values of **R**.

VII. Conclusion

From the Experimental results and the tables, it can be concluded that the Improved LATE algorithm works well for Hadoop in heterogeneous environments, and also the execution time is decreased by the inclusion of variable number of reducer count and the task count assigned to each node in a multi-node environment. The results obtained shows the increased efficiency of Improved LATE scheduler over Hadoop native scheduling algorithm.

VIII. Future Work

In future, The LATE algorithm can be further tested for larger values of Reducers and also extending the functionality of the Hadoop nodes by performing both Map and Reduce operations under heterogeneous environments.

References

[1] Aysan Rasooli, Douglas G. Down. "A Hybrid Scheduling Approach for Scalable Heterogeneous Hadoop Systems", 2012 SC Companion: High Performance Computing,Networking Storage and Analysis, 2012.

[2] Jiong Xie. "Improving MapReduce performance through data placement in heterogeneous Hadoop clusters", 2010 IEEE International Symposium on Parallel &Distributed Processing Workshops and Phd Forum (IPDPSW), 04/2010.

[3] Advances in Intelligent Systems and Computing, 2016.

[4] Xiaohong Zhang. "An effective data locality aware task scheduling method for MapReduce framework in heterogeneous environments", 2011 International Conference on Cloud and Service Computing, 12/2011.

[5] Mohanapriya, S., and P. Natesan. "A micropartitioning technique for massive data analysis using MapReduce", International Conference on Information Communication and Embedded Systems (ICICES2014), 2014.

[6] Hsiao, J. H., and S. J. Kao. "A usage-aware scheduler for improving MapReduce performance in heterogeneous environments", 2014 International Conference on Information Science Electronics and Electrical Engineering, 2014.

[7] Konstantin Shvachko. "The Hadoop Distributed File System", 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST), 05/2010.

[8] Xu, Guanghui, Feng Xu, and Hongxu Ma. "Deploying and researching Hadoop in virtual machines", 2012 IEEE International Conference on Automation and Logistics,2012.

[9] Caruana, Godwin(Li, M). "MapReduce based RDF assisted distributed SVM for high throughput spam filtering", Brunel University School of Engineering and Design PhD Theses, 2013.

[10] Thakkar, S., Patel, S.: Scheduling in Big Data Heterogeneous Distributed System Using Hadoop: Proceedings of International Conference on ICT for Sustainable Development, Advances in Intelligent Systems and Computing (2016).

[11] Thirumala Rao, B., Sridevi, N. V., KrishnaReddy, V., Reddy, L. S. S. (2011). Performance issues of heterogeneous Hadoop clusters in cloud computing. Global Journal of Computer Science and Technology, 11(8), Version 1.0 May 2011.

[12] Dean, J., & Ghemawat, S. (2008). MapReduce: Simplified data processing on large clusters Communications of theACM, 51(1), 107–113.

[13] Zaharia, M., Konwinski, A., Joseph, A., Zatz, Y. & Stoica, I. (2008). Improving map reduce performance in heterogeneous environments. In OSDI'08: 8th USENIX Symposium on Operating Systems Design and Implementation, October 2008.

[14] Konwinski, A. Improving map reduce performance in heterogeneous environments Technical Report No. UCB/EECS-2009-183.

[15] Thakkar, S., Patel, S.: Improving Map Reduce Performance in Heterogeneous Distributed System using HDFS Environment-A Review: International Journal on Recent and Innovation Trends in Computing and Communication, March 2015.

[16] Jeyakumar, G. and ShunmugaVelayutham, C. "Distributed Heterogeneous Mixing of Differential and Dynamic Differential Evolution Variants for Unconstrained Global Optimization", Soft Computing – Springer, Volume 18, Issue 10 (2014), Page 1949-1965, October -2014.

[17] [Online] www.ijritcc.org