IOSR Journal of Computer Engineering (IOSR-JCE)
e-ISSN: 2278-0661,p-ISSN: 2278-8727, Volume 17, Issue 6, Ver. V (Nov – Dec. 2015), PP 64-75
www.iosrjournals.org

Map-Reduce Synchronized and Comparative Queue Capacity Scheduler in Hadoop for Extensive Data

Sukhmani Goraya[1], Vikas Khullar[2]
Department of Computer Science and EngineeringCT Institute of Engineering, Management and Technology, Shahpur, Jalandhar, India.

Abstract: *Map-Reduce is drawing attention of both industrial and academic for processing of big data. In this paper, we have mainly focused on core scheduler of Hadoop i.e. Capacity Scheduler. The scheduler assigns tasks to the resources. Capacity Scheduler shares various resources in a cluster between concurrent jobs. We have performed various experiments based on Map-Reduce benchmark applications i.e. TestDFSIO, Pi, Word Count for analyzing and performance enhancement of capacity scheduler in terms of both execution time as well as capacity. We have proposed a method Synchronized and Comparative Queue Capacity Scheduler (SCQ) that makes better resource utilization which reflects enhanced performance in terms of execution time, throughput and average IO rate.*

Keywords: *Map-Reduce, Capacity Scheduler, Fair Scheduler, HDFS, TestDFSIO, Word Count, Pi, Synchronized and Comparative Queue (SCQ)*

I. Introduction

Map-Reduce model has being successfully used by various industries to perform large computations. After being strongly promoted by Google, it has also been implemented by the open source community through the Hadoop [1] project, maintained by the Apache Foundation and supported by Yahoo! and even by Google itself. Since 82% of information is in "unstructured" kind, it should be formatted in a very unique manner that creates it suitable for ulterior analysis and data processing [2]. Hadoop is a platform for creating such data into the structured form which helps in analysis of big data. Hadoop has its origins in Apache Nutch, an open supply net program, itself a locality of the Lucene project [2] [4]. As we know that large data is being generated and for processing such large data in less time, scheduling needs to be done. The role of the scheduler is to schedule Map and Reduce task to minimize the job completion time and maximize resource utilization. The goal of Hadoop is to supply economical and high performance process of massive information applications [5].

A. Overview Of Hadoop

Hadoop consist two components mainly as Hadoop Map-Reduce and HDFS [6] [7].

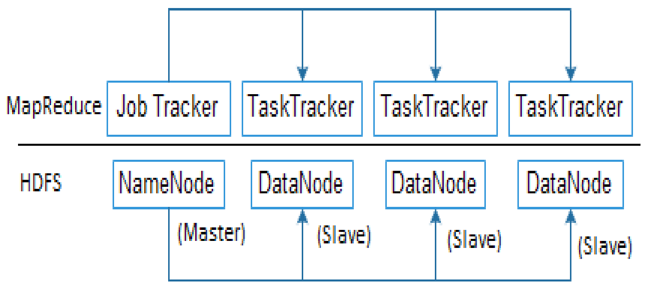

Fig. 1.1: Components of Hadoop (Map-Reduce and HDFS)

Figure 1.1 shows the components of Hadoop described as follow:

HDFS: Hadoop Distributed File System (HDFS) mimicking Google File System (GFS) [8] and Hadoop Map-Reduce. The storage system of Hadoop is HDFS (Hadoop Distributed File System). For processing data in Hadoop, it uses Map-Reduce engine which runs on the top of HDFS as shown in Figure 1. The engine of Hadoop works as master-slave architecture. As we can see in the Figure 1 there is one master node and many slave nodes. The master node has JobTracker and NameNode where as the slave node has TaskTracker and DataNode. The JobTracker keeps the track of all the TaskTracker. The jobs are parallelized automatically to the TaskTracker. The NameNode has all the information about all the DataNodes. Task Scheduling technology [9], one of the key technologies of Hadoop platform, mainly controls the order of task running and the allocation of computing resources, which is directly related with overall performance of the Hadoop platform and system resource utilization. Data Nodes conjointly perform block creation, deletion, and replication upon instruction

from the Name Node [10]. Existing research [11], [12] shows that the performance of Map-Reduce applications depends on the cluster configuration, job configuration settings and input data.
Map-Reduce It has been developed by Google in 2004. In Hadoop, applications consist of map and reduce tasks that operate on data stored on the HDFS file system [3].

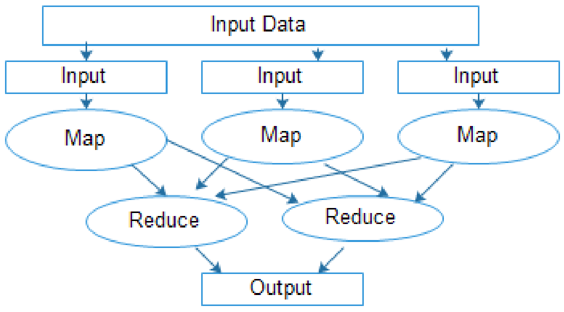

Fig. 1.2: Map-Reduce Framework

In Figure 1.2, job is divided by Hadoop`s Map phase into fixed sized pieces called input split. Reduce phase collects the output and reduces it into a single output. But the main component of Map-Reduce is job scheduler. It schedules Map and Reduce task so that it completes in minimum time. In Hadoop version 1, there is an assumption taken by the scheduler that the resources such as memory, CPU, network etc are of homogenous type. Where as in Hadoop version 2(Hadoop Yarn), resource aware job schedulers to Map-Reduce have been introduced. The paper is organized as; Section II is described as Related Research Work. Section III presents Experimental setup and work done. Section IV has results and discussions. Section V includes the conclusion.

II. Related Research Work

The Job runs on the JobTracker and it plays an important role in deciding where job will be executed in the cluster. Hadoop Yarn provides the resource management and a platform to deliver consistent security and operations. Yarn is created by separating the Map-Reduce capabilities of processing and resource management. Resource Management can help in predicting the behavior of resources on the cluster.
There are three core schedulers of Hadoop FIFO scheduler, Fair Scheduler and Capacity Scheduler.

B. FIFO Scheduler

In the earliest Hadoop Map-Reduce computing architecture, the essential job sort is massive batch jobs that a single user submits the job, thus Hadoop use inventory accounting (First in 1st out) rule in early planning algorithm [13].

FIFO is the default Hadoop scheduler [14]. The scheduling is based on the arrival time; heterogeneity in the system is ignored. The experience from deploying Hadoop in large systems shows basic scheduling algorithms like FIFO can cause severe performance degradation; particularly in systems that share data among multiple users [15]. The Scheduler is single queued and jobs are executed in sequential manner. FIFO is designed only for single type of job so, when multiple users run the job on the cluster, the performance degrades.

C. Fair Scheduler

Fair Sharing is a Hadoop scheduler introduced to address the shortcomings of FIFO, when dealing with small jobs and user heterogeneity [16]. The scheduling allocates resources to pools. The scheduler is responsible for defining the pool for each user. The aim of the scheduler is fair sharing of resources among the user. Resources to jobs such every job gets, on average, an equal share of resources over time [17]. It lets short jobs complete among an inexpensive time whereas not starving long jobs [18]. The objective of honest scheduling rule is to try and do a equal distribution of compute resources among the users/jobs within the system [1]. Fair scheduling gives better performance for large clusters in comparison to FIFO scheduling. Also, it will limit the quantity of coincidental running tasks per pool [19]. Tao et. al. introduced an improved truthful scheduling formula, that takes under consideration job characteristics and information locality, that decreases both information transfer and therefore the execution time of jobs. This algorithm does not consider the duty weight of each and every node, that this is often a crucial disadvantage of it.

D. Capacity Scheduler

Capacity Scheduler [20] originally developed at Yahoo addresses a usage scenario where the number of users is large, and there is a need to ensure a fair allocation of computation resources amongst users. Fair scheduler and capacity scheduling algorithm is very similar but capacity scheduler used queue instead of job pool. Each queue is assigned to associate. The scheduling allocates resources to the pools and there is FIFO scheduling within each pool. Capacity scheduler puts the job into various queues or hierarchy of queues according to the conditions, and allocates bound to system capacity for every queue. It also supports hierarchy of queues and resources are shared among the sub queues and each user has the limit of some percentage to use the resources. If a queue has serious load, it seeks unallocated resources, then makes redundant resources allotted equally to every job [4]. Once the job put into the queue and running task is completed then, the resources are given back to the main queue. It also allows priority primarily based programming of jobs in associate degree organization queue [21]. Queues in Hadoop are not created automatically. For doing all this we need to know about the system`s information. After knowing about the system`s configuration, the capacity of the queues can be also set which can define the bound on the elasticity of the queue. A single job does not use more resources then its queue but the capacity of the queue can be increased using the property of elasticity. If a queue has serious load, it seeks unallocated resources, then makes redundant resources allotted equally to every job [22]. It also allows priority primarily based programming of jobs in associate degree organization queue [23]. Queues are monitored and are assigned more free resources beyond its capacity if needed. The foremost advanced among three schedulers is a vital drawback in capability algorithm [24]. Capacity Scheduler has other limits to that it can accept the job for a queue or not. It also supports hierarchy of queues and resources are shared among the sub queues and each user has the limit of some percentage to use the resources [25]. If needed queues are monitored and are assigned more free resources beyond its capacity. Creation of the queues is not done automatically, [26] for this the user needs to know about the system information. Queues are monitored by resource manager and if needed more resources are assigned beyond its capacity.

The Capacity scheduler works according to the following:

1. CapacityScheduer.xml contains the existing configuration of the cluster. It contains the task scheduler settings. The queues are set using this information.
2. When the job is submitted to the cluster, the scheduler checks for job submission limit to evaluate that job can be accepted or not.
3. If the job is submitted, it is assigned a queue.
4. The JobTracker gets the heartbeat from the TaskTracker and starts the processing.
5. If required the hierarchy of queues is created and distributed among the various TaskTracker.

III. Experimental Setup

The performance of Hadoop is dependent on the available resource utilization. We had studied the Capacity Scheduler which shares computing resources among the queues. We had performed various experiments on the Map-Reduce application benchmark example TestDFSIO, Pi, Word Count. The purpose of this task is to task of the scheduler and get optimized performance from optimized performance.

E. Approach

With the advancement Hadoop from version 1 to version 2, there was increase in performance in terms of execution time and capacity of the tasks performed. In Hadoop version 1 fair scheduler was used and in latest version of Hadoop approach of capacity scheduler is used. Although capacity scheduler has enhanced the performance, the limitation was that it works on the approach of fair scheduler which can be improved further. There is a scope of further increasing the performance of capacity scheduler by enhancing the concept of pipelining. Once the tasks are performed in a synchronous way and performance of capacity scheduler will increase further in terms of both execution time as well as capacity.

F. Methodology

The methodology used for setting up the system has been shown in the figure 2. Firstly we need to install Java. Then we need to create a group for Hadoop users. SSH certification is used for security purpose in Hadoop. Secondly we need to install Hadoop. After configuring Hadoop, we need to add Dynamic Scheduler to yarn-site.xml.

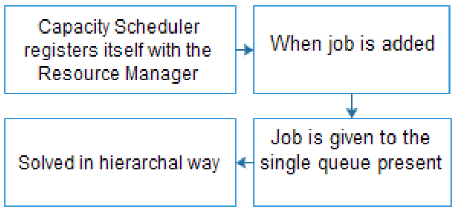
Capacity Scheduler registers itself with the Resource Manager
When job is added
Job is given to the single queue present
Solved in hierarchal way

Fig 2: Working of default scheduler

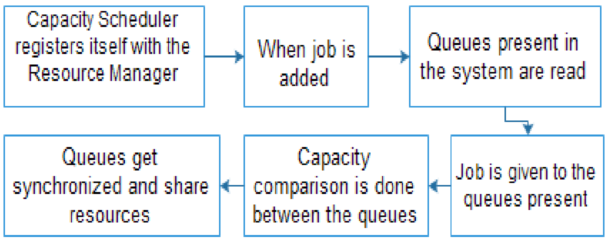
Capacity Scheduler registers itself with the Resource Manager
When job is added
Queues present in the system are read
Job is given to the queues present
Capacity comparison is done between the queues
Queues get synchronized and share resources

Fig 3: Working of our Proposed Method SCQ

Figure 2 and 3 shows the working of default scheduler and the scheduler working with our proposed method SCQ.

G. Working of our Proposed Method SCQ

1. Once the resource manager gets started, everything is happening in the form of events.
2. Capacity scheduler registers itself with the events and acts on those events. When the node is added, ResourceTrackService registers with the node manager.
3. When application or job is added, it will be submitted to the queue. We have added capacity scheduler to the file yarn-site.xml for processing Hadoop files.
4. Once the scheduler starts, the method run on Capacity scheduler gets started. The Component container of the Resource Manager takes the responsibility of all the resources like disk, CPU, memory, etc. Then the job is given to the existing queue and it is solved in the hierarchical way.
5. Now, the queues are being created by the method addNewCSQueue.

```
private void addNewCSQueues( Map queues, Map newQueues)
{
For(Map.Entryx:newQueues.entrySet()){
String queueName = x.getKey();
CSQueue queue = x.getValue();
if
(!queues.containsKey(queueName)) {
queues.put(queueName, queue);                    }
}
}
```

6. Comparison is done between the queues and capacity allocated to the Queues is being calculated by the method CSComparator.

```
static final CSComparator
queueComparator = new Comparator ()
{
public int compare(CSQueue x1, CSQueue x2)
{
if (x1.getUsedCapacity()<x2.getUsedCapacity())
{
        return -1;
}
Else if
(x1.getUsedCapacity () >x2.getUsedCapacity ())
{
        return 1;
}
return x1.getQueuePath().compareTo(x2.getQueuePath ());
}
};
```

7. The synchronization is between the queues is done by using the method SynchroniseCSQueue. While synchronizing, the queues get information about the resources, they share resources accordingly.

We have performed various experiments using the bench mark example Pi, WordCount and TestDFSIO.

H. System configuration:

1) Configuration of Single-Node Machine

Table 1.1 Configuration Of Machine

Hadoop Version	Hadoop 2.6.0
File System	Hadoop File System
Bench Program	Pi, WordCount and TestDFSIO
Operating System	Linux Mint 17.02
Clustered node	Single Node
Processor	CPU I5
CPU	4 core
RAM	4 GB

Table 1.1 shows the single node machine configuration for examining the performance.

2) Configuration of Cluster Machines

Table 1.2 Configuration Of Cluster Machines

Hadoop Version	Hadoop 2.6.0
File System	Hadoop File System
Bench Program	Pi, WordCount and TestDFSIO
Operating System	Linux Mint 17.02
Clustered node	4 Nodes
Processor	CPU I5
CPU	4 core for each node
RAM	4 GB for each node

Table 1.2 shows the configuration of 4 node cluster for examining the performance.

IV. Results and Discussions

We have taken two cases for experiment using the benchmark examples PI, WordCount and TestDFSIO. In case of TestDFSIO we have combined the results of read and write operation and calculated throughput, average IO rate and execution time.

I. In the first experiment, using the configuration of Single Machine from Table 1.1 comparison is done between default configuration of Hadoop version 2.6.0 and our proposed method SCQ on single node of Hadoop version 2.6.0.

For this experiment we have created a single clustered node with Hadoop Version 2.6.0 on the Operating System of Linux Mint 17.02. Experiments are performed using the processor I5.

1) PI

TABLE 2(a) Single-node execution Time of PI in case of default Hadoop version 2.6.0 and our proposed method SCQ on Hadoop version 2.6.0

No. of Maps	**No. of Maps per Sample**	**Execution Time (sec) of default Hadoop (2.6.0)**	**Execution Time (sec) of our Proposed Method SCQ on Hadoop (2.6.0)**	**Estimated Value in both the cases**
5	5	19.965	17.086	3.68
10	10	21.157	21.08	3.2
20	20	37.27	33.211	3.17
30	30	48.321	43.252	3.137777778
40	40	59.346	55.277	3.15
40	45	95.306	79.032	3.148888889

50	60	70.293	66.347	3.141333333
60	70	81.905	79.816	3.14

Table 2(a) shows the execution time of PI in case of default Hadoop version 2.6.0 and our proposed method SCQ on Hadoop version 2.6.0. Default Hadoop defines the Systems default configuration setting and Proposed Method defines the system that we have created SCQ.

We have verified our results, while doing the above experiment by estimating value of Pi, in both the cases i.e. default Hadoop version 2.6.0 and our proposed method SCQ on Hadoop version 2.6.0 was same but the execution time of our proposed was less.

2) WordCount

WordCount has also been executed with different data sets i.e. 12 MB, 36MB, 300 MB, 500 MB, 1 GB, 1.5 GB and 2 GB.

TABLE 2(b), Single-node execution time of WordCount in case of default Hadoop version 2.6.0 and our proposed method SCQ on Hadoop 2.6.0

Data Size	Execution Time (HH:MM:SS) of default Hadoop version 2.6.0	Execution Time (HH:MM:SS) of our proposed method SCQ on Hadoop version 2.6.0
12-2 MB	0:00:20	0:00:18
36-5 MB	0:00:24	0:00:22
300 MB	0:00:42	0:00:40
500 MB	0:00:47	0:00:45
938 MB	0:01:38	0:01:28
1500 MB	0:02:15	0:01:45
2 GB	0:03:43	0:02:26

Table 2(b) shows the execution time of Default Hadoop version 2.6.0 and Proposed Method SCQ on Hadoop version 2.6.0 is better than the Default settings of Hadoop.

3) TestDFSIO

TestDFSIO is used to measure performance of system. The commands read and write file in HDFS that is useful in measuring system performance. A majority of Map-Reduce workloads are IO bound more than compute and hence TestDFSIO can provide an accurate initial picture of such scenarios.

TABLE 2(c) Experimental Results of Throughput

Number of files	Total MBs processed	Throughput (MB/sec) of default Hadoop version 2.6.0	Throughput (MB/sec) of our Proposed Method SCQ on Hadoop version 2.6.0
2	2000	60.60363406	77.39978595
3	3000	44.72412292	47.87551606
4	4000	27.05919615	33.31398167
5	5000	20.17321106	25.96059815

In Table 2(c) shows Throughput of Default Hadoop version 2.6.0 and Proposed Method SCQ on Hadoop version 2.6.0.

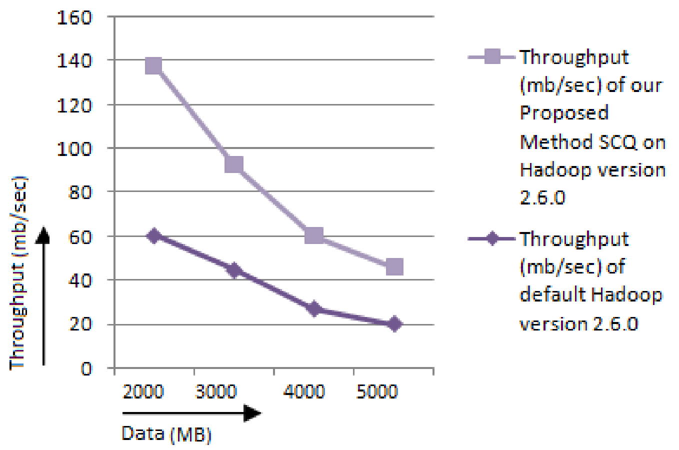

Fig. 4(a): Throughput of default Hadoop version 2.6.0 and our proposed method SCQ on Hadoop 2.6.0
Fig. 4(a) we can see that Throughput of our proposed method SCQ on Hadoop version 2.6.0 is better than the default Hadoop version 2.6.0. As the size of the data is increasing the number bytes to be processed is more. Graph shows that bytes processed are more in case of our proposed method SCQ.

TABLE 2(d) Experimental Results of Average IO rate

Number of files	Total MBs processed	Average IO rate (MB/sec) of default Hadoop version 2.6.0	Average IO rate (MB/sec) of our Proposed Method SCQ on Hadoop version 2.6.0
2	2000	60.60419083	77.42837143
3	3000	44.73816299	47.88214874
4	4000	27.0789423	33.46420876
5	5000	20.18930435	25.99404621

Table 2(d) shows the Average IO rate of default Hadoop version 2.6.0 and our Proposed Method SCQ on Hadoop version 2.6.0.

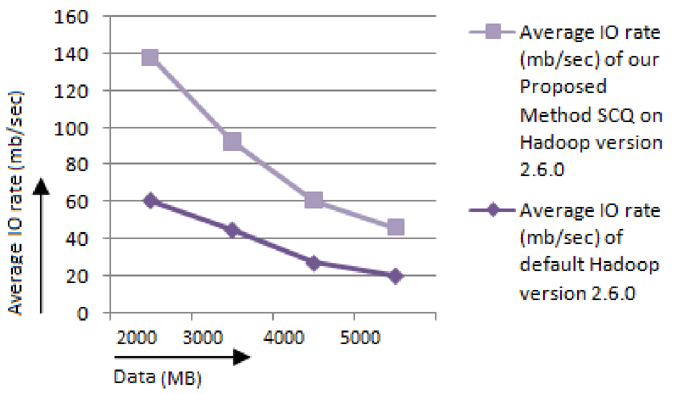

Fig. 4(b): Average IO rate of default Hadoop version 2.6.0 and our proposed method SCQ on Hadoop 2.6.0

Figure 4(b) shows as the size of data increases the average IO rate decreases. As the load increases, our proposed method SCQ shows better results than the default configuration of Hadoop.

TABLE 2(e) Experimental Results of Execution Time

Number of files	Total MBs processed	Test exec time(sec) of default Hadoop version 2.6.0	Test exec time(sec) of our Proposed Method SCQ on Hadoop version 2.6.0
2	2000	100.409	83.508
3	3000	132.31	118.066
4	4000	198.175	164.038
5	5000	289.096	203.145

Table 2(e) shows the Test exec time (sec) of processing Total number bytes of default Hadoop version 2.6.0 and our proposed method SCQ on Hadoop version 2.6.0.

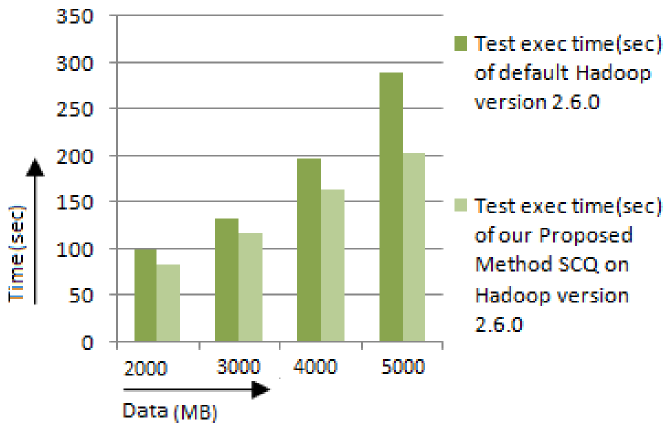

Fig. 4(c) Execution Time of default Hadoop version 2.6.0 and our proposed method SCQ on Hadoop 2.6.0

Figure 4(c) shows as the size of data increases the execution time for processing also increases. From the graph we can see that our proposed method SCQ on Hadoop version 2.6.0 takes less time than the default Hadoop version 2.6.0.

TABLE 2(f) CPU performance of default Hadoop version 2.6.0 and our proposed method SCQ on Hadoop 2.6.0 in case of write operation

No. of files	Total MBs processed	CPU performance of Default Hadoop version 2.6.0	CPU performance of our Proposed Method SCQ on Hadoop version 2.6.0
2	2000	25%	28%
3	3000	24%	30%
4	4000	28%	50%
5	5000	38%	60%

TABLE 2(g) CPU performance of default Hadoop version 2.6.0 and our proposed method SCQ on Hadoop 2.6.0 in case of read operation

No. of files	Total MBs processed	CPU performance of Default Hadoop version 2.6.0	CPU performance of our Proposed Method SCQ on Hadoop version 2.6.0
2	2000	6%	10%
3	3000	6%	10%
4	4000	9%	14%
5	5000	8%	14%

Tables 2(f) and 2(g) also shows the CPU performance of default Hadoop version 2.6.0 and our proposed method SCQ on Hadoop version 2.6.0. Our Proposed method SCQ on Hadoop version 2.6.0 shows better resource utilization than the default configuration of Hadoop version 2.6.0.

J. In the second experiment, using the configuration of Clustered Machine from Table 1.2, comparison has also been done on the cluster of 4 nodes of Hadoop 2.6.0 with our proposed method SCQ and Hadoop 2.6.0 with default configuration.

Table 1.2 shows that in the second experiment, we have used the cluster of 4 nodes with RAM 4GB for each node. In this cluster, we are using the processor I5.

1) PI

TABLE 3(a): Multi-node execution Time of PI in case of default Hadoop version 2.6.0 and our proposed method SCQ on Hadoop 2.6.0

No. of Maps	**No. of samples per Map**	**Execution Time (sec) of default Hadoop(2.6.0)**	**Execution Time (sec) of our proposed method SCQ on Hadoop(2.6.0)**	**Estimated Value in both the cases**
5	5	20.74	19.499	3.68
10	10	23.289	20.168	3.2
20	20	36.368	32.301	3.17
30	30	51.365	44.265	3.13777778
40	40	64.484	55.347	3.15
40	45	63.488	56.349	3.14888889
50	60	77.53	67.526	3.14133333
60	70	94.01	79.428	3.14

Table 3(a) shows the execution time of PI in case of multi-node with default configuration of Hadoop (2.6.0) and our proposed method SCQ on Hadoop (2.6.0). The Execution Time of our Proposed Method SCQ is less. In this case also, we have verified our results as the estimated value in both the cases are same.

2) WordCount

TABLE 3(b) Multi-node execution Time of WordCount in case of default Hadoop version 2.6.0 and our proposed method SCQ on Hadoop 2.6.0

Data size	**Execution time (HH:MM:SS) of default Hadoop version 2.6.0**	**Execution time (HH:MM:SS)of our Proposed Method SCQ on Hadoop version 2.6.0**
12-2 MB	0:00:20	0:00:19
36-5 MB	0:00:23	0:00:22
300 MB	0:00:47	0:00:41
500 MB	0:00:57	0:00:52
938 MB	0:01:56	0:01:46
1500 MB	0:02:16	0:02:09
2 GB	0:03:00	0:02:13

Table 3(b) shows the execution time of default Hadoop (2.6.0) and our proposed method SCQ on Hadoop (2.6.0). Hadoop (2.6.0) with our proposed method SCQ gives better results.

3) Test DFSIO

TABLE 3(c) Experimental Results of Throughput on cluster

Number of files	**Total MBs processed**	**Throughput (MB/sec) of default Hadoop version 2.6.0**	**Throughput (MB/sec) of our Proposed Method SCQ on Hadoop version 2.6.0**
2	2000	33.53691306	37.17452264
3	3000	13.35859242	15.60451619
4	4000	14.30819768	16.76003192
5	5000	12.89387136	15.05275675

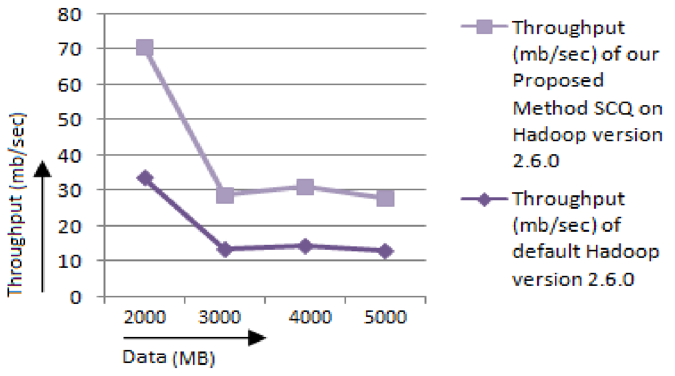

Fig. 5(a) Throughput of default Hadoop (2.6.0) and our proposed method on Hadoop (2.6.0) on cluster

Figure 5(a) and table 3(c) shows that the throughput of Hadoop version 2.6.0 with our proposed method SCQ even on the cluster gives better results i.e. its processing more number of byte than the default configuration of Hadoop version 2.6.0.

TABLE 3(d) Experimental Results of Average IO rate

Number of files	**Total MBs processed**	**Average IO rate (MB/sec) of default Hadoop version 2.6.0**	**Average IO rate (MB/sec) of our Proposed Method SCQ on Hadoop version 2.6.0**
2	2000	33.53774309	37.17828274
3	3000	13.36210346	15.60545063
4	4000	14.309093	16.76963973
5	5000	12.91100812	15.10412097

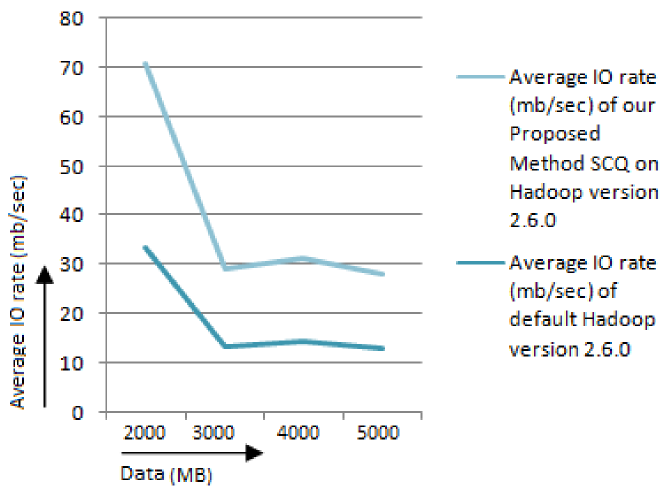

Fig. 5(b) Average IO rate of default Hadoop (2.6.0) and our proposed method on Hadoop (2.6.0) on cluster

TABLE 3(e) Experimental Results of Execution Time

Number of files	**Total MBs processed**	**Test exec time(sec) of default Hadoop version 2.6.0**	**Test exec time(sec) of our Proposed Method SCQ on Hadoop version 2.6.0**
2	2000	251.586	242.818
3	3000	411.852	385.603
4	4000	516.319	465.723
5	5000	650.049	577.913

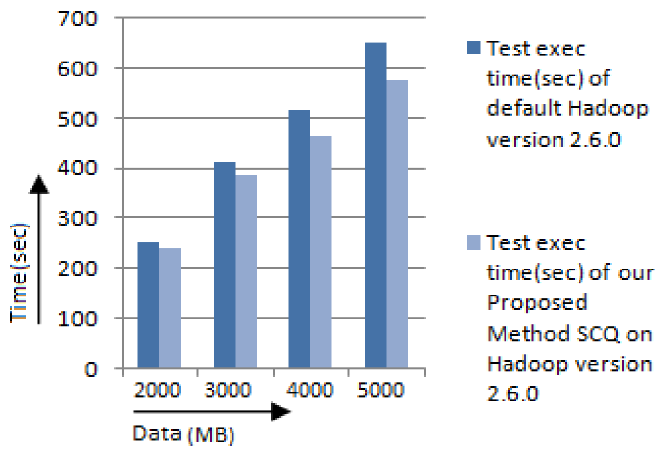

Fig. 5(c) Execution Time of default Hadoop version 2.6.0 and our proposed method SCQ on Hadoop 2.6.0 on cluster

Similarly from the table 3(d) and figure 5(b) shows that our Average IO rate with our proposed method SCQ on Hadoop (2.6.0) gives better results on cluster also. Table 3(e) and figure 6(c) also shows that the Execution Time taken by Hadoop (2.6.0) with our proposed method SCQ is less than default Hadoop 2.6.0 on cluster.

The execution time in case of a cluster is more than the single node of Hadoop because of cluster data replication is there. Due to which it take more time for write operation.

TABLE 3(f) CPU performance of default Hadoop version 2.6.0 and our proposed method SCQ on Hadoop 2.6.0 in case of write operation

No. of files	Total MBs processed	CPU performance of Default Hadoop version 2.6.0	CPU performance of our Proposed Method SCQ on Hadoop version 2.6.0
2	2000	26%	30%
3	3000	35%	48%
4	4000	35%	42%
5	5000	44%	50%

TABLE 3(g) CPU performance of default Hadoop version 2.6.0 and our proposed method SCQ on Hadoop 2.6.0 in case of read operation

No. of files	Total MBs processed	CPU performance of Default Hadoop version 2.6.0	CPU performance of our Proposed Method SCQ on Hadoop version 2.6.0
2	2000	22%	26%
3	3000	20%	24%
4	4000	20%	26%
5	5000	26%	30%

Tables 3(f) and 3(g) also shows the CPU performance of cluster on default Hadoop (2.6.0) and our proposed method SCQ on Hadoop (2.6.0). Here also we can see that the performance of cluster with our proposed method SCQ on Hadoop (2.6.0) is better.

V. Conclusion

Our proposed method is implemented on one of the core schedulers of Hadoop i.e. Capacity scheduler. In this paper, we have performed various experiments using the benchmark examples PI, WordCount and TestDFSIO. Firstly analysis has been done on default configuration of Hadoop (2.6.0) and our proposed method SCQ on the same version on the single node. Secondly analysis is done on cluster of 4 nodes of Hadoop (2.6.0) with our proposed method SCQ and Hadoop (2.6.0) with default configuration. We have also verified our results using the benchmark example PI. Analysis is also done on the performance of CPU and it also shows that the resource utilization is better in case of our proposed method.

From the above two comparisons we can see that our proposed SCQ on Hadoop version 2.6.0 shows better results than default configuration of Hadoop version 2.6.0.

References

[1] The Apache Hadoop Project, http://www.hadoop.org.
[2] Inukollu, V. N., Arsi, S., & Ravuri, S. R. (2014). "Security Issues Associated With Big Data In Cloud Computing". International Journal of Network Security & Its Applications (IJNSA), 6(3).
[3] T. White, Hadoop: The Definitive Guide. O'Reilly Media, Inc., 2009.
[4] Umesh V. Nikam, Anup W. Burange, Abhishek A. Gulhane, "Big Data and HADOOP: A Big Game Changer", International Journal of Advance Research in Computer Science and Management Studies, Volume 1, Issue 7, ISSN: 2321-7782, DEC 2013.
[5] White, T., 2012, "Hadoop: The Definitive Guide", ed. Third, Tokyo: Yahoo press.
[6] Rasooli, Aysan, and Douglas G. Down. "A Hybrid Scheduling Approach For Scalable Heterogeneous Hadoop Systems." In High Performance Computing, Networking, Storage and Analysis (SCC), 2012 SC Companion: pp. 1284-1291. IEEE, 2012.
[7] Kurazumi, Shiori, Tomoaki Tsumura, Shoichi Saito, and Hiroshi Matsuo. "Dynamic Processing Slots Scheduling for I/O Intensive Jobs of Hadoop Map-Reduce." In ICNC, pp. 288-292. 2012.
[8] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The google file system," in 19th ACM Symposium on Operating Systems Principles, Lake George, NY, Oct. 2003.
[9] Jilan Chen, Dan Wang and Wenbing Zhao," A Task Scheduling Algorithm for Hadoop Platform" in Journal of Computers , vol. 8, no. 4, April 2013.
[10] Shvachko, Konstantin, Hairong Kuang, Sanjay Radia, and Robert Chansler. "The hadoop distributed file system." In Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on, pp. 1-10. IEEE, 2010.
[11] H. Herodotou and S. Babu, "Profiling, what-if analysis, and cost-based optimization of Map-Reduce programs," VLDB Endowment (PVLDB), vol. 4, no. 11, pp. 1111–1122, 2011.

[12] G. Wang, A. Butt., P. Pandey, and K. Gupta, "A simulation approach to evaluating design decisions in Map-Reduce setups," in MASCOTS '09, London, UK, Sep. 2009, pp. 1–11.

[13] M. Isard, M. Budiu, Y. Yu, "Distributed Data-Parallel Programs from Sequential Building Blocks," Proceedings of the 2nd ACM SIGOPS European Conference on Computer Systems, ACM, 59-72. 2007.

[14] J. Dean and S. Ghemawat, "Map-Reduce: simplified data processing on large clusters," Communications of the ACM, vol. 51, pp. 107–113, January 2008.

[15] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "A view of Cloud computing," Communications of the ACM, vol. 53, no. 4, pp. 50–58, April 2010. [Online]. Available: http://doi.acm.org/10.1145/1721654.1721672

[16] M. Zaharia, D. Borthakur, J. S. Sarma, K. Elmeleegy, S. Shenker, and I. Stoica, "Delay scheduling: a simple technique for achieving locality and fairness in cluster scheduling," in Proceedings of the 5th European conference on Computer systems, Paris, France, 2010, pp. 265–278.

[17] Hadoop's Fair Scheduler. https://hadoop.apache.org/docs/r1.2.1/fair_scheduler. [As accessed on 9 Feb. 2015].

[18] B.P Andrews and A. Binu,"Survey on Job Schedulers in Hadoop Cluster", IOSR Journal of Computer Engineering, Vol.15, NO. 1, Sep. - Oct. 2013, pp. 46-50.

[19] Y. Tao Y, Q. Zhang, L. Shi and P. Chen. "Job Scheduling Optimization for Multi-user Map-Reduce Clusters", In: The fourth International symposium on parallel Architectures, algorithms and programming. IEEE 2011. p. 213–17.

[20] Y. Chen, S. Alspaugh, and R.H. Katz, ""Interactive Analytical Processing in Big Data Systems: A Cross-Industry Study of Map-Reduce Workloads"" Proc. VLDB Endowment, vol. 5, no. 12, Aug. 2012.

[21] N. Tiwari, "Scheduling and Energy Efficiency Improvement Techniques for Hadoop Map-Reduce: State of Art and Directions for Future Research (Doctoral dissertation, Indian Architectures, algorithms and programming." IEEE; 2011. p. 213–17.

[22] Umesh V. Nikam, Anup W. Burange, Abhishek A. Gulhane, "Big Data and HADOOP: A Big Game Changer", International Journal of Advance Research in Computer Science and Management Studies, Volume 1, Issue 7, ISSN: 2321-7782, DEC 2013.

[23] N. Tiwari, "Scheduling and Energy Efficiency Improvement Techniques for Hadoop Map-Reduce: State of Art and Directions for Future Research (Doctoral dissertation, Indian Architectures, algorithms and programming." IEEE; 2011. p. 213–17.

[24] http://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn- site/CapacityScheduler.html.

[25] Sukhmani Goraya, Vikas Khullar, "Comparative Analysis of Core Schedulers of Hadoop", Multi-Track Conference on Sciences, ISBN No.978-81-929077-2-7, Volume 1.

[26] Sukhmani Goraya, Vikas Khullar, " Enhancing Dynamic Capacity Scheduler for Data Intensive Jobs", International Journal of Computer Applications (0975 – 8887) Volume 121 – No.12, July 2015.