Imperial Journal of Interdisciplinary Research (IJIR)
Vol-2, Issue-10, 2016
ISSN: 2454-1362, http://www.onlinejournal.in

A Modified FIFO with Distributed and Pipelining Scheduling in Hadoop

Shivani Sharma[1] & Dr. Maninder Singh[2]
[1]Department of Computer Science, Punjabi University, Patiala, Punjab.
[2]Department of Computer Science, Punjabi University, Patiala, Punjab.

Abstract- *Big Data is a large amount of data that is being dumped by various companies. Apache Hadoop is a framework that enables the distributed processing of huge data sets across clusters of commodity computers using a simple programming model.*
Scheduling refers to allocation of resources in the best possible way among multiple tasks and arrangement of tasks in queue. A scheduler is responsible for managing that who will get what and on what basis i.e. how many resources for a particular task, what is the priority that could be given to a particular task, on what basis it could be given, how the tasks are arranged in the queue. The default Hadoop scheduler is FIFO- first in and first out. The default FIFO is operated without pipeline i.e. the preceding task has to wait until the ongoing task has finished its execution. This makes the CPU idle for some time until the next task comes for execution. This makes the CPU usage inefficient. So, the concept of pipelining is introduced with FIFO scheduler, so as to decrease the job completion time and make CPU usage efficient. The modified scheduler is tested with five jobs of different sizes and then compared with the default scheduler. On experimental analysis, it is found that the modified scheduler successfully reduces the job completion time, thereby making the CPU usage efficient.

Key Words: *Pipelining, CPU usage, job completion time.*

1. Introduction

As we are living in an era of data, the requirement for storage of large volumes of data is increasing day by day. It's not easy to compute the total amount of large data stored electronically. However, IDC had estimated the digital universe's size approximately around 1.8 zetabytes for the year 2011[1]. One zetabyte is equal to 10^{21} bytes, or one thousand exabytes, one million petabytes, or equivalently one billion terabytes. This is roughly same as one disk drive for each person in the world. So a term is associated to such a large data sets, known as BIG DATA. Big data is a term for a massive amount of both structured as well as unstructured data that is so difficult to process, store and manage using traditional database system and software techniques within a tolerable progressive time. To process, analyze and store such a large amount of data in multiple nodes in a cluster, Hadoop is required. Apache Hadoop is a framework that enables the distributed processing of huge data sets across clusters of commodity computers using a simple programming model. It constitutes of components which are MapReduce, Hadoop Distributed File System, Pig, HBase, Hive, Sqoop, Flume, Oozie, and Zookeeper [2]. Most important amongst them is MapReduce as it is responsible for providing parallel programming model in order to distribute and execute jobs. This programming model consists of Map tasks and Reduce tasks. The Map tasks processes the data first, it generates a kind of key- value pair, and based on that key-pair value, the Reduce tasks reduces the work to give the final output [3].
The load of the cluster increases as the number of jobs submitted by the user increase. To manage such a load in the cluster, a scheduling approach is required so as to enhance the overall performance of the cluster. The scheduling approach should be such that it is able to reduce the job completion time.
The default Hadoop scheduler works using a FIFO queue. After partition of jobs into individual tasks, they are loaded into queue and are then assigned to free slots as they become accessible on TaskTracker nodes. The jobs in the queue would have to wait for their turn in case of default FIFO scheduling. It results in inefficient CPU usage, as the CPU would also have to wait for the next upcoming tasks.
The already working schedulers are also not able to make the CPU usage efficiently. So, a modified FIFO with distributed and pipelining scheduling algorithm in Hadoop is proposed in which the concept of pipelining is used in distribution of tasks to increase the CPU throughput i.e. to make CPU usage efficient. Pipelining increases the task throughput of CPU – the number of tasks finished per unit time. The increase in task throughput means that a job runs with a faster speed and therefore has lesser total execution time.

Until 2008, only a single scheduler was supported with Hadoop that was in-built with the JobTracker logic. However, after this the scenario changed to pluggable schedulers which could be implemented easily. The use of these pluggable schedulers enables the use of new scheduling algorithms in helping optimizing the jobs that have particular characteristics.
The remainder of the paper is organized as follows: Section II consists of the related work. Section III contains the proposed work. Section IV consists of the pipelining algorithm. Section V comprises of results and comparisons. Section VI has conclusion and future works.

2. Related work

Various job scheduling algorithms have been gone through in the literature survey, and therefore many job scheduling algorithms are available [4-14]. The default Hadoop scheduler is FIFO which schedules the job the basis of first come first [5]. But it does not provide fairness and also the execution is sequential which results in more execution time. A fair share of the resources amongst the users is provided by the Fair scheduler [6]. A guaranteed share of cluster capacity amongst the jobs is provided by Capacity scheduler using hierarchical queues [7]. However, it does not allow job preemption.
The main factor that affects the scheduling decisions is synchronization. The synchronization overhead is decreased by MapReduce cluster which re-schedules a speculative copy of late mappers on the other node. In [10-12], in heterogeneous environment the speculative execution of tasks is done. However, there can be improvement in data locality in order to launch speculative map tasks. The synchronization overhead is removed by using asynchronous processing in [13, 14]. In [13], a predefined number of reduce tasks is first started and then the reduces results are collected incrementally from map tasks. Two levels of map and reduce phase: local and global are implemented in [14]. The constraint based schedulers like priority, resource aware and deadline schedulers are responsible for scheduling the tasks based on their priority, resource consumption and deadline of that job [5]. In order to increase the data locality rate the relaxation of ordered jobs for tasks allocation is done in [8]. In [9] matchmaking algorithm is used to provide equal opportunity to each slave node in order to grasp local map tasks.
Therefore, in literature [4-14] various job scheduling algorithms for MapReduce have been proposed. Different parameters have been taken into consideration in order to improve the functionalities of various schedulers. However, till now, making CPU usage efficient through pipelining is not taken into consideration. The processing is sequential during scheduling. So there was a need to focus on this issue too, as it waste a lots of time of CPU in waiting for the tasks. Waiting for a single task by CPU might seem to be a small issue for small sized data, but for the big data, it accounts for a bigger problem. So, a modified FIFO with distributed and pipelining scheduling algorithm in Hadoop is proposed in which the concept of pipelining is used in distribution of tasks to increase the CPU throughput i.e. to make CPU usage efficient. Pipelining increases the task throughput of CPU – the number of tasks finished per unit time. The increase in task throughput means that a job runs with a faster speed and therefore has lesser total execution time.

3. Proposed work

The default Hadoop scheduler works using a FIFO queue. After partition of jobs into individual tasks, they are loaded into queue and are then assigned to free slots as they become accessible on TaskTracker nodes. The jobs in the queue would have to wait for their turn in case of default FIFO scheduling. It results in inefficient CPU usage, as the CPU would also have to wait for the next upcoming tasks.
The already working schedulers are also not able to make the CPU usage efficiently. So, a modified FIFO with distributed and pipelining scheduling algorithm in Hadoop is proposed in which the concept of pipelining is used in distribution of tasks to increase the CPU throughput i.e. to make CPU usage efficient. Pipelining increases the task throughput of CPU – the number of tasks finished per unit time. The increase in task throughput means that a job runs with a faster speed and therefore has lesser total execution time.

4. Pipelining Algorithm

In order to reduce the job completion time and make CPU usage efficient, the pipelining scheduler is used with FIFO scheduler. This modifies the FIFO scheduler with pipelining processing instead of sequential processing. The concept of pipelining and its algorithm are being explained as under:

4.1 Concept of Pipelining

This scheduler helps in arranging the waiting tasks in the form of pipes. As the job to be executed is splitted into various tasks by the MapReducer. These tasks get processed one by one in a single user environment of Hadoop. So a large time gets wasted in waiting for these tasks to be executed. So, the concept of pipelining is introduced which allows the tasks to change its stage each time the succeeding task is getting processed. This could be better explained diagrammatically as:

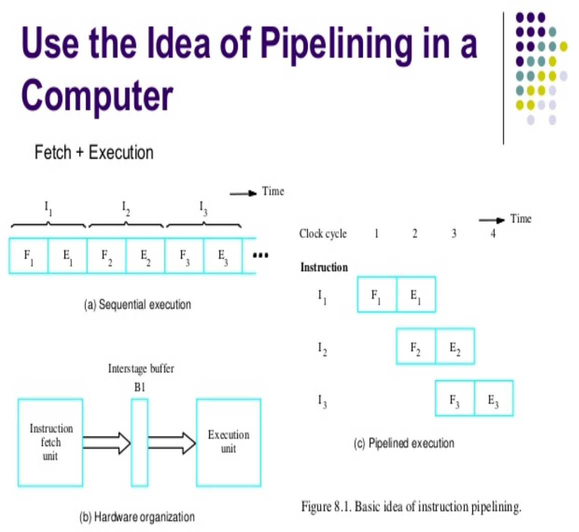

Figure 1. Concept of Pipelining in Computer

The processing through pipelining results in parallel execution of the splitted tasks. Now, the tasks do not have to wait for their stage to get changed. As the succeeding tasks move forward, the task takes its place, and this process goes on as all tasks have finished their part. This parallel execution helps to reduce the execution time for a job to complete. Henceforth, this helps in having efficient CPU usage as the throughput of CPU instruction cycle have increased.

4.2 Algorithm for Pipelining Scheduling in Hadoop

The algorithm required for the development of Pipelining Scheduler in Hadoop is as under:

```
1) Initialize from Config file
2) Minimum allocation, maximum allocation,
node locality
3) Initialize queues
4) For (Csleaf Q: queueManager)
5) {
6) Resource. add (resTopreempt)
7) If (Resource greater than Resource Calculator)
8) Preempt + resource (quemagr. Get LeafQueue)
 // collect running container
9) for (sched: scheds)
10) {
11) If (Resource. greater than Resource Calculator)
12) {
13) for (appsched: sched.getAppSchedulable)
14) {
15) For (RMContainer: getLiveContainer)
16) {
17) Running container. add(c)
18) Apps.put (c, appsched.getApp ()
19) Queues.put (c, sched)
20) }
21) }
22) }
//kill container
25) If (time! = null)
26) {
27) If (time + waitTime BeforeKill <
clock.getTime ())
28) Create preempted container Status (container
.getContainerId ())
        Rescheduled the manager ()
        Initialize the queueManager () {
        Task recalculated ()}
29) CompletedContainer (containers, status,
RMcontainer.KILL)
30) }
31) }
```

4.3 Explanation of algorithm

The proposed algorithm is designed for a single-node Hadoop. This algorithm first of all initializes all the parameters and then resources are added. resTopreempt() function is used to add the resources. If resources required are greater than resources available then running containers are collected using Get LeafQueue() function. In the same way, one more running container is collected. In order to send the task for execution it makes the container and puts the task in it. Simultaneously, another container is collected and preceding task is put into it. These two containers are then added using add() function in order to make a pipeline. Now, if there is some task left whose time spent and still wait time before killing is less than the pre-set time, then status of container to be preempted is created using container .getContainerId () function so as the whole process is rescheduled and the task is recalculated.
In this way by making a pipelining various tasks are scheduled.

5. Results and Comparisons

The proposed scheduling method is implemented in Hadoop2.2 software through VMware tool of version VMware-workstation-full-9.0.1-894247_2 on PC with Intel Core i5 CPU and 8GB RAM and 1 TB Hard Disk, under Ubuntu environment.

The performance of the proposed method is evaluated through best case, average case and worst case by testing each job for 10 times. The method is then compared with the Default FIFO scheduler by testing the same 5 jobs for 10 times. Best case, average case and worst case in this case is noted and then compared with the proposed scheduler.
On comparison we found that the modified scheduler has lesser job completion time than the default scheduler in each best case, average case and worst case.
The comparative view is shown through graph as:

i) For best case:

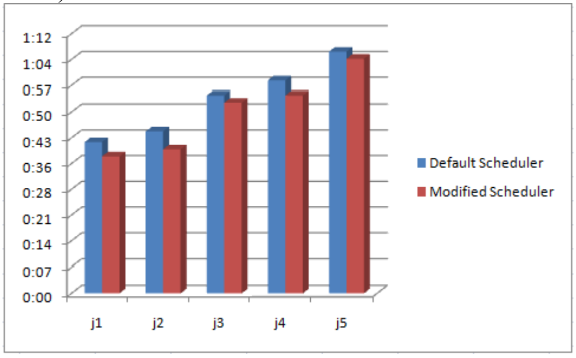

Figure 2. Graph showing results for best case

ii) For Average Case

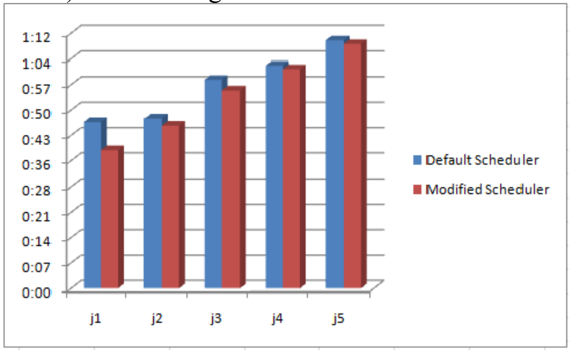

Figure 3. Graph showing results for average case

iii) For Worst Case

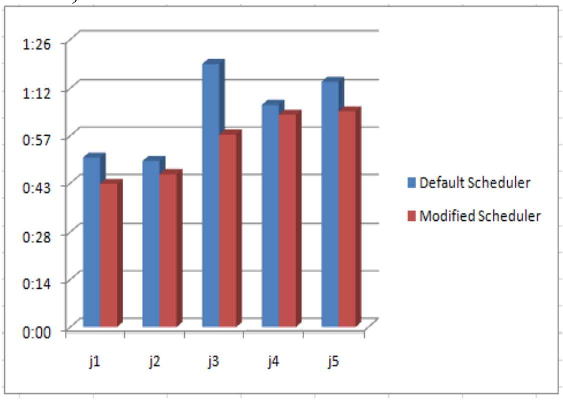

Figure 4. Graph showing results for worst case

From the above experimental analysis, we could conclude that the job completion time in case of the proposed scheduler is lesser than the default scheduler. This is because of pipelining scheduling which we have used in the proposed algorithm. This enables the faster execution of various tasks, thereby making the CPU usage efficient as the CPU does not have to waste its time in waiting for the splitted tasks.

6. Conclusion and Future work

On studying various schedulers, we come to know that the CPU usage is not taken into consideration while developing different schedulers. So there was a need to focus on this issue too. Hence a pipelining algorithm is used with the default FIFO in order to have efficient CPU usage by decreasing the job execution time.
On comparison we found that the modified scheduler has lesser job completion time than the default scheduler in each best case, average case and worst case.

6.1 Future work

- We have applied the pipeline scheduler in a single node Hadoop. It could be further used for a multiple node Hadoop i.e. for a Hadoop cluster also.
- We have performed the analysis for a single job at a time, in case of a Hadoop cluster; this analysis could be performed for multiple jobs at a time.
- In the proposed method we have used only one parameter for evaluation. In future, few more parameters such as earliest deadline first and workload of the job which would give best results in case of the Hadoop cluster.

7. Acknowledgements

We are thankful to Department of Computer Science for kind supports.

8. References

[1] Tom White, “How MapReduce Works”, in Hadoop The Definitive Guide, Third ed. CA: O’REILLY, 2012.

[2] P. Supriya and M.A. Mehta, “Job Aware Scheduling in Hadoop for Heterogeneous Cluster,” International Advance Computing Conference (IACC), 2015 IEEE International Conference, pp. 778 – 783, Bangalore, India, June 2015.

[3] J. Dean and S. Ghemawat, “MapReduce: Simplified Data Processing on Large Clusters,” in Proceeding of the 6th Symposium on Operating systems Design and Implementation (OSDI), pp. 137-150. USENIX Association, December 2004.

[4] D. Yoo and K. M. Sim, "A Comparative Review of Job Scheduling For Mapreduce," in Proceedings of IEEE Cloud Computing and Intelligence Systems (CCIS), pp. 353-358, September 2011.

[5] B. Thirumala Rao and Dr. L. S. S. Reddy, "Survey on Improved Scheduling in Hadoop MapReduce in Cloud Environments," International Journal of Computer Applications (0975 – 8887), November 2011.

[6] "Hadoop MapReduce Next Generation – Fair Scheduler [Online]." Available: http://hadoop.apache.org/docs/current/Hadoop-yarn/Hadoopyarn-site/FairScheduler.html [Last accessed: November, 2014].

[7] "Hadoop MapReduce Next Generation – Capacity Scheduler [Online]." Available: http://hadoop.apache.org/docs/current/Hadoop-yarn/Hadoopyarn-site/CapacityScheduler.html [Last accessed: November, 2014].

[8] M. Zaharia, D. Borthankur, J. Sarma, K. Elmellegy, 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, ACM, pp. 265-278, 2010.

[9] C. He, Y. Lu, and D. Swanson, "Matchmaking: A new mapreduce scheduling technique," IEEE Third International Conference on Cloud Computing Technology and Science (CloudCom), pp. 40-47, December 2011.

[10] M. Zaharia, A. Kowinski, A. Joseph, R. Katz, and I. Stoica, "Improving MapReduce Performance in Heterogeneous Environments," USENIX OSDI, 2008.

[11] Q. Chen, D. Zhang, M Guo, Q. Deng, and S. Guo, "SAMR: A Self-Adaptive MapReduce Scheduling Algorithm In Heterogeneous Environment," IEEE 10th International Conference on Computer and Information Technology (CIT 2010), pp. 2736-2743, July 2010.

[12] X. Sun, C. He and Y. Lu, "ESAMR: An Enhanced Self- Adaptive MapReduce Scheduling Algorithm," IEEE 18th International Conference on Parallel and Distributed Systems, pp. 148-155, December 2012.

[13] M. Elteit, H. Lin, and W. Feng, "Enhancing MapReduce via Asynchronous Data Processing," in Proceedings of IEEE 16th International Conference on Parallel and Distributed Systems (ICPADS), pp. 397-405, December 2010.

[14] K. Kambatla, N. Rapolu, S. Jagannathan, and A. Grama, "Asynchronous Algorithm in MapReduce," in Proceedings - IEEE International Conference on Cluster Computing (ICCC), pp. 245-254, September 2010.