SCHEDULING IN BIG DATA – A REVIEW

Harsh Bansal[1], Vishal Kumar[2] and Vaibhav Doshi[3]

[1]*Assistant Professor, Chandigarh University*
[2]*Assistant Professor, Chandigarh University*
[3]*Assistant Professor, Chandigarh University*
[1]*harshbansal.cse@cumail.in*, [2]*vishalkumar.cse@cumail.in*,
[3]*vaibhavdoshi.cse@cumail.in*

Abstract

In the era of cloud computing where data represents a lot about our world, it becomes important to analyze the data and Hadoop is a promising architecture for processing big data. Hadoop uses MapReduce framework wherein the data is divided into chunks and is parallel processed at multiple nodes via HDFS in very small amount of time. In this paper various scheduling algorithms are reviewed and the challenges faced while scheduling the job onto multiple nodes are discussed considering how factors such as data locality, slow nodes, load distribution, dynamic scheduling could affect and improve the response time of the job.

Keywords: *Big Data, Hadoop, HDFS, MapReduce, Task Scheduling.*

1. Introduction

Before the dawn of the internet, employees of the organizations were only generating the data but with the evolution of the internet now the users have started to generate data of their own and that too in large amount and social media like Facebook being one such place leading to scalability of data. Not only users but machines also talk about sensors, servers which are generating huge amount of data. Formerly, most of the data generated was structured data but not the data being generated in semi structured and unstructured. Structured data was easy to process and mine but semi structured data and unstructured data which are hefty in amount are difficult to process and mine as they require huge amount of computational processing power and the traditional way of processing is not apt. Need of the hour is to use innovative technologies to process enormous data being generated and Big Data is one such technology wherein large amount of data is split into smaller chunks and is then parallel processed by multiple machines/servers.

1.1 Hadoop

Hadoop is open source software allows distributed processing of large data sets allowing to scale up from single machine to thousands of machines hence giving the power to handle the intricacies of high volume i.e. processing terabytes to petabytes of data, velocity i.e. how fast data is being generated and variety of data i.e. heterogeneous and complex data in various formats. MapReduce [1] coupled with Hadoop Distributed File System (HDFS) help to supervise the data which is being split onto multiple machines and being processed. So, mainly Hadoop and MapReduce are the two which are driving big data which can help search for new insights into the data being generated and use it for the benefit of the human race.

The architecture followed by HDFS is of master and slave. It has single master node (also termed as Name Node) which is responsible for managing the operations on files in the global namespace and multiple Data Nodes usually one per node in the bunch which stores the files in the form of data blocks in size of 64MB, 128 MB, 256 MB, 512 MB

etc. Name Node is responsible for mapping of blocks to the Data Nodes and managing the entire HDFS metadata. Besides the above mentioned two nodes there is a node called Secondary Name Node which is not a backup node to Name Node but is responsible for performing checkpoints in the HDFS, so is also termed as Checkpoint Node.

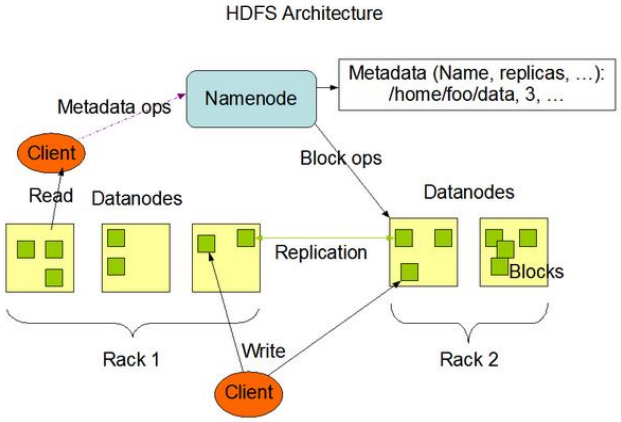

Figure 1. The HDFS Architecture [2]

Job execution is tracked by two nodes i.e. the Job Tracker and the Task Tracker. Job Tracker purpose is to schedule the jobs on the data nodes and Task Trackers run on each data node to perform map and reduce tasks and also send progress back to the job tracker. It's recommended to place Job Tracker on better hardware otherwise all data about the tasks would be lost.

2. Job Scheduling in MapReduce

Today, we have computational resources like computer labs which are free at night, servers which are not being used to their optimum level, etc are available throughout the globe and these computational resources could be used as data nodes at low cost. As these data nodes need not to be of high quality or availability as they perform low-level read and write requests from the clients file system. We take the assumption [2] that the nodes are prone to failure but to deal with this problem we impose failure detection mechanism wherein the data nodes sends heartbeat signals to the name node at small frequent intervals, failing to send signifies that the node has failed so an additional duplicated must be created. Name Node usually creates 3 replicas of each data block in the HDFS in order to provide fault tolerance. High cumulative data bandwidth is the necessity as it has large data sets and should must scale to hundreds of nodes in a single cluster. Locality plays very important role in computation wherein we believe that moving computation closer to the application which requires processing rather than moving the data for computation as it will not only give high bandwidth but will also minimize the network congestion.

It becomes important to focus on how the jobs and tasks are scheduled in the cluster so that maximum efficiency could be achieved. An excellent scheduling algorithm will help reduce the response time of the jobs thereby maximizing the utilization of the underlying cluster. The job scheduling algorithms available in MapReduce are broadly classified as First in First out Scheduler, Capacity Scheduler and Fair Scheduler wherein each one has its own pros and cons.

2.1 First In First Out (FIFO) Scheduler

This was the earliest algorithm which was deployed in Hadoop for scheduling jobs wherein the oldest job in the priority queue is executed first. The job is divided into smaller independent tasks which are executed by the data nodes on the slots available. The slots can further be classified as map slots and reduce slots which are fixed in number. The con is that the response time for shorter jobs is poor in comparison to large jobs i.e. if a large job is submitted just before a small high priority job, the user of the job would have to wait unreasonably for a long time. So, priority based FIFO was introduced where in smaller high priority jobs can now be moved high up in the order but still in this case the preemption is not possible, so the smaller job still had to wait for long time. To improve the turnaround time Capacity Scheduler and Fair Scheduler were introduced.

2.2 Capacity Scheduler

This is the default scheduler which comes up with YARN in Hadoop and was developed at Yahoo[12]. In this case the jobs are divided on the basis of users or groups of users (organizations) with an idea that the same cluster could be rented out to multiple organizations and resource may be divided to serve specific requests as per the SLAs for the organizations. So, it has multiple queues for the organizations wherein each queue is given a portion of resources in the cluster. The allocations can be hard or soft depending upon the SLAs. If there is only one job in the queue and there is no other job in any other queue then in that case the job would take up as many resources as available in the cluster. When another job appears in other queue then the tasks of the first job will be killed to free up the slots for the new job.

2.3 Fair Scheduler

It is very similar to capacity scheduler with the difference that queues are termed as pools wherein the jobs are picked from the pools and would be given their portions of the resources. If another job comes in the pool, the capacity scheduler will process it on FIFO basis. Previously high priority job has to wait for a long time, so this situation is little improved in fair schedulers i.e. the jobs which waited in the queue would be picked up and would be processed in parallel so as to give a better response time.

3. Literature Review

This section mines the various proposed algorithms which help improve the response time of the scheduled tasks considering factors such as locality of the nodes, slow nodes, historical information about the nodes, analyzing the jobs at runtime as follows:

Matei Zaharia et. al. [4] proposed Longest Approximate Time to End (LATE) algorithm, which is highly robust for scheduling of tasks in heterogeneous environment and could improve the Hadoop response times of speculative tasks specially of slow tasks because of slow nodes due which could be due to high CPU load, slow background process by prioritizing them and scheduling them on faster nodes depending upon how badly they can impact the response time of the submitted job. LATE resolves the problem of slow nodes which are 2-3 times slower than the mean thereby scheduling them on faster nodes which complete their tasks and asks for a new one. It does not take data locality into

account for launching speculative map tasks and assumes most maps are data-local and network utilizations during the map phase is low.

Jord`a Polo et. al. [3] proposed task scheduling algorithm which evaluates the fulfillment time of the each submitted job wherein the job is separated into huge number of errands/tasks and the advancement of the job can be seen at runtime, in this manner observing the undertaking length of effectively finished tasks so to foresee the job consummation time and in like manner the quantity of task slots that each job has been apportioned are balanced which results in runtime execution administration of the tasks. The scheduler makes utilization of min scheduler and max scheduler way to deal with dispense of the resources.

Quan Chen et. al. [6] expanded the thought as proposed by [4] for LATE scheduling algorithm by sparing the execution time of the MapReduce thereby distinguishing the slow tasks, not statically as in LATE as they can't locate the fitting tasks which truly drag out the execute time however its done by fusing chronicled data recorded on every node to tune parameters and find slow tasks dynamically in heterogeneous computing environment. It likewise help spare the framework assets there by characterizing slow nodes as map slow nodes and reduce slow nodes and propelling reinforcement of backup map tasks on fast nodes. The outcomes are analyzed and the historical information is continually refreshed.

Xiaoyu Sun et. al. [11] proposed ESAMR algorithm which not just uses chronicled data of the nodes to alter weights of map and reduce tasks as proposed an updated form of SAMR by Quan Chen [6] yet additionally characterizes the nodes into the k clusters by making utilization of k-means clustering algorithm and keeping in mind while scheduling, it utilizes the cluster’s weights to gauge the execution time of the job’s tasks on the node which thusly serves to accurately recognize slow tasks and re-execute them. ESAMR significantly enhances the execution of MapReduce scheduling in terms of evaluating task execution time and propelling reinforcement tasks.

Radheshyam Nanduri et. al. [7] proposed job aware scheduling algorithm for MapReduce Framework wherein it tries to schedule the tasks in such a way that they are resource compatible on the nodes in terms of CPU, memory, disk and network to avoid race condition for resources on a node of the cluster thereby optimally utilizing the resources without overloading by using heuristic approach which makes use of cosine similarity and machine learning based solutions which makes use of Naïve- Bayes classifier to calculate the probability that a task is compatible . Task Selection algorithm and Task Assignment algorithm are used to select the task that is best suitable on a particular node. This can be a valuable addition to the features of Capacity and Fair Scheduler.

Mohammad Hammoud et. al. [5] proposed data locality based scheduling algorithm while performing out the reduce operation upon the tasks as Hadoop endeavors to schedule map tasks in nearness to input splits with the goal that they require not to be exchanged over the network while it doesn't contemplate data locality while scheduling reduce operation. As moving the data crosswise over nodes on the distinctive racks would prompt network blockage and performance debasement, it makes Hadoop's reduce task scheduler aware of partitions' network locations and sizes so as to mitigate network

traffic and endeavors to schedule reducers as close as conceivable to their most extreme measure of input data.

Xiaohong Zhang et. al. [10] worked on data locality aware task scheduling algorithm to enhance Hadoop MapReduce framework performance in heterogeneous computing environment, in this manner ideally scheduling the task on the requesting node if the input data exists there and if not the strategy chooses the task whose input data is closest to the requesting node and afterward settles on the choice whether to save the task for the node having the input data or schedule the task on the requesting node by transferring the input data to the requesting node. It optimizes the response time by making tradeoff between the waiting time and the transmission time at runtime.

Tseng-Yi Chen et. al. [8] proposed locality aware scheduling algorithm wherein it prohibits the assigning of data node which has data which is relatively scare considering it as rare resource otherwise data is very common in free slot node. It calculates the weight of data interference on each node by determining the weight of each data in the node which is calculated from factors such as number of map free slots on each node and number of required data. Based on this it keeps the node reserved hence enabling lesser data movement leading to reduced network latency and better response time.

Weina Wang et. al. [9] worked on locality of the data as well as also considered balancing the load of the nodes as assigning all the tasks locally without considering the load on the machine could lead to poor response time of the tasks getting executed. Some other algorithms make the tasks wait while scheduling them on the nodes to attain higher locality. This algorithm proposes two queues for nodes instead of one to minimize the waiting time and congestion in the cluster with respect to locality where in one is used for storing the local tasks which are associated with the machine and other queue which is common to all the nodes which works in combination with two stage scheduling algorithm wherein task assignment to one of the two queues is done via Join the Shortest Queue policy in which when a task comes to the master it compares the length of all the local queues and the common queue and whichever is the shortest will be assigned with the task and via the Max Weight policy in which machines will serve the task which are scheduled depending upon the availability of same.

4. Conclusion

While scheduling the tasks on the nodes it is viable to analyze the slow nodes so to schedule the tasks on faster nodes. To make the scheduling more efficient, historical information recorded on each node can also be incorporated which can help find slow tasks dynamically. Also, analyzing the advancement of the job at the runtime could help manage the job completion time thereby adjusting the number of task slots required for the job. Scheduling the tasks on resource compatible nodes in terms of CPU, memory, disk and network could help avoid race conditions. Another factor which can help drastically is data locality i.e. scheduling the tasks closer to the nodes with the data so as to avoid network congestion and performance degradation and also making the Hadoop's reduce task scheduler aware of partitions could help mitigate the network traffic thereby scheduling reducers closer to input data. Sometimes tasks are kept await while scheduling, in that case making tradeoff between the waiting time and the transmission time by scheduling the task at requesting node instead of task with input data at runtime could help elevate the performance.

References

[1] Hadoop, hadoop.apache.org

[2] HDFS Architecture, https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html#NameNode+and+DataNodes.

[3] Jord`a Polo, David Carrera, Yolanda Becerra, Jordi Torres and Eduard Ayguad´, Malgorzata Steinder and Ian Whalley, "Performance-Driven Task Co-Scheduling for MapReduce Environments", IEEE/IFIP Network Operations and Management Symposium - NOMS, 2010, pp 373-380.

[4] Matei Zaharia, Andy Konwinski, Anthony D. Joseph, Randy Katz, Ion Stoica, "Improving MapReduce Performance in Heterogeneous Environments", 8th USENIX Symposium on Operating Systems Design and Implementation, 2008, pp 29-42.

[5] Mohammad Hammoud and Majd F. Sakr, "Locality-Aware Reduce Task Scheduling for MapReduce", Third IEEE International Conference on Cloud Computing Technology and Science, 2011, pp 570-576.

[6] Quan Chen, Daqiang Zhang, Minyi Guo, Qianni Deng, Song Guo, "SAMR: A Self-adaptive MapReduce Scheduling Algorithm in Heterogeneous Environment", 10th IEEE International Conference on Computer and Information Technology, 2010, pp 2736 - 2743.

[7] Radheshyam Nanduri, Nitesh Maheshwari, Reddyraja. A and Vasudeva Varma, "Job Aware Scheduling Algorithm for MapReduce Framework", Third IEEE International Conference on Cloud Computing Technology and Science, 2011, pp 724-729.

[8] Tseng-Yi Chen, Hsin-Wen Wei, Ming-Feng Wei, Ying-Jie Chen, Tsan-sheng Hsu, Wei-Kuan Shih," LaSA: A Locality-aware Scheduling Algorithm for Hadoop-MapReduce Resource Assignment", International Conference on Collaboration Technologies and Systems (CTS), 2013, pp. 342-346.

[9] Weina Wang, Kai Zhu, Lei Ying, Jian Tan, and Li Zhang, "MapTask Scheduling in MapReduce With Data Locality: Throughput and Heavy-Traffic Optimality", IEEE/ACM Transactions on Networking, Volume: 24, Issue: 1 , Feb. 2016, pp 1-14.

[10] Xiaohong Zhang, Yuhong Feng, Shengzhong Feng, Jianping Fan and Zhong Ming, "An Effective Data Locality Aware Task Scheduling Method for MapReduce Framework in Heterogeneous Environments", International Conference on Cloud and Service Computing, 2011, pp. 235-242.

[11] Xiaoyu Sun, Chen He and Ying Lu, "ESAMR: An Enhanced Self-Adaptive MapReduce Scheduling Algorithm", IEEE 18th International Conference on Parallel and Distributed Systems, 2012, pp. 148 – 155.

[12] Yahoo! inc. Capacity Scheduler. http://hadoop.apache. org/docs/stable/capacity_scheduler.html.