IOSR Journal of Computer Engineering (IOSR-JCE)
e-ISSN: 2278-0661,p-ISSN: 2278-8727, Volume 17, Issue 1, Ver. IV (Jan – Feb. 2015), PP 36-40
www.iosrjournals.org

Big Data Analysis and Its Scheduling Policy – Hadoop

Divya S[1]., Kanya Rajesh R[2]., Rini Mary Nithila I[3]., Vinothini M[4].,
PG Scholars[1, 2, 3, 4] Department of Information Technology,
Francis Xavier Engineering College, Anna University, TamilNadu, India

***Abstract**: This paper is deals with Parallel Distributed system. Hadoop has become a central platform to store big data through its Hadoop Distributed File System (HDFS) as well as to run analytics on this stored big data using its MapReduce component. Map Reduce programming model have shown great value in processing huge amount of data. Map Reduce is a common framework for data-intensive distributed computing of batch jobs. Hadoop Distributed File System (HDFS) is a Java-based file system that provides scalable and reliable data storage that is designed to span large clusters of commodity servers. In all Hadoop implementations, the default FIFO scheduler is available where jobs are scheduled in FIFO order with support for other priority based schedulers also. During this paper, we are going to study a Hadoop framework, HDFS design and Map reduce Programming model. And also various schedulers possible with Hadoop and provided some behavior of the current scheduling schemes in Hadoop on a locally deployed cluster is described.*
***Keywords:** Hadoop, Map-Reduce, Big Data Analytics, Scheduling Algorithms*

I. Introduction

Apache Hadoop is a software framework for processing Big Data such as data in the range of petabytes . The framework was originally developed by Doug Cutting, the creator of Apache Lucene, as part of his Web Search Engine Apache Nutch . Hadoop is an open source, large scale, batch data processing, distributed computing framework for big data storage and analytics. It facilitates scalability and takes care of detecting and handling failures. Hadoop ensures high availability of data by creating multiple copies of the data in different nodes throughout the cluster. In Hadoop, the code is moved to the location of the data instead of moving the data towards the code. Hadoop is a highly efficient and reliable cloud computing platform, which can be deployed in a cloud computing data center. Hadoop users need not concern on the low-level details of distributed system but focus on their business need when they develop distributed applications. That is why lots of famous Internet service providers, including Facebook, Twitter, Yahoo, Baidu, and Alibaba, have already chosen Hadoop as one of the core components to build their own cloud systems to provide more efficient services.

Hadoop MapReduce is a programming model and software framework for writing applications that rapidly process vast amounts of data in parallel on large clusters of compute nodes. In Hadoop, the Map Reduce programming model is used. MapReduce is the framework for processing large volume of datasets as key value pairs. MapReduce divides each job in to two types of functions, map and reduce. Both Map and Reduce functions take input as key value pairs and emits the result as another set of key value pairs. Each job is divided in to number of map tasks and Reduce tasks. The input is initially processed in distributed map tasks and aggregate the result with the help of reduce tasks. In order to complete the transactions submitted by users efficiently, Hadoop needs the right job scheduling algorithm and appropriate task scheduling algorithm. Job and task are different concepts in Hadoop. When a user submits a transaction, Hadoop will create a job and put it in the queue of jobs waiting for the job scheduler to dispatch it. Then, the job will be divided into a series of tasks, which can be executed in parallel. The task scheduler is responsible for dispatching tasks by certain task scheduling algorithm. Several job scheduling algorithms have been developed, including first-in–first-out scheduling, fair scheduling, and capacity scheduling.

The rest of the paper is organized as follows. Section 2 describes why Hadoop for big data analytics. Section 3 provides a brief introduction to Hadoop, HDFS and Map/Reduce framework. Section 4 presents our analysis of Hadoop policies in Hadoop cluster.

II. Why Hadoop For Big Data Analytics?

Big Data is moving from a focus on individual projects to an influence on enterprises strategic information architecture. Dealing with data volume, variety, velocity and complexity is forcing changes to many traditional approaches. This realization is leading organizations to abandon the concept of a single enterprise data warehouse containing all information needed for decisions. Instead, they are moving towards multiple systems, including content management, data warehouses, data marts and specialized file systems tied together with data services and metadata, which will become the logical enterprise data warehouse. There are various

systems available for big data processing and analytics, alternatives to Hadoop such as HPCC or the newly launched Red Shift by Amazon. However, the success of Hadoop can be gauged by the number of Hadoop distributions available from different technological companies such as IBM Info Sphere Big Insights, Microsoft HDInsight Service on Azure, Cloudera Hadoop, Yahoo's distribution of Hadoop, and many more. There are basically four reasons behind its success:

- It's an open source project.
- It can be used in numerous domains.
- It has a lot of scope for improvement with respect to fault tolerance, availability and file systems.

There are a number of reasons why Hadoop is an attractive option. Not only does the platform offer both distributed computing and computational capabilities at a relatively low cost, it's able to scale to meet the anticipated exponential increase in data generated by mobile technology, social media, the Internet of Things, and other emerging digital technologies

III. Hadoop Framework

Hadoop is an open source software framework that dramatically simplifies writing distributed data intensive applications. It provides a distributed file system, which is modeled after the Google File System and a map/reduce implementation that manages distributed computation.

3.1 HDFS

Hadoop Distributed File System (HDFS) is a Java-based file system that provides scalable and reliable data storage that is designed to span large clusters of commodity servers. The HDFS is designed as master/slave architecture (Fig. 1). A HDFS cluster consists of Name Node, a master node that manages the filing system name space and regulates access to files by clients. Additionally, there are various Data Nodes, usually one per node within the cluster, that manage storage connected to the nodes that they run on. HDFS exposes a filing system name space and permits user information to be hold on in files. Internally, a file is split into one or a lot of blocks and these blocks are stored in a set of Data Nodes. The Name Node executes filing system name space operations like gap, closing, and renaming files and directories. It determines the mapping of blocks to Data Nodes. Clients read and write request are served by the Data node. The Data Nodes is also responsible for block creation, deletion, and replication as per the instruction given by Name Node.

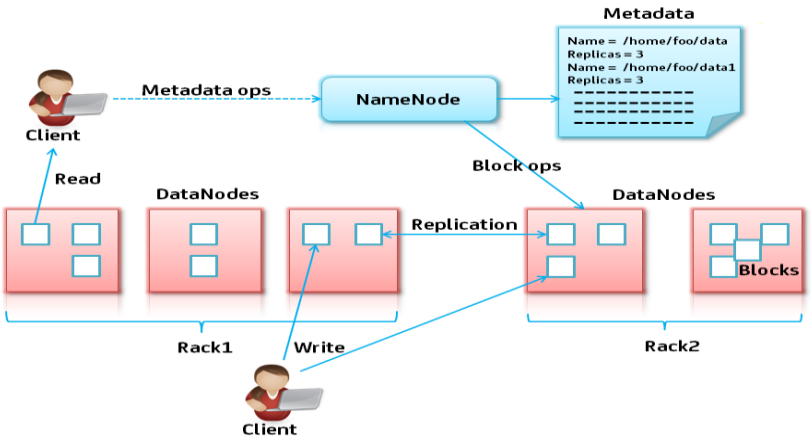

Figure 1: HDFS Architecture

3.1.1 How HDFS works?

An HDFS cluster is comprised of a Namenode which manages the cluster metadata and DataNodes that store the data. Files and directories are represented on the NameNode by nodes. Nodes record attributes like permissions, modification and access times, or namespace and disk space quotas. The file content is split into large blocks (typically 128 megabytes), and each block of the file is independently replicated at multiple DataNodes. The blocks are stored on the local file system on the datanodes. The Namenode actively monitors the number of replicas of a block. When a replica of a block is lost due to a DataNode failure or disk failure, the NameNode creates another replica of the block. The NameNode maintains the namespace tree and the mapping of blocks to DataNodes, holding the entire namespace image in RAM. The NameNode does not directly send requests to DataNodes. It sends instructions to the DataNodes by replying to heartbeats sent by those

DataNodes. The instructions include commands to: replicate blocks to other nodes, remove local block replicas, re-register and send an immediate block report, or shut down the node.

3.2 Map Reduce Programming Model

The Map Reduce programming model is designed to process large data set in parallel by distributing the Job into a various independent Tasks. The Job considered to here as a complete Map Reduce program, which is the execution of a Mapper or Reducer across a set of data. A Task is an executing a Mapper or Reducer on a chunks of the data. Then the Map Reduce Job normally splits the input data into independent portions, which are executed by the map tasks in a fully parallel fashion. The Hadoop Map Reduce framework consists of a one Master node that runs a Job tracker instance which takes Job requests from a client node and Slave nodes everyone running a Task Tracker instance. The Job tracker is responsible for distributing the job to the Slave nodes, scheduling the job's component tasks on the Task Trackers, monitoring them as well as reassigning tasks to the Task Trackers when they at the time of failure. It also provides the status and diagnostic information to the client. The task given by the Job tracker is executed by the Task Tracker. Fig. 2 depicts the Map Reduce framework.

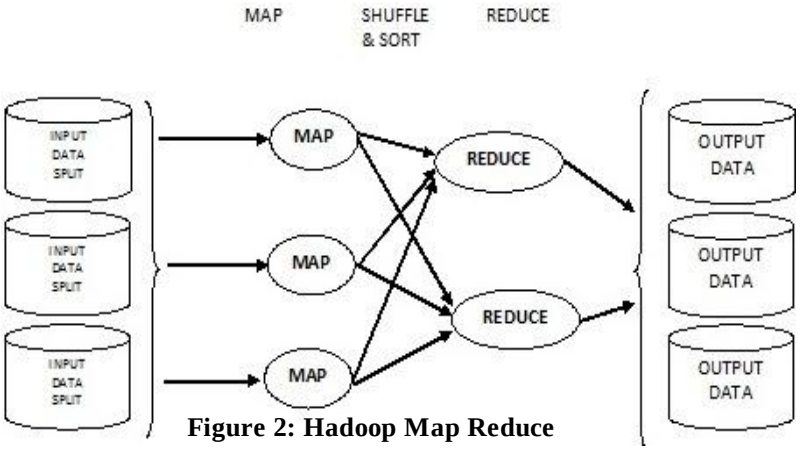

Figure 2: Hadoop Map Reduce

3.2.1 Execution of Map Task

Every map task is provided with a portion of the input file called as split. By default, a split contains a one HDFS block, so the total number of the number of map tasks is equal to the total number of file blocks. The execution of a map task is divided into two passes:

1) The map pass reads the task's split from HDFS, interpreted it into records (key/value pairs), and applies the map function to every record.

2) After applying map function to every input record, the commit phase stores the final output with the Task-Tracker, Then Task Tracker informs the Job-Tracker that the task has finished its execution. The map method specifies an Output Collector instance, which collects the output records created by the map function. The output of the map step is consumed by the reduce step, so that Output Collector stores output produce by mapper in a simpler format so that it is easy to consume for reduce task. The Task-Tracker will read these data and index files when servicing requests from reduce tasks.

3.2.2 Execution Of Reduce Task

The execution of a reduce task is divided into three passes.

1) The shuffle phase is responsible for fetching the reduce task's input file. Each reduce task is appointed a partition of the key range generated by the map pass, so that the reduce task must receive the content of this partition from each map task's output.

2) The sort phase group together records having same key. Public interface Reducer <K2, V2, K3, V3>
{Void Reduce (K2 key, Iterator<V2> values, Output Collector <K3, V3> output); Void close () ;}

3) The reduce phase appoint the user-defined reduce function to every key and corresponding list of values. Within the shuffle phase, a reduce task fetches data from every map task by sending HTTP requests to a configurable number of Task-Trackers at once .The Job-Tracker relays the location of each Task-Tracker that hosts map output to each Task-Tracker that executes a reduce task. Reduce task can't receive the output of a map task till the map has finished execution and committed its final output to disk. The output of both map and reduce tasks is written to disk before it can be consumed.

IV. Scheduling Policies In Hadoop

4.1 Job Scheduling In Hadoop

When Hadoop started out, it was designed mainly for running large batch jobs such as web indexing and log mining. Users submitted jobs to a queue, and the cluster ran them in order. However, as organizations placed more data in their Hadoop clusters and developed more computations they wanted to run, another use case became attractive: sharing a MapReduce cluster between multiple users. The benefits of sharing are tremendous: with all the data in one place, users can run queries that they may never have been able to execute otherwise, and costs go down because system utilization is higher than building a separate Hadoop cluster for each group. However, sharing requires support from the Hadoop job scheduler to provide guaranteed capacity to production jobs and good response time to interactive jobs while allocating resources fairly between users. The scheduler in Hadoop became a pluggable component and opened the door for innovation in this space. The result was two schedulers for multi-user workloads: the Fair Scheduler developed at Facebook, and the Capacity Scheduler, developed at Yahoo.

4.1.1 Default FIFO Scheduler

The default Hadoop scheduler operates using a FIFO queue. After a job is partitioned into individual tasks, they are loaded into the queue and assigned to free slots as they become available on TaskTracker nodes. Although there is support for assignment of priorities to jobs, this is not turned on by default.

4.1.2 Fair Scheduler

The Fair Scheduler was developed at Facebook to manage access to their Hadoop cluster, which runs several large jobs computing user metrics, etc. on several TBs of data daily. Users may assign jobs to pools, with each pool allocated a guaranteed minimum number of Map and Reduce slots. Free slots in idle pools may be allocated to other pools, while excess capacity within a pool is shared among jobs. In addition, administrators may enforce priority settings on certain pools. Tasks are therefore scheduled in an interleaved manner, based on their priority within their pool, and the cluster capacity and usage of their pool. Over time, this has the effect of ensuring that jobs receive roughly equal amounts of resources. Shorter jobs are allocated sufficient resources to finish quickly. At the same time, longer jobs are guaranteed to not be starved of resources.
The Fair Scheduler is based on three concepts:

- Jobs are placed into named "pools" based on a configurable attribute such as user name, UNIX group, or specifically tagging a job as being in a particular pool through its jobconf.
- Each pool can have a "guaranteed capacity" that is specified through a config file, which gives a minimum number of map slots and reduce slots to allocate to the pool. When there are pending jobs in the pool, it gets at least this many slots, but if it has no jobs, the slots can be used by other pools.
- Excess capacity that is not going toward a pool's minimum is allocated between jobs using fair sharing. Fair sharing ensures that over time, each job receives roughly the same amount of resources. This means that shorter jobs will finish quickly, while longer jobs are guaranteed not to get starved.

4.1.3 Capacity Scheduler

The Capacity Scheduler is a pluggable scheduler for Hadoop that allows multiple tenants to securely share a large cluster. Resources are allocated to each tenant's applications in a way that fully utilizes the cluster, governed by the constraints of allocated capacities. Queues are typically set up by administrators to reflect the economics of the shared cluster. The Capacity Scheduler supports hierarchical queues to ensure that resources are shared among the sub-queues of an organization before other queues are allowed to use free resources. The Capacity Scheduler is designed to run Hadoop applications as a shared, multi-tenant cluster in an operator-friendly manner while maximizing the throughput and the utilization of the cluster. The Capacity Scheduler is designed to allow sharing a large cluster while giving each organization capacity guarantees. The central idea is that the available resources in the Hadoop cluster are shared among multiple organizations who collectively fund the cluster based on their computing needs. There is an added benefit that an organization can access any excess capacity not being used by others. This provides elasticity for the organizations in a cost-effective manner. Sharing clusters across organizations necessitates strong support for multi-tenancy since each organization must be guaranteed capacity and safe-guards to ensure the shared cluster is impervious to single rouge application or user or sets thereof.

The Capacity Scheduler provides a stringent set of limits to ensure that a single application or user or queue cannot consume disproportionate amount of resources in the cluster. Also, the Capacity Scheduler provides limits on initialized/pending applications from a single user and queue to ensure fairness and stability of the cluster. The primary abstraction provided by the Capacity Scheduler is the concept of queues. To provide further control and predictability on sharing of resources, the Capacity Scheduler supports hierarchical queues to ensure resources are shared among the sub-queues of an organization before other queues are allowed

to use free resources, there-by providing affinity for sharing free resources among applications of a given organization.

4.2 Resource Aware Scheduling

The Fair Scheduler and Capacity Scheduler described above attempt to allocate capacity fairly among users and jobs without considering resource availability on a more fine-grained basis. As CPU and disk channel capacity has been increasing in recent years, a Hadoop cluster with heterogeneous nodes could exhibit significant diversity in processing power and disk access speed among nodes. Performance could be affected if multiple processor-intensive or data-intensive tasks are allocated onto nodes with slow processors or disk channels respectively. This possibility arises as the Job Tracker simply treats each Task Tracker node as having a number of available task slots. Even the improved LATE speculative execution could end up increasing the degree of congestion within a busy cluster, if speculative copies are simply assigned to machines that are already close to maximum resource utilization.

. Scheduling in Hadoop is centralized, and worker initiated. Scheduling decisions are taken by a master node, called the Job Tracker, whereas the worker nodes, called TaskTracker are responsible for task execution. The Job Tracker maintains a queue of currently running jobs, states of TaskTracker in a cluster, and list of tasks allocated to each TaskTracker. Each Task Tracker node is currently configured with a maximum number of available computation slots. Although this can be configured on a per-node basis to reflect the actual processing power and disk channel speed, etc. available on cluster machines, there is no online modification of this slot capacity available.

Two possible resource-aware Job Tracker scheduling mechanisms are: 1) Dynamic Free Slot Advertisement-Instead of having a fixed number of available computation slots configured on each Task Tracker node, this number is computed dynamically using the resource metrics obtained from each node. In one possible heuristic, overall resource availability is set on a machine to be the minimum availability across all resource metrics. In a cluster that is not running at maximum utilization at all times, this is expected to improve job response times significantly as no machine is running tasks in a manner that runs into a resource bottleneck. 2) Free Slot Priorities/Filtering- In this mechanism, cluster administrators will configure maximum number of compute slots per node at configuration time. The order in which free TaskTracker slots are advertised is decided according to their resource availability. As TaskTracker slots become free, they are buffered for some small time period (say, 2s) and advertised in a block. TaskTracker slots with higher resource availability are presented first for scheduling tasks on. In an environment where even short jobs take a relatively long time to complete, this will present significant performance gains. Instead of scheduling a task onto the next available free slot (which happens to be a relatively resource-deficient machine at this point), job response time would be improved by scheduling it onto a resource-rich machine, even if such a node takes a longer time to become available. Buffering the advertisement of free slots allowed for this scheduling allocation.

V. Conclusion & Future Work

Hadoop is a popular open source implementation platform of MapReduce model and used to process and analyze large-scale data sets in parallel. Ability to make Hadoop scheduler resource aware is one the emerging research problem that grabs the attention of most of the researchers as the current implementation is based on statically configured slots. This paper summarizes pros and cons of Scheduling policies of various Hadoop Schedulers developed by different communities. Each of the Scheduler considers the resources like CPU, Memory, Job deadlines and IO etc. All the schedulers discussed in this paper addresses one or more problem(s) in scheduling in Hadoop. Nevertheless all the schedulers discussed above assumes homogeneous Hadoop clusters. Future work will consider scheduling in Hadoop in Heterogeneous Clusters.

References

[1]. Apache Hadoop. http://hadoop.apache.org
[2]. J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. OSDI '04, pages 137–150, 2004
[3]. Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung.The Google file system. In 19th Symposium on Operating Systems Principles, pages 29–43, Lake George, New York, 2003.
[4]. Hadoop Distributed File System, http://hadoop.apache.org/hdfs
[5]. Hadoop's Fair Scheduler http://hadoop.apache.org/common/docs/r0.20.2/fair_schedu ler.html
[6]. Hadoop's Capacity Scheduler: http://hadoop.apache.org/core/docs/current/capacity_sched uler.html
[7]. K. Kc and K. Anyanwu, "Scheduling Hadoop Jobs to Meet Deadlines", in Proc. CloudCom, 2010, pp.388-392.
[8]. Mark Yong, Nitin Garegrat, Shiwali Mohan: "Towards a Resource Aware Scheduler in Hadoop" in Proc. ICWS, 2009, pp:102-109
[9]. B.Thirmala Rao, N.V.Sridevei, V. Krishna Reddy, LSS.Reddy. Performance Issues of Heterogeneous Hadoop Clusters in Cloud Computing. Global Journal Computer Science & Technology Vol. 11, no. 8, May 2011, pp.81-87
[10]. Matei Zaharia, Dhruba Borthakur, Joydeep Sen Sarma, Khaled Elmeleegy, Scott Shenker, and Ion Stoica. Delay scheduling: a simple technique for achieving locality and fairness in cluster scheduling. In EuroSys '10: Proceedings of the 5th European conference on Computer systems, pages 265–278, New York, NY, USA, 2010. ACM.