Scientific Journal Impact Factor (SJIF) : 1.711
International Journal of Modern Trends in Engineering and Research
ISSN (ONLINE) : 2349 - 9745 ISSN (PRINT) : 2393 - 8161

Study on Hadoop and MapReduce Framework

N.Kiruthika[1], T.K.P.Rajagopal[2]

[1]*Department of Computer Science and Engineering, Kathir College of Engineering,*
[2] *Department of Computer Science and Engineering, Kathir College of Engineering,*

Abstract— Hadoop, a Java Software Framework, supports data intensive data-intensive distributed applications. Hadoop is developed under open source license. It enables applications to work with thousands of nodes and petabytes of data. Hadoop has formed framework for Big Data analysis. Its MapReduce technique made it more useful for huge amout of data processing. Hadoop is incorporated with cloud computing. In this paper, we described about the introduction of Hadoop, MapReduce framework, application and real time users of Hadoop.
Keywords- MapReduce, Yarn, HDFS, Mapper, Reducer, Job Tasker, Task Tracker, Key, Values.

I. INTRODUCTION

The Hadoop framework is designed to provide a reliable, shared storage and analysis infrastructure to the user community. The storage portion of the Hadoop framework is provided by a distributed file system solution such as HDFS, while the analysis functionality is presented by *MapReduce*. Several other components are part of the overall Hadoop solution suite. The *MapReduce* functionality is designed as a tool for deep data analysis and the transformation of very large data sets. Hadoop enables the users to explore/analyze complex data sets by utilizing customized analysis scripts/commands. In other words, via the customized *MapReduce* routines, unstructured data sets can be distributed, analyzed, and explored across thousands of shared-nothing processing systems/clusters/nodes. Hadoop's HDFS replicates the data onto multiple nodes to safeguard the environment from any potential data-loss.

Apache Hadoop is a set of algorithms (an open-source software framework written in Java) for distributed storage and distributed processing of very large data sets (Big Data) on computer clusters built from commodity hardware. All the modules in Hadoop are designed with a fundamental assumption that hardware failures (of individual machines, or racks of machines) are commonplace and thus should be automatically handled in software by the framework.

The core hadoop consists of a storage part called Hadoop Distributed File System(HDFS) and a processing part MapReduce. It splits files into large blocks and distributes the blocks among nodes in the cluster. For processing the data, Hadoop Map/Reduce transfers the codes to node, then nodes process in parallel.

The base Apache Hadoop framework is composed of the following modules:

- *Hadoop Common* – contains libraries and utilities needed by other Hadoop modules;
- *Hadoop Distributed File System (HDFS)* – a distributed file-system that stores data on commodity machines, providing very high aggregate bandwidth across the cluster;
- *Hadoop YARN* – a resource-management platform responsible for managing compute resources in clusters and using them for scheduling of users' applications; and
- *Hadoop MapReduce* – a programming model for large scale data processing.

1.1 Hadoop Sub Projects

While Hadoop is in general best known for the MapReduce and the HDFS components, there are several other subprojects that as a unit, comprise the Hadoop offering:

- Hadoop Common: The common utilities that support the other Hadoop modules.

@IJMTER-2015, All rights Reserved

- HDFS: A distributed file system that provides high-throughput access to application data.
- Hadoop YARN: A framework for job scheduling and cluster resource management.
- Hadoop MapReduce: A YARN-based system for parallel processing of large data sets
- Avro: A data serialization system
- Cassandra: A scalable multi-master database with no single points of failure
- Chukwa: A data collection system for managing large distributed systems
- HBase: A scalable, distributed database that supports structured data storage for large tables
- Hive: A data warehouse infrastructure that provides data summarization and ad hoc querying
- Mahout: A Scalable machine learning and data mining library
- Pig: A high-level data-flow language and execution framework for parallel computation
- ZooKeeper: A high-performance coordination service for distributed applications.

Hadoop is basically designed to efficiently process very large data volumes by linking many commodity systems together to work as a parallel entity. In other words, the Hadoop philosophy is to bind many smaller (and hence more reasonably priced) nodes together to form a single, cost-effective compute cluster environment. It has to be pointed out that performing compute operations on large volumes of data in a distributed setup has obviously been done before. What separates Hadoop from the herd is the simplified programming model that allows users to rapidly develop and test distributed systems, as well as the efficient, automated distribution of data and work across the nodes (by effectively utilizing the underlying parallelism of the CPU cores).

II. MAPREDUCE FRAMEWORK

MapReduce is a programming model for processing and generating large data sets with a parallel, distributed algorithm on a cluster. Conceptually similar approaches have been very well known since 1995 with the Message Passing Interface [3] standard having reduce and scatter operations.

A MapReduce program is composed of a Map() procedure that performs filtering and sorting and a Reduce() procedure that performs a summary operation. The "MapReduce System" (also called "infrastructure" or "framework") orchestrates the processing by marshalling the distributed servers, running the various tasks in parallel, managing all communications and data transfers between the various parts of the system, and providing for redundancy and fault tolerance.

In a nutshell, the MapReduce programming Model is to process large datasets. The model itself is based on the concept of parallel programming. Based on the parallel programming notion, processing is decomposed into *n* sub-entities that are executed concurrently. The instructions for each sub-entity are executing simultaneously on different CPU's. Depending on the IT infrastructure setup, the CPU's may be available on the same server system or on remote nodes that are accessible via some form of interconnect/network. The MapReduce model itself is derived from the *map* and *reduce* primitives available in a functional language such as *Lisp*. A MapReduce job usually splits the input datasets into smaller chunks that are processed by the map task in a completely parallel manner. The outputs obtained from all the map tasks are then sorted by the framework and made available as input(s) into the reduce tasks. It has to be pointed out that a MapReduce model is normally suitable for long running batch jobs, as the *reduc*e function has to accumulate the results for the entire job, a process that is associated with overhead concerning the collection and submission of the processed datasets. In the MapReduce model, actual parallelism is achieved via a split/sort/merge/join process (see Figure 1) that can be described as:

- Initially, a master program (basically a central coordinator) is started by the MapReduce job that utilizes the predefined datasets as input.
- According to the Job Configuration, the master program initiates the *Ma*p and the *Reduce* programs on various systems. Next, it starts the input reader to retrieve the data from some directory (hosted in a distributed file system). The input reader then decomposes the data into small chunks and submits them to randomly chosen mapper programs. This process concludes the split phase and

@IJMTER-2015, All rights Reserved

initiates the parallel processing stage.

- After receiving the data, the mapper program executes a user supplied map function, and generates a collection of [key, value] pairs. Each produced item is sorted and submitted to the reducer.
- The reducer program collects all the items with the same key values and invokes a user supplied reduce function to produce a single entity as a result (this is labeled the merge phase).
- The output of the reduce program is collected by the output writer (effective join phase) and this process basically terminates the parallel processing phase.

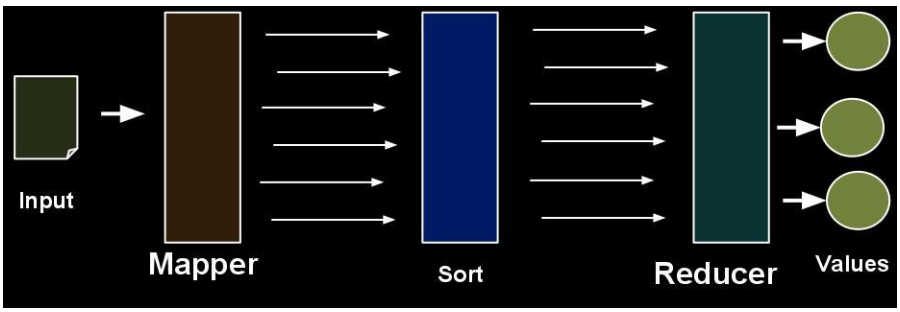

Fig 1: MapReduce Dataflow

2.1 Hadoop Map Reduce Architecture

The Hadoop MapReduce MRv1 framework is based on a centralized master/slave architecture. The architecture utilizes a single master server (*JobTracker*) and several slave servers (*TaskTracker's*). Please see Appendix A for a discussion on the MapReduce MRv2 framework. The JobTracker represents a centralized program that keeps track of the slave nodes, and provides an interface infrastructure for job submission. The TaskTracker executes on each of the slave nodes where the actual data is normally stored. In other words, the JobTracker reflects the interaction point among the users and the Hadoop framework. Users submit MapReduce jobs to the JobTracker, which inserts the jobs into the pending jobs queue and executes them on a FIFO basis (it has to be pointed out that other job schedulers are available - see Hadoop Schedulers below). The JobTracker manages the map and reduce task assignments with the TaskTracker's. The TaskTracker's execute the jobs based on the instructions from the JobTracker and handle the data movement between the *map* and *reduce* phases, respectively. Any map/reduce construct basically reflects a special form of a Directed Acyclic Graph (DAG).

A DAG can execute anywhere in parallel, as long as one entity is not an ancestor of another entity. In other words, parallelism is achieved when there are no hidden dependencies among shared states. In the MapReduce model, the internal organization is based on the map function that transforms a piece of data into entities of [key, value] pairs. Each of these elements is sorted (via their key) and ultimately reaches the same cluster node where a reduce function is used to merge the values (with the same key) into a single result (see code below). The Map/Reduce DAG is organized as depicted in Figure 2.

Map Function

```
map(input_record){
…
emit(k1,v1)
…
emit(k2,v2)
…
}
```

Reduce Function

```
reduce(key,values){
while(values.has_next){
aggregate=merge(values.next)
  }
collect(key,aggregate)
}
```

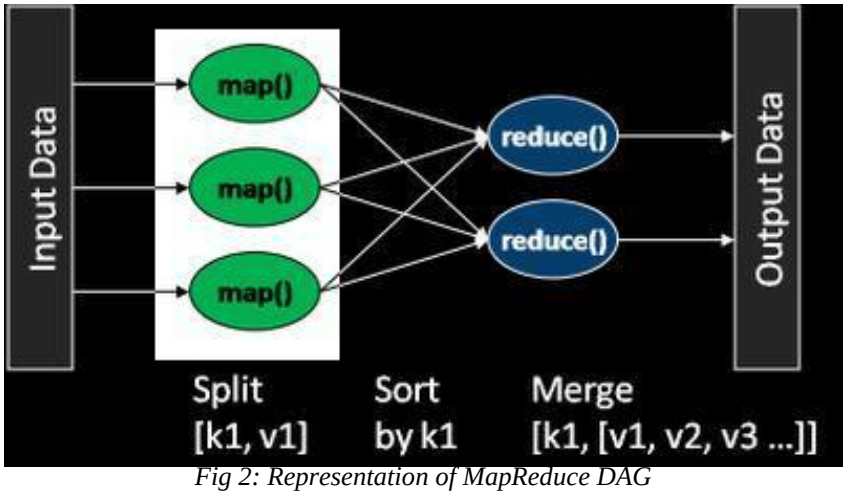

Fig 2: Representation of MapReduce DAG

The Hadoop MapReduce framework is based on a *pull* model, where multiple TaskTracker's communicate with the JobTracker requesting tasks (either map or reduce tasks). After an initial setup phase, the JobTracker is informed about a job submission. The JobTracker provides a job ID to the client program, and starts allocating map tasks to idle TaskTracker's requesting work items (see Figure 3). Each TaskTracker contains a defined number of task slots based on the capacity potential of the system. Via the heartbeat protocol, the JobTracker knows the number of free slots in the TaskTracker (the TaskTracker's send heartbeat messages indicating the free slots - true for the FIFO scheduler). Hence, the JobTracker can determine the appropriate job setup for a TaskTracker based on the actual availability behavior. The assigned TaskTracker will fork a *MapTask* to execute the map processing cycle (the MapReduce framework spawns 1 MapTask for each InputSplit generated by the InputFormat). In other words, the MapTask extracts the input data from the splits by using the RecordReader and InputFormat for the job, and it invokes the user provided map function, which emits a number of [key, value] pairs in the memory buffer.

After the MapTask finished executing all input records, the commit process cycle is initiated by flushing the memory buffer to the index and data file pair. The next step consists of merging all the index and data file pairs into a single construct that is (once again) being divided up into local directories. As some map tasks are completed, the JobTracker starts initiating the reduce tasks phase. The TaskTracker's involved in this step download the completed files from the map task nodes, and basically concatenate the files into a single entity. As more map tasks are being completed, the JobTracker notifies the involved TaskTracker's, requesting the download of the additional region files and to merge the files with the previous target file. Based on this design, the process of downloading the region files is interleaved with the on-going map task procedures.

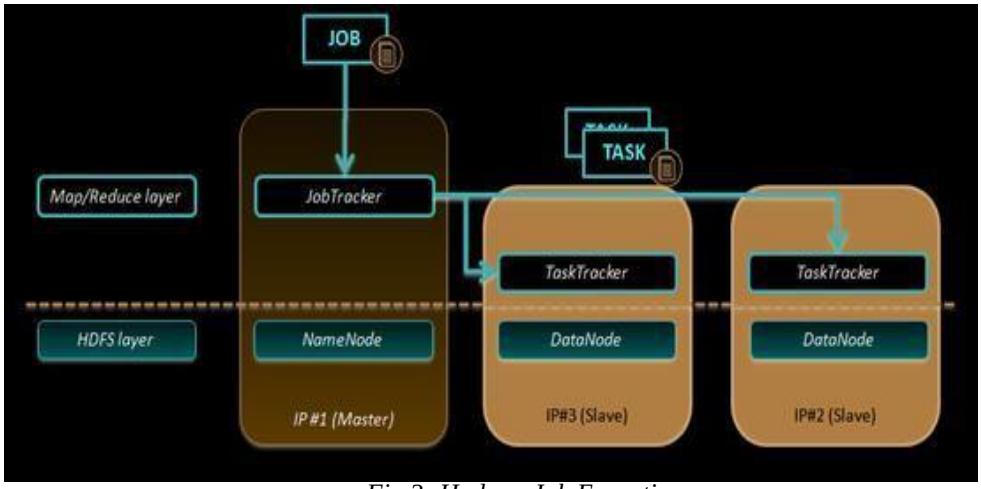

Fig 3: Hadoop Job Execution

Eventually, all the map tasks will be completed, at which point the JobTracker notifies the involved TaskTracker's to proceed with the reduce phase. Each TaskTracker will fork a *ReduceTask* (separate JVM's are used), read the downloaded file (that is already sorted by key), and invoke the reduce function that assembles the key and aggregated value structure into the final output file (there is one file per reducer node). Each reduce task (or map task) is single threaded, and this thread invokes the reduce [key, values] function in either ascending or descending order. The output of each reducer task is written to a temp file in HDFS. When the reducer finishes processing all keys, the temp file is atomically renamed into its final destination file name.

As the MapReduce library is designed to process vast amounts of data by potentially utilizing hundreds or thousands of nodes, the library has to be able to gracefully handle any failure scenarios. The TaskTracker nodes periodically report their status to the JobTracker that oversees the overall job progress. In scenarios where the JobTracker has not been contacted by a TaskTracker for a certain amount of time, the JobTracker assumes a TaskTracker node failure and hence, reassigns the tasks to other available TaskTracker nodes. As the results of the map phase are stored locally, the data will no longer be available if a TaskTracker node goes offline. In such a scenario, all the m*ap tasks* from the failed node (regardless of the actual completion percentage) will have to be reassigned to a different TaskTracker node that will re-execute *all* the newly assigned splits. The results of the reduce phase are stored in HDFS and hence, the data is globally available even if a TaskTracker node goes offline. Hence, in a scenario where during the reduce phase a TaskTracker node goes offline, only the set of incomplete r*educe tasks* have to be reassigned to a different TaskTracker node for re-execution.

Even within today's high-performance IT infrastructures, network bandwidth is still considered a relatively scarce resource. The Hadoop design aims at conserving network bandwidth by utilizing as much local (input) data as possible. In other words, the MapReduce master node takes the location information of the input files into account and attempts to schedule a map task on a system that already contains a replica of the corresponding input data. If such a scenario is not possible, an attempt is made to schedule the map task near a replica node, basically on a system that is located within the same network (switch) environment as the node containing the data. Hence, the design goal is to assure that most input data is available locally and hence, the read requests do not consume any network bandwidth (or at least as little as possible).

III. APPLICATIONS

The HDFS file system is not restricted to MapReduce jobs. It can be used for other applications, many of which are under development at Apache. The list includes the HBase database, the Apache Mahout machine learning system, and the Apache Hive Data Warehouse system. Hadoop can in theory be used for any sort of work that is batch-oriented rather than real-time, is very data-intensive, and benefits from parallel processing of data. It can also be used to complement a real-time system, such as lambda architecture.

Commercial applications of Hadoop included:

- Log and/or clickstream analysis of various kinds
- Marketing analytics
- Machine learning and/or sophisticated data mining
- Image processing
- Processing of XML messages
- Web crawling and/or text processing
- General archiving, including of relational/tabular data, e.g. for compliance

@IJMTER-2015, All rights Reserved

IV. USERS OF HADOOP

4.1 Twitter

With Hadoop, you can mine Twitter, Facebook and other social media conversations for sentiment data about you and your competition, and use it to make targeted, real-time, decisions that increase market share.

4.2 Yahoo!

On February 19, 2008, Yahoo! Inc. launched what it claimed was the world's largest Hadoop production application. The Yahoo! Search Webmap is a Hadoop application that runs on a Linux cluster with more than 10,000 cores and produced data that was used in every Yahoo! web search query.

There are multiple Hadoop clusters at Yahoo! and no HDFS file systems or MapReduce jobs are split across multiple datacenters. Every Hadoop cluster node bootstraps the Linux image, including the Hadoop distribution. Work that the clusters perform is known to include the index calculations for the Yahoo! search engine.

On June 10, 2009, Yahoo! made the source code of the version of Hadoop it runs in production available to the public. Yahoo! contributes all the work it does on Hadoop to the open-source community. The company's developers also fix bugs, provide stability improvements internally, and release this patched source code so that other users may benefit from their effort.

4.3 Facebook

In 2010 Facebook claimed that they had the largest Hadoop cluster in the world with 21 PB of storage. On June 13, 2012 they announced the data had grown to 100 PB. On November 8, 2012 they announced the data gathered in the warehouse grows by roughly half a PB per day.

4.4 Other Users

- Amazon/A9
- AOL
- Fox interactive media
- Google
- IBM
- New York Times
- PowerSet (now Microsoft)
- Quantcast
- Rackspace/Mailtrust
- Veoh
-

REFERENCES

[1]. "Hadoop Releases". *apache.org*. Apache Software Foundation. 2014-12-06.
[2]. "What is the Hadoop Distributed File System (HDFS)?". *ibm.com*. IBM, 2014-10-30.
[3]. “Data Science and Big Data Analytics: Discovering, Analyzing, Visualizing and Presenting Data”. John Wiley & Sons. 2014-12-19. p. 300. ISBN 9781118876220, 2015-01-29.
[4]. Ashlee Vance (2009-03-17). "Hadoop, a Free Software Program, Finds Uses Beyond Search". *The New York Times*, 2010-01-20.
[5]. Dean, Jeffrey & Ghemawat, Sanjay (2004). "MapReduce: Simplified Data Processing on Large Clusters", November 23, 2011.
[6]. Matt Williams (2009). "Understanding Map-Reduce", April 13, 2011.
[7]. Apache Hadoop. (2012). [Online]. Available: http://hadoop.apache.org/mapreduce
[8]. “Hadoop”, https://hadoop.apache.org › Hadoop › hadoop-mapreduce-client

@IJMTER-2015, All rights Reserved