Apache Hadoop

An opensource software framework for distributed applications

Purshottam Vishwakarma
M.S. Computer Science
Indiana University
pvishwak@indiana.edu

***Abstract*— We have stepped into era which is facing data deluge like never before especially after the inception of web 2.0 and social media like Facebook and twitter. Thousands of terabytes of data is generated every day and it becomes increasingly important for us to store, retrieve and analyze this data to make sense of it or to perform data-intensive task like search. This problem has been addressed by Hadoop to a large extent. Hadoop is an open source software framework to run distributed applications in parallel which is being increasingly adopted by plethora of academic institutions and commercial organizations. In this survey paper, we provide an insight into its inception, core concepts, architecture, evolution and implementation. We also discuss MapReduce programming model, HDFS file system, scheduling algorithms and describe how Hadoop is being integrated with different applications to solve problems involving large data sets.**

***Keywords*- Google File System(GFS), HBase, HDFS, MapReduce, Scheduling.**

I. INTRODUCTION

While the data on the web was exponentially on the rise, Google faced a unique challenge of indexing and searching through this whopping amount of data efficiently. This lead to the introduction of technologies by Google called MapReduce and Google File System (GFS) in 2004 which efficiently supported distributed computing on large clusters of computers. Inspired by Google's MapReduce and GFS, Doug Cutting created Hadoop to support his Nutch project, which itself came from the Lucene search engine library. Later Hadoop became the top-level Apache project. Though Google pioneered this architecture the fact Hadoop is open source means that anyone can use it or improve it. Yahoo! has been the greatest contributor to Hadoop and is also probably its largest user. Facebook, Amazon, IBM, NetFlix too are few of its many prime users and highly influencing top corporations like Microsoft to adopt it in their suite of technologies.

II. ARCHITECTURE

Hadoop implements a master/slave model. Each Hadoop cluster has a single master node called jobtracker and several slave nodes called tasktrackers. Users submit map/reduce jobs to the jobtracker which puts them in a queue of pending jobs and executes them as first-come/first-served basis. The jobtracsker then distributes the job to the tasktrackers for execution. For efficient division of tasks the file system provides location awareness i.e. name of the node where the tasktracker is running. Hadoop uses this information to assign task to the tasktracker which is closest to the data thereby reducing network traffic.

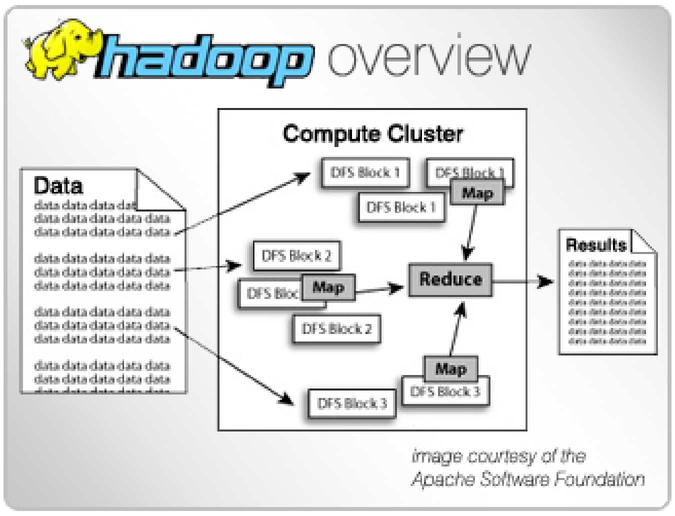

Fig.1: A multi-code Hadoop Cluster

A. *Hadoop MAP/Reduce execution Framework*

Map/Reduce is a programming model pioneered by Google specifically implemented for large data sets. It uses two phases *map* and *reduce* based on functional language like lisp.

- *Map phase*: The master node splits the data set in the form of key-value pairs into large number of fragments and assigns each fragment to a map task. This map task can further divide into multiple fragments assigning to other map tasks thus creating a hierarchy. Each *map* task invokes a map function which takes data as key/value pairs (K,V), performs some operation to produce zero or more intermediate key/value pairs (K’,V’) and passes on the answer to back to the master node (jobtracker).

- *Reduce phase*: The master node then collects the results from all the tasktrackers and combines them to form the output.

Map and Reduce operations can be performed independently provided there is no dependence between map tasks, allowing for parallel processing giving the user an extremely fast response. After map phase the framework sorts the intermediate data set by key and produces a set of (K’, V’*) tuples so that all the values associated with a particular key appear together. It also partitions the set of tuples into a number of fragments equal to the number of reduce tasks. If any of the tasktracker fails, the assigned task can be divided among the remaining running tasktrackers. If the task takes too long, they send a signal periodically to the master node with its status. If any of the worker nodes falls silent for a long time, the node is declared as dead and distributes the assigned work among other worker nodes.

MapReduce implementations are not necessarily reliable as the master node is a single point of failure. MayReduces’s stable inputs and outputs are usually stored in a distributed file system. The transient data is usually stored on local disk and fetched remotely by the reducers.

B. Hadoop Distributed File System

Hadoop relies on Google File system, but Hadoop Distributed file system (HDFS). It is scalable, distributed and portable filesystem written in Java. It spans multiple computers and runs in user space. HDFS assumes that hardware is unreliable and will eventually fail. Hence it replicates data blocks across multiple machines in the cluster allowing fault tolerance. HDFS also assumes that the data will be written only once and is able to gain extra performance by optimizing for subsequent reads while disallowing subsequent writes. Because moving large volumes of data between nodes is extremely expensive, HDFS moves the computation to the nodes where the data resides. HDFS has been designed to be easily portable from one platform to another. This facilitates widespread adoption of HDFS as a platform of choice for a large set of applications. HDFS is said to be "rack aware" meaning that it can be configured to know about the proximity of machines to one another and replicate data near the nodes which might need it. The logical unit of HDFS is called data block. The default block size is 64MB. The data comprising larger files will be spread across multiple data blocks. HDFS is capable of replicating the data blocks across machines. The default replication value is 3 i.e. data is stored on three nodes: 2 on the same rack and 1 on a different rack. The block size and replication factor are configurable on a per file basis. An application can specify the number of replicas of a file. The replication factor can be specified at file creation time and can be changed later. Files in HDFS are write-once and have strictly one writer at any time.

HDFS defines two types of nodes: *NameNode* and *DataNode*. DataNode is the storage node and responsible for low-level operations including block creation, deletion, reads and writes. A *NameNode* keeps track of which DataNodes have which blocks of data and uses this information to manage the hierarchy of the overall file system. Each cluster will have exactly one NameNode, but will have many DataNodes. One machine may fill several of these roles simultaneously in a very small Hadoop installation. There is a secondary NameNode for checkpointing which continuously takes snapshots of the primary NameNode’s directory information which is then saved to local or remote directories. Data nodes can communicate with each other to rebalance data, move copies of data around the cluster and to keep the replication of data high. One of the limitations of HDFS is that it cannot be directly mounted by an existing operating system.

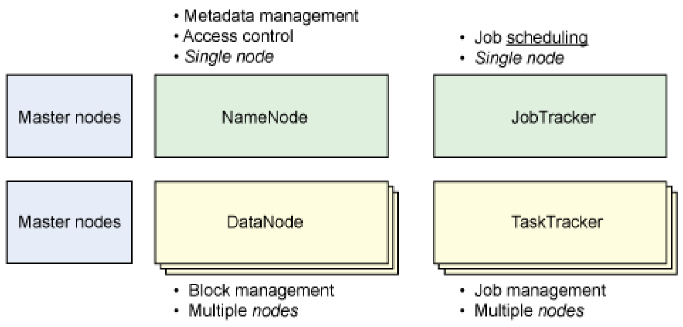

Fig 2: Elements of a Hadoop Cluster

III. SCHEDULING IN HADOOP

Since Hadoop creates multiple jobs to be run on multiple nodes, it can exploit the opportunity of using efficient scheduling algorithms to more optimally map jobs to resources in a way that optimizes their use. Hadoop uses pluggable scheduling algorithms which can be tailored to the application it is being used for and is open to greater experimentation. This pluggable scheduler framework allows optimal use of a Hadoop cluster over a varied set of workloads with varying priority and performance constraints. Here we discuss two of the algorithms used today (fair scheduling and capacity scheduling) and under what scenarios they are relevant.

A. FIFO Scheduling

This is the original scheduling algorithm incorporated in Hadoop in its early days. It is integrated within the jobtracker. When the user submits job it is inserted into a queue. The jobtracker pulls jobs from this queue, oldest job first. This traditional algorithm was simple to implement and had no consideration for priority and size of the job.

B. Fair Scheduling

The central idea behind designing the fair scheduler was to assign resources to jobs such that on average over time, each job gets an equal share of the available resources. The result is that jobs that require less time are able to execute and finish quickly. Jobs that require more time also get enough time to execute on the CPU preventing starvation. This behavior allows for some interactivity among Hadoop jobs and improves the response time of the cluster as a whole. Fair scheduling arose out of Facebook's need to share its data warehouse between multiple users.

The Hadoop implementation creates a set of pools into which jobs are placed for selection by the scheduler. Each pool can be assigned a set of shares to balance resources across jobs in pools. By default, all pools have equal shares, but configuration is possible to provide more or fewer shares depending upon the job type. The number of jobs active at one time can also be constrained, if desired, to minimize congestion and allow work to finish in a timely manner. The scheduler also includes a number of features for ease of administration, including the ability to reload the config file at runtime to change pool settings without restarting the cluster, limits on running jobs per user and per pool, and use of priorities to weigh the shares of different jobs.

C. Capacity Scheduling

Capacity scheduling, developed at Yahoo, was defined for large clusters, which may have multiple, independent consumers and target applications. For this reason, capacity scheduling provides greater control as well as the ability to provide a minimum capacity guarantee and share excess capacity among users. In capacity scheduling several queues are created each with a configurable number of map and reduce slots. The difference between the capacity and fair scheduler is that each queue is assigned with a guaranteed capacity. The scheduler monitors the queues continuously to determine if the allocated capacity is utilized. If any queue is not consuming its allocated capacity, this excess capacity is temporarily allocated to another queue which might need it. Thus any available capacity is redistributed for use by other users.

Another difference with respect to fair scheduling is assigning priority to the jobs within each queue wherein jobs with higher priority will have access to resources sooner than the low priority jobs. Capacity scheduler also implements access control mechanism across queues which restrict the ability to submit jobs to queues and the ability to view and modify jobs in queues. Multiple Hadoop configuration files are used to configure capacity scheduling. Individual queue properties include capacity percentage (where the capacity of all queues in the cluster is less than or equal to 100), the maximum capacity (limit for a queue's use of excess capacity), and whether the queue supports priorities. Most importantly, these queue properties can be manipulated at run time, allowing them to change and avoid disruptions in cluster use.

When to use each scheduler?

If the cluster size is large, with multiple clients and different types and priorities of jobs, then the capacity scheduler would be the ideal choice to ensure guaranteed access with the potential to reuse unused capacity and prioritize jobs within queues. If the workload is limited then fair scheduler works well with both small and large clusters. The fair scheduler is useful in the presence of diverse jobs, because it can provide fast response times for small jobs mixed with larger jobs.

IV. HBASE

As HDFS was inspired by Google File System, similarly HBase was inspired by Google's Bigtable. HBase is a NoSQL, distributed and scalable Database running over HDFS. HBase is used when we need random, realtime read/write access to very large Data, perhaps with hundreds of millions or billions of rows. With few thousand/millions of rows, using a traditional RDBMS would be a wiser choice. HDFS is a general purpose file system, and does not provide fast individual record lookups in files. HBase, on the other hand, is built on top of HDFS and provides fast record lookups (and updates) for large tables. HBase hosts very large tables over the Hadoop Cluster. Facebook elected to implement its new messaging platform using HBase in November 2010. HBase has many features which supports both linear and modular scaling. HBase clusters expand by adding RegionServers that are hosted on commodity class servers. If a cluster expands from 10 to 20 RegionServers, for example, it doubles both in terms of storage and as well as processing capacity.

V. HADOOP API

Hadoop provides an API for programming with the MapReduce Model. There are 3 main components to a MapReduce Job: the Mapper, Reducer and the JobConf.

The *JobConf* describes the pertinent details of the job itself, including its name, the data types for the key and value, the class to use for the map and reduce functions, and information about the input and output data used for the job.

The *Mapper* class does the work described by the map function. It has a single method map which is passed a key, a value, an OutputCollector for accepting the key/value pairs created as output, and a Reporter which is used for indicating progress. The key and value are both defined using generics and the key type generally implements the Comparable interface. The Hadoop API defines the Writable interface, typically implemented by both the key and the value, which allows Hadoop to gain extra performance by using a custom serialization protocol. To aid in implementing both the Comparable and Writable interfaces, Hadoop also provides the WritableComparable interface along with a number of convenience classes which implement it for common data types.

The *Reducer* class does the work described by the reduce function. It has a single method reduce which is passed a key, an Iterator which supplies values, an OutputCollector for gathering output pairs, and a Reporter for indicating progress. Like the Mapper, the key and value types are defined using generics and the key typically implements the Comparable interface. The values provided by the Iterator typically implement both Comparable and Writable.

VI. INTEGRATING HADOOP WITH OTHER APPLICATIONS

Hadoop is primarily used for two classes of applications: *advanced analytics* and *data processing*. Organizations are putting tremendous work in integrating Hadoop into their day-to-day enterprise applications to solve business problems. Ortbitz, a travel company, has implemented distributed data analysis with Hadoop and R to analyze the travel bookings (R is an open source statistical package with visualization). VMware introduced Spring Hadoop which integrates the Spring Framework and the Apache Hadoop platform to make it easy for enterprise developers to build distributed processing solutions with Apache Hadoop. Analytics software provider SAS has increased access to big data sources with Hadoop support in its updated SAS Enterprise Data Integration Server. Now Oracle Big Data Connectors product can be used to integrate data stored in Hadoop and Oracle NoSQL Database with Oracle Database 11g. It enables data analysis using Oracle's distribution of open source R analysis directly on Hadoop data.

VII. CONCLUSION

Apache Hadoop is an exciting project which makes a low-cost, high-performance architecture available to everyone. This survey paper introduced the core concepts of MapReduce and the Hadoop infrastructure, how companies are using it to solve business problems. Although Hadoop may not be appropriate for solving every problem, it's certainly worth considering when we need a scalable and reliable batch-oriented approach for processing large volumes of data.

VIII. ACKNOWLEDGEMENT

We would like to thank Dr. Judy Qiu for her guidance and assistance to choose and find relevant material on Hadoop project. This exercise has given me deep insight and provided the opportunity to explore further into the arena of distributed systems. I would further like to thank all the Associate Instructors and fellow students in the B534 class at IU, Bloomington for their valuable discussion and thoughts into this survey.

REFERENCES

[1] Apache hadoop Project. http://hadoop.apache.org/

[2] Hadoop Documentation for HDFS. http://hadoop.apache.org/common/docs/r0.20.2/index.html

[3] Exploring Scalable Data Processing with Apache Hadoop http://jnb.ociweb.com/jnb/jnbNov2008.html

[4] http://en.wikipedia.org/wiki/Apache_Hadoop.

[5] MapReduce programming with Apache Hadoop http://www.javaworld.com/javaworld/jw-09-2008/jw-09-hadoop.html?page=1.

[6] Hands-on Hadoop for cluster computing. http://archive09.linux.com/feature/150395

[7] Scheduling in Hadoop. http://www.ibm.com/developerworks/linux/library/os-hadoop-scheduling/index.html?ca=drs-#resources

[8] Job Scheduling in Hadoop. http://www.cloudera.com/blog/2008/11/job-scheduling-in-hadoop/

[9] Apache HBase Project. http://hbase.apache.org/.

[10] http://en.wikipedia.org/wiki/HBase

[11] http://www.infoq.com/presentations/Distributed-Data-Analysis-with-Hadoop-and-R