Lesson 2: Brief Review of Hadoop

2.1. Introduction

Hadoop is an open source framework that supports the processing of large data sets in a distributed computing environment. Hadoop consists of MapReduce, the Hadoop distributed file system (HDFS) and a number of related projects such as Apache Hive, HBase and Zookeeper. MapReduce and Hadoop distributed file system (HDFS) are the main component of Hadoop.

Apache Hadoop is an open-source, free and Java based software framework offers a powerful distributed platform to store and manage Big Data. It is licensed under an Apache V2 license. It runs applications on large clusters of commodity hardware and it processes thousands of terabytes of data on thousands of the nodes. Hadoop is inspired from Google's MapReduce and Google File System (GFS) papers.

The major advantage of Hadoop framework is that it provides reliability and high availability

2.1.1. Core Hadoop Components

There are two major components of the Hadoop framework and both of them does two of the important
task for it.

1. Hadoop MapReduce is the method to split a larger data problem into smaller chunk and distribute it to many different commodity servers. Each server have their own set of resources and they have processed them locally. Once the commodity server has processed the data they send it back collectively to main server. This is effectively a process where we process large data effectively and efficiently
2. Hadoop Distributed File System (HDFS) is a virtual file system. There is a big difference between any other file system and Hadoop. When we move a file on HDFS, it is automatically split into many small pieces. These small chunks of the file are replicated and stored on other servers (usually 3) for the fault tolerance or high availability.
3. Namenode: Namenode is the heart of the Hadoop system. The NameNode manages the file system namespace. It stores the metadata information of the data blocks. This metadata is stored permanently on to local disk in the form of namespace image and edit log file. The NameNode also knows the location of the data blocks on the data node. However the NameNode does not store this information persistently. The NameNode creates the block to DataNode mapping when it is restarted. If the NameNode crashes, then the entire Hadoop system goes down. Read more about Namenode
4. Secondary Namenode: The responsibility of secondary name node is to periodically copy and merge the namespace image and edit log. In case if the name node crashes, then the namespace image stored in secondary NameNode can be used to restart the NameNode.
5. DataNode: It stores the blocks of data and retrieves them. The DataNodes also reports the blocks information to the NameNode periodically.
6. Job Tracker: Job Tracker responsibility is to schedule the client's jobs. Job tracker creates map and reduce tasks and schedules them to run on the DataNodes (task trackers). Job Tracker also checks for any failed tasks and reschedules the failed tasks on another DataNode. Job tracker can be run on the NameNode or a separate node.
7. Task Tracker: Task tracker runs on the DataNodes. Task trackers responsibility is to run the map or reduce tasks assigned by the NameNode and to report the status of the tasks to the NameNode.

2.2. MapReduce and Relational Databases

Why can't we use databases with lots of disks to do large-scale batch analysis? Why is MapReduce needed?

The answer to these questions comes from another trend in disk drives: seek time is improving more slowly than transfer rate. Seeking is the process of moving the disk's head to a particular place on the disk to read or write data. It characterizes the latency of a disk operation, whereas the transfer rate corresponds to a disk's bandwidth.

If the data access pattern is dominated by seeks, it will take longer to read or write large portions of the dataset than streaming through it, which operates at the transfer rate. On the other hand, for updating a small proportion of records in a database, a traditional B-Tree (the data structure used in relational databases, which is limited by the rate it can perform seeks) works well. For updating the majority of a database, a B-Tree is less efficient than MapReduce, which uses Sort/Merge to rebuild the database.

In many ways, MapReduce can be seen as a complement to an RDBMS. (The differences between the two systems. MapReduce is a good fit for problems that need to analyze the whole dataset, in a batch fashion, particularly for ad hoc analysis. An RDBMS is good for point queries or updates, where the dataset has been indexed to deliver low-latency retrieval and update times of a relatively small amount of data. MapReduce suits applications where the data is written once,and read many times, whereas a relational database is good for datasets that are continually updated.

Another difference between MapReduce and an RDBMS is the amount of structure in the datasets that they operate on. Structured data is data that is organized into entities that have a defined format, such as XML documents or database tables that conform to a particular predefined schema. This is the realm of the RDBMS. Semi-structured data, on the other hand, is looser, and though there may be a schema, it is often ignored, so it may be used only as a guide to the structure of the data: for example, a spreadsheet, in which the structure is the grid of cells, although the cells themselves may hold any form of data. Unstructured data does not have any particular internal structure: for example, plain text or image data. MapReduce works well on unstructured or semi- structured data, since it is designed to interpret the data at processing time. In other words, the input keys and values for MapReduce are not an intrinsic property of the data, but they are chosen by the person analyzing the data.

Relational data is often normalized to retain its integrity and remove redundancy. Normalization poses problems for MapReduce, since it makes reading a record a non- local operation, and one of the central assumptions that MapReduce makes is that it is possible to perform (high-speed) streaming reads and writes.

A web server log is a good example of a set of records that is not normalized (for example, the client hostnames are specified in full each time, even though the same client may appear many times), and this is one reason that logfiles of all kinds are particularly well-suited to analysis with MapReduce.

MapReduce is a linearly scalable programming model. The programmer writes two functions— a map function and a reduce function—each of which defines a mapping from one set of key value pairs to another. These functions are oblivious to the size of the data or the cluster that they are operating on, so they can be used unchanged for a small dataset and for a massive one.

More important, if you double the size of the input data, a job will run twice as slow. But if you also double the size of the cluster, a job will run as fast as the original one. This is not generally true of SQL queries.

Over time, however, the differences between relational databases and MapReduce systems are likely to blur—both as relational databases start incorporating some of the ideas from MapReduce (such as Aster Data's and Greenplum's databases) and, from the other direction, as higher-level query languages built on MapReduce (such as Pig and Hive) make MapReduce systems more approachable to traditional database programmers.

2.3. Analyzing data with Hadoop

To take advantage of the parallel processing that Hadoop provides, we need to express our query as a MapReduce job. After some local, small-scale testing, we will be able to run it on a cluster of machines.

2.3.1. Map And Reduce

MapReduce works by breaking the processing into two phases: the map phase and the reduce phase. Each phase has key-value pairs as input and output, the types of which may be chosen by the programmer. The programmer also specifies two functions: the map function and the reduce function.

The input to our map phase is the raw NCDC data. We choose a text input format that gives us each line in the dataset as a text value. The key is the offset of the beginning of the line from the beginning of the file, but as we have no need for this, we ignore it.

Our map function is simple. We pull out the year and the air temperature, since these are the only fields we are interested in. In this case, the map function is just a data preparation phase, setting up the data in such a way that the reducer function can do its work on it: finding the maximum temperature for each year. The map function is also a good place to drop bad records: here we filter out temperatures that are missing, suspect, or erroneous.

To visualize the way the map works, consider the following sample lines of input data (some unused columns have been dropped to fit the page, indicated by ellipses):

```
0067011990999991950051507004...9999999N9+00001+99999999999...
0043011990999991950051512004...9999999N9+00221+99999999999...
0043011990999991950051518004...9999999N9-00111+99999999999...
0043012650999991949032412004...0500001N9+01111+99999999999...
0043012650999991949032418004...0500001N9+00781+99999999999...
```

These lines are presented to the map function as the key-value pairs:

```
(0, 0067011990999991950051507004...9999999N9+00001+99999999999...)
(106, 0043011990999991950051512004...9999999N9+00221+99999999999...)
(212, 0043011990999991950051518004...9999999N9-00111+99999999999...)
(318, 0043012650999991949032412004...0500001N9+01111+99999999999...)
(424, 0043012650999991949032418004...0500001N9+00781+99999999999...)
```

The keys are the line offsets within the file, which we ignore in our map function. The map function merely extracts the year and the air temperature (indicated in bold text), and emits them as its output (the temperature values have been interpreted as integers):

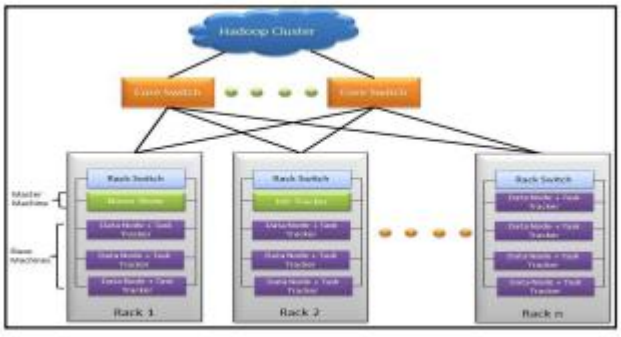
(1950, 0)
(1950, 22)
(1950, −11)
(1949, 111)
(1949, 78)
The output from the map function is processed by the MapReduce framework before being sent to the reduce function. This processing sorts and groups the key-value pairs by key. So, continuing the example, our reduce function sees the following input:

(1949, [111, 78])
(1950, [0, 22, −11])
Each year appears with a list of all itsair temperature readings. All the reduce function has to do now is iterate through the list and pick up the maximum reading:

(1949, 111)
(1950, 22)
This is the final output: the maximum global temperature recorded in each year.

2.4. Hadoop Cluster Physical Architecture

Hadoop cluster is a special type of computational cluster designed for storing and analyzing vast amount of unstructured data in a distributed computing environment. These clusters run on low cost commodity computers. Hadoop clusters are often referred to as "shared nothing" systems because the only thing that is shared between nodes is the network that connects them. Large Hadoop Clusters are arranged in several racks. Network traffic between different nodes in the same rack is much more desirable than network traffic across the racks.
A Real Time Example: Yahoo's Hadoop cluster. They have more than 10,000 machines running Hadoop and nearly 1 petabyte of user data.
A small Hadoop cluster includes a single master node and multiple worker or slave node. As discussed earlier, the entire cluster contains two layers. One of the layer of MapReduce Layer and another is of HDFS Layer. Each of these layer have its own relevant component. The master node consists of a JobTracker, TaskTracker, NameNode and DataNode. A slave or worker node consists of a DataNode and TaskTracker. It is also possible that slave node or worker node is only data or compute node. The matter of the fact that is the key feature of the Hadoop.



2.5. Hadoop Limitations

Despite Hadoop being able to process large datasets, it has some limitations:

i. Network File system is the oldest and the most commonly used distributed file system and was designed for the general class of applications, Hadoop only specific kind of applications can make use of it.

ii. It is known that Hadoop has been created to address the limitations of the distributed file system, where it can store the large amount of data, offers failure protection and provides fast access, but it should be known that the benefits that come with Hadoop come at some cost.

iii. Hadoop is designed for applications that require random reads; so if a file has four parts the file would like to read all the parts one-by-one going from 1 to 4 till the end. Random seek is where you want to go to a specific location in the file; this is something that isn't possible with Hadoop. Hence, Hadoop is designed for non- real-time batch processing of data.

iv. Hadoop is designed for streaming reads caching of data isn't provided. Caching of data is provided which means that when you want to read data another time, it can be read very fast from the cache. This caching isn't possible because you get faster access to the data directly by doing the sequential read; hence caching isn't available through Hadoop.

v. It will write the data and then it will read the data several times. It will not be updating the data that it has written; hence updating data written to closed files is not available. However, you have to know that in update 0.19 appending will be supported for those files that aren't closed. But for those files that have been closed, updating isn't possible.

vi. In case of Hadoop we aren't talking about one computer; in this scenario we usually have a large number of computers and hardware failures are unavoidable; sometime one computer will fail and sometimes the entire rack can fail too. Hadoop gives excellent protection against hardware failure; however the performance will go down proportionate to the number of computers that are down. In the big picture, it doesn't really matter and it is not generally noticeable since if you have 100 computers and in them if 3 fail then 97 are still working. So the proportionate loss of performance isn't that noticeable. However, the way Hadoop works there is the loss in performance. Now this loss of performance through hardware failures is something that is managed through replication strategy.

Lesson 2: Review Questions

1. State four advantages of using Hadoop.
2. Discuss brief history of Hadoop.
3. Explain three components of Hadoop Cluster, illustrate with diagrams.
4. Discuss work-flow of How File is Stored in Hadoop.