A Map Reduce Implementation on Open Source Platform: EUCALYPTUS

Nilesh Mangtani
Master of Engg. IInd Year
Computer Dept.
MITCOE, Pune

Jyoti B. Rathi
Assistant Professor
Department of CSE
J.D.I.E.T, Yavatmal

ABSTRACT

Cloud computing is one of the important emerging technologies now-a-days. In recent years many of the applications are developed by using the Cloud computing. It mainly works by using the clusters of all the available resources in an organization or a company. Also recently Hadoop framework has also emerged which work in the distributed environments only. Hadoop being a open-source is used by many companies recently. In this paper, we have tried to propose a solution of merging the Hadoop technology with the cloud by using a open-source platform EUCALYPTUS. Since both of the above platforms are open source many of the companies can earn more profit by integrating with them. In this case the MapReduce an important part of Hadoop is being discussed and is tried to merge out with the Cloud by using EUCALYPTUS. MapReduce is a programming model that is developed by Google but widely used by Hadoop. Thus in this paper we have discussed few of scenarios where Hadoop can fails and also proposed the solution of those by using the Cloud technology.

General Terms

Integration of Hadoop with Cloud by using open-source cloud platform EUCALYPTUS

Keywords

Hadoop, MapReduce, Cloud Computing, EUCALYPTUS

1. INTRODUCTION

A cloud service can outsource management, maintenance and administration of large clusters of servers for any company, organization or even a private person still providing the benefits. The infrastructure to control and maintain the cloud can be proprietary like Microsoft Hyper-V Cloud [1], VMware vCloud [2] and Citrix Open Cloud [3], but there are also a number of free and open-source solutions like Eucalyptus Cloud, OpenNebula [4] and CloudStack [5]. In recent years, cloud computing and the services it provides were deploying rapidly.

An Apache open source distributed computing framework is Hadoop. It has been applied in many sites such as Amazon, Facebook, and Yahoo and so on. It takes the advantage of the power of clusters, with high-speed computing with scalability across a large cluster. It works in parallel and thus it speedup the processing speed. The most important part of this framework is MapReduce and HDFS. Although it is meant to run on the dedicated servers, but there is no problem to run it on the virtual machine. Before the application of MapReduce, Yahoo! Took 26 days of processing to build automatic completion of indexes for their search engines which was reduced to just 20 minutes by applying the MapReduce model with a cluster of computing nodes [6]. It was developed by Google for processing a very large amount of data to generate the useful information. By hiding the details of distributed computing system the complexity of distributed programming is reduced. An example is WordCounter [7]: an application that counts the number of times a word appears in a collection of documents. Map emits a key/value pair, (word, 1). Reduce combine the value from the same word and we can get the number of times a word appears. MapReduce computation is used in many applications [8-10] such as Distributed Grep, Count of URL Access Frequency, Distributed Sort, Security Enhanced DNS Group, and other real world applications. The most important is Google still uses it in its new search engine technology, Percolator [11] to analyze and produce the initial search index. Also the user must know that it is not easy to implement the MapReduce model. If any user wants to use it, they have to set up their machine's MapReduce environment first to compute a large-scale data set. The code for the MapReduce program is nearly impossible without suitable background and training.

In this paper, section 2 describes a detailed introduction about the Hadoop framework. Further in section 3 the description about the open source platform EUCALYPTUS is provided. Finally in section 4 the problem statement and solution is provided which is followed by future work and conclusion.

2. HADOOP FRAMEWORK

Hadoop is an open source software package from the Apache foundation which is used for batch processing of large data sets on a physical cluster of machines. It contains many different systems that are aimed at file storage, analysis and processing of large amounts of data ranging from few Gigabytes to several Petabytes. It contains a distributed file system called Hadoop Distributed File System (HDFS), a Common set of commands, scheduler, and the MapReduce evaluation framework. The HDFS and MapReduce are the most important part of the Hadoop which are detailed analyzed in the next section. Hadoop is written in Java which contains around 3,00,000 lines of code (LOC). The reason behind such a huge code is that it contains the entire complexity of cluster management, redundancy in HDFS, consistency and reliability in case of node failure is included in the framework itself. Hadoop is popular for processing huge data sets, especially in social networking, targeted advertisements, internet log processing etc.

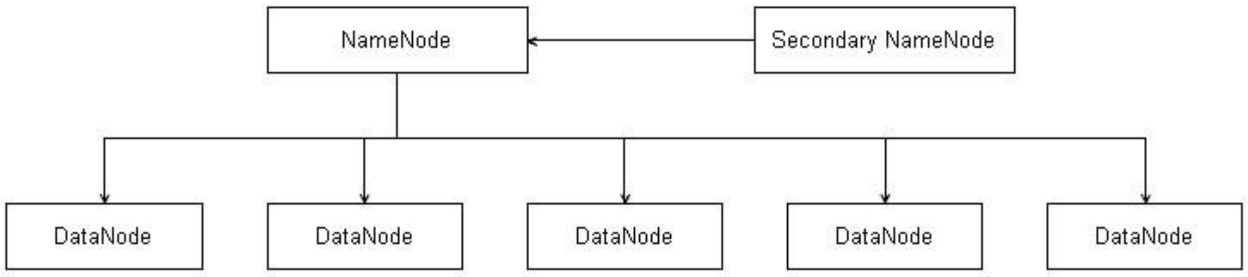

Figure 1: Hadoop Distributed File System

2.1 HDFS

HDFS is designed to be a filesystem to store large amounts of data across a distributed system of computers that gives a fast access rate and reliability for very large datasets. As mentioned earlier also it is written in Java which is used to communicate with other instances that are on the networked of HDFS through RPC to store blocks of data across cluster. It has a slower access rate and higher latency because it focuses more on delivering high amount of data between physical machines [12]. It is designed to work with files which can vary from gigabytes to hundreds or thousands of Petabytes.

As shown in the Figure 1, HDFS contains three important software parts. The first is the NameNode which is also called as "master" of the filesystem. The main function of this is to keep records of where and how files are stored in the system i.e. it maintains the file system and the tree index of all files and directories. HDFS stores files in its filesystem in the form of blocks. It splits the original data in blocks of any configurable size (defined in the NameNode configuration) but the default size is 64 MB. This is compared to a normal disk block which is 512 bytes [12]. After splitting the datafile in different blocks, each block is sent to different DataNodes or rather we can say to multiple DataNodes which will provide redundancy and higher throughput when another system requests access to the file. The DataNode is the second part of HDFS which is also called as "slave" in the system. The metadata about each file like which original datafile it belongs, its relation to other blocks is stored on-disk in NameNode. It is stored in form of two files: namespace image and edit log. The perfect block locations on the DataNodes are always rebuild on startup by communicating with DataNodes [12].

Since NameNode keeps tracks of tree structure and metadata of the files of the whole filesystem it is also a single point of failure. If it breaks down the whole HDFS will be invalid even though if the DataNodes are present since DataNodes doesn't have any information about the structure. The third part Secondary NameNode also cannot work without NameNode. The function of the secondary NameNode is only responsible for validating the namespace of NameNode. The NameNode and secondary NameNode should be different machines (and separated from DataNodes) on a large system [12].

By default the NameNode keep three copies of each block on different DataNodes to provide both redundancy and more throughputs for the client. This value can be increased or decreased but keeping minimum three is consider to be good. Also to provide better redundancy and better throughput HDFS is rack-aware. It wants to know which rack each node resides in and how "far" in terms of bandwidth which can help the NameNode to keep more copies of blocks on one rack for faster throughput. Also the additional copies are always kept on other rack to provide better redundancy.

When a user wants to read any file from the HDFS it first contacts the NameNode. Then the NameNode provides the blocks locations and inform to clients. This forces the client to read and merge the blocks into one file from DataNodes. However this task is not as easy to read from an operating system since there are bindings to HTTP and FTP.

2.2 MapReduce

MapReduce is a programming model for processing and analyzing very large datasets in a fast, scalable and distributed way using distributed computing on a computer cluster. It is invented and patented by Google. MapReduce has the advantage of handling large data sets, so it is suitable for cloud computing platform [13]. The word MapReduce is derives from two typical functions used within functional programming, the Map and Reduce functions [14]. Through a license from Google, Hadoop has taken this framework and implemented it to run on top of cluster of computers. For utilizing the combined resources of a large cluster of any commodity hardware Hadoop MapReduce is mainly used. HDFS is a file system where it relies although it supports few more distributed filesystem.

MapReduce main job is to divide the data into many logic blocks such that the programs can process on distributed clusters in parallel. Its input data set is a set of key/value pairs and also output is same. The keys and values can be user-implemented, but they are required to be serialized since they are communicated across the network. Keys and values can be of any data types. Users will work into two blocks: Map and Reduce. The phases of whole process including sub-phases are explained in detailed later. It is important to know that while the different phases in a MapReduce job is considered to be sequential, they are working in parallel as much as possible. Figure 2 shows the MapReduce data process model.

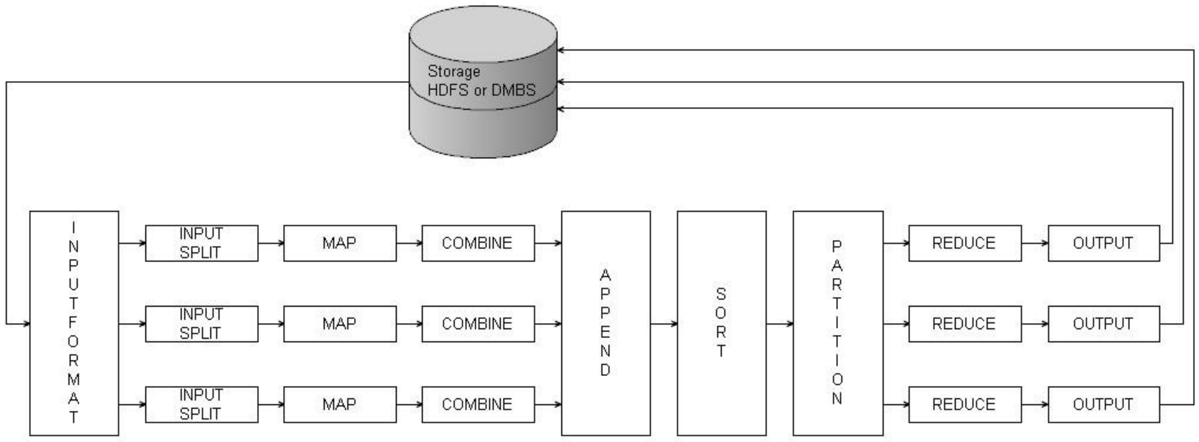

Figure 2: MapReduce phases in Hadoop MapReduce

The MapReduce framework contains the following phases:

2.2.1 InputFormat

This is the input phase which takes the input from user of some sort and splits it into InputSplits phase. The input file is normally a file on DFS, tables from a DBMS or anything the programmer wants to read.

2.2.2 InputSplits

This phase is always dependent on what the input is. It is a subset data and the splits are created in such a way that one InputSplit is sent to each Map task.

2.2.3 MAP

User defined Map function takes an input in the form of key-value pair generated through InputSplit. Each node runs one map task in parallel with each other. MapReduce library put all values with the same intermediate key together, then pass them to Reduce function. The Map function is expressed as :

$$map\ (inkey, invalue)\ \rightarrow (opkey, intermediatevalue) list$$

2.2.4 Combine

This is an optional phase that is run after each Map task on each node to mini-reduce all keys that are same generated from the current Map task.

2.2.5 Shuffle

This phase occurs when all nodes have completed their own task. In this data is communicated with each node. Key-value pairs are passed to append, sort and partition it.

2.2.6 Shuffle-append

This phase is automatically carried out by the framework. This is phase to just put all the data together.

2.2.7 Shuffle-sort

The sort phase is carried out to sort the keys either in a default way or in a programmer-implemented way.

2.2.8 Shuffle-Partition

This is the last phase of shuffle phase. This phase can be handled in two ways- default way or programmer-implemented way. This is used to calculate how the combined data should be split out to pass to the reducers. Passing equal amount of data to each reducer always provides the better performance.

2.2.9 Reduce

Reduce function accepts an intermediate key and related value. It is done by taking all the key-value pairs with the same key and performing some kind of reducing on values. The reducers always have a unique key with them i.e. if a reducer has one key, no other reducer will receive that key. The function is expressed as :

$$reduce\ (outkey, intermediatevalue\ list) \rightarrow outvalue\ list$$

2.2.10 Output

One output to storage is generated by each reducer. Normally the output files are generated in part-files for each reducer such as part-r-00000, part-r-00001 etc. this can be programmatically changed by implementing the OutputFormat.

Normally the output produce by Reduce is in the form of 0 or 1. The value list is controlled based on memory size with the help of an iterator and reduce function [15].

The working of MapReduce can be explained using a simple word count example which will illustrate all the important computing phases. In this example a large document is scanned and number of particular word occurs is counted. The whole process is carried out as follows:

1. Firstly the document is divided into many splits. The input key is considered as the ID of the split and the split is value corresponding to that key. Finally the document gets divided into many splits of which each contains (key, value) pairs.
2. Suppose that we are having m no of Mappers for this work such that each Mapper contains one input split based on any of the scheduling policy available. (If the Mapper consumes its split, it asks for the next split). Each Mapper process the (key, value) pair by using some user defined Map function to produce intermediate key value pairs. Suppose in our case we will consider that the word 'Mumbai' occurs 2 times and 'Pune' occurs 3 times. So the output of the Mapper would be as follows:
   (Mumbai, 1)
   (Mumbai, 1)
   (Pune, 1)

(Pune, 1)
(Pune, 1)

Here the above format indicates one occurrence of each key i.e. either Mumbai or Pune.

3. Then the above set of intermediate pairs is then pulled by the "Reducer" to carry out further work.
4. Then according to a user define function the Reducer work to combine all the values of above intermediate key. Thus in above example after processing the set of intermediate (key, value) pairs the output would be as follow:

   (Mumbai, 2)

   (Pune, 3)

5. This is the final output that is written to disk as the final output of the job which provides the word count of two different words 'Mumbai' and 'Pune'. We can also provide more words to carry out this example.

To reduce the network traffic between Mapper and Reducer as stated above we can always use a Combiner phase.

3. EUCALYPTUS PLATFORM

3.1 Introduction

Eucalyptus is an open-source free cloud management system that provides the functionality in terms of IaaS deployment and also can be used as private, hybrid or even public cloud system with enough hardware. It uses the same API as the AWS use which enables tools that are originally developed for AWS to be used with the added benefit of Eucalyptus being open-source and free. The Eucalyptus Machine Image (EMI) is the instances running inside Eucalyptus which can be created or downloaded as pre-packaged version. As soon as it starts it contacts its different components to determine the layout and setup of system it controls which are generally configured by using the configuration files in each of those components.

As shown in Figure 3, Eucalyptus have the components that provides same features as Amazon in terms of computation clouds with different names but with equal functionality and API [16].

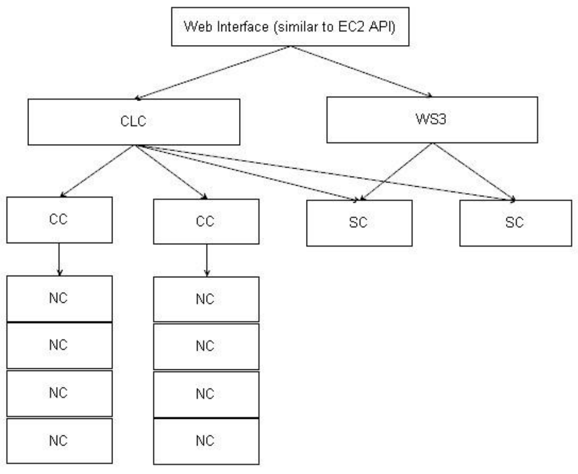

Figure 3: EUCALYPTUS COMPONENTS

3.1.1 Cloud Controller (CLC)

CLC is equivalent to Amazon's Elastic Compute Cloud and it is the starting point of the Eucalyptus cloud. The main role of CLC is to provide the computational power such as CPU and RAM to the user also it stats stops and controls the instances in the system. Based on this particular information on the infrastructure's load and resource availability, it decides the available resources so to dispatch the load to the clusters. Thus we can say that it is the frontend for managing the whole infrastructure of the cloud. It contains both web service interface for end users and web interfaces for administrators. It is also written in Java and it contacts the hypervisors indirectly through Cluster Controllers (CC) and Node Controllers (NC).

3.1.2 Cluster Controller (CC)

Cluster can be defined as a collection of machines grouped together in the same network broadcast domain. CC being the entry point to the cluster manages the NCs and several instances running on them. It communicates with CLC and NC, receives request for deploying instances from CLS and choose the NC that will be used to deploy. It also controls the virtual network between instances and collects all the information on NCs and transfers it to the CLC.

3.1.3 Storage Controller (SC)

The Storage Controller (SC) is responsible for providing fast dynamic storage devices with low latency and variable storage size as Elastic Block Storage does in Amazon. It provides persistent storage for instances on the cluster level in the form of block level storage volumes. The instances use ATA over Ethernet or Internet SCSI protocol to mount the virtual storage devices. The SC is also written in Java.

3.1.4 Walrus Storage Controller (WS3)

Similar to Amazon S3 Walrus is the name of the storage container that stores data in buckets with same API to read and write data in a redundant system: put/get storage model to create and delete buckets, create objects and also put or get those objects from buckets. The main function of WS3 is to store the machine images and snapshots. It is written in Java and is accessible through the same means as S3 i.e. SOAP, REST or Web Browser.

3.1.5 Node Controller (NC)

Node Controller always runs on each and every node present on UEC. It controls the instances on the node. It gathers the data about physical resource availability on the node and their utilization and data about instances and finally reports to the Cluster Controller. It also communicates with the OS and the Hypervisor running on the node, and Cluster Controller.

4. OUR PROPOSED WORK

As we have seen above Hadoop splits any job to the number of available Mappers and executes them using predefined Map function. The problem with this scenario that we are concentrating on is what if the numbers of nodes that are available are less than the number of Mappers. In this case, some Mappers have to wait till the other Mappers finish their own work. But again waiting for such task will decrease the performance as this will take more time and cost to execute the task. Also parallelism will also be not maintained. Another similar scenario can be that if any of the nodes i.e. Mapper or Reducer fails during execution of a particular task the whole task of that is restarted again. We also consider one more scenario where the user wants to get job finishes in less time.

In this case also the user has to increase the number of Mappers and Reducers which will ultimately increase the number of machines and thus cost.

The only solution that is feasible to this work is to carry out the whole process of Hadoop on Cloud. We have defined the open source platform EUCALYPTUS for using the cloud. This provides the solution to above scenarios. For example consider the first scenario where the numbers of nodes available are less than Mappers. In this case, we just have to deploy a single cloud and increase the number of Node Controller (NC) to the number of Mappers that are available. In the next scenario if any of the node fails at a particular state then that node will be replaced by a new node as soon as it gets fails. Thus in this case also the performance of the whole system remains good. In the last scenario also if the user wants to get its job done more quickly than just have to increase the number of NC so that the Mappers and Reducers can carry out their work in parallel. The working all the different phases of Hadoop knows how to work in parallel so their working is just maintained by Hadoop framework. The problem to increase the Node Controller in EUCALYPTUS is a feature that is available by Cloud Computing so in this case also the work will gets divide as soon as any new NC is added into the system.

Thus the above proposed solution is only possible if we use the Hadoop on the Cloud. We have started on how to integrate the Hadoop Framework with the EUCALYPTUS cloud.

5. FUTURE WORK

We have proposed the solution about the slaves of the Hadoop Framework. We are working to above proposed solution. Also as explained in the Hadoop framework that it is controlled by a single system which is known as "master". So the future work is that whether we can add the "master" node or not because it can be possible that the "master" node fails due to some conditions while executing the task. In such a case if the "master" node fails the whole task gets start from the beginning by new "master" node. Also the other implementation of Hadoop such as filesystem that it uses i.e. HDFS can be added runtime i.e. we can even add any new file system while executing the job. Also using queues to overlap the map and shuffling stage seems to be a promising approach to improve MapReduce performance.

6. CONCLUSION

It is clear that any large-scale system can be simplified if we built it using the Cloud. In this paper, using MapReduce as an example we have tried to explained that it is possible to overcome the limitations of the cloud without decreasing the performance. We have tried to propose a new fully distributed architecture of MapReduce programming model by using the general techniques that can be used for other systems also. In our model, MapReduce is implemented on the top of Cloud OS using the open-source platform EUCALYPTUS. This can be used to carry out the working of MapReduce in a better way by using the features of Cloud Computing. Even though a full scale performance evaluation is beyond the scope of this paper, our preliminary proposed solution indicates that this can be practically implemented and its performance is better with that of Hadoop.

7. REFERENCES

[1] Microsoft. Microsoft private cloud. http://www.microsoft.com/virtualization/en/us/private-cloud.aspx, 2011. Retrieved 2011-04-26.

[2] VMware. Vmware vcloud. http://www.microsoft.com/virtualization/en/us/private-cloud.aspx, 2011. Retrieved 2011-04-26.

[3] Citrix. Citrix open cloud platform. http://www.citrix.com/English/ps2/products/subfeature.asp?contentID=2303748, 2011. Retrieved 2011-04-26.

[4] OpenNebula. Opennebula - the opensource toolkit for cloud computing. http://opennebula.org/, 2011. Retrieved 2011-04-26.

[5] Cloud.com. The cloud os for the modern datacenter. http://cloud.com/, 2011. Retrieved 2011-04-26.

[6] G. Orenstein, "Digging Deeper Into Data With Hadoop," Available at http://gigaom.com/2009/06/07/digging-deeper-into-data-with-hadoop, 2009

[7] Jeffrey Dean and Sanjay Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters", Symposium on Operating Systems Design and Implementation, 2004

[8] Hadoop: The Definitive Guide, Tom Wbite, 2010

[9] Hadoop in Action, Chuck Lam, 2010

[10] NCHC Cloud Computing Research Group website, http://trac.nchc.org.tw/cloud

[11] Daniel Peng and Frank Dabek, "Large-scale Incremental Processing Using Distributed Transactions and Notifications", Operating Systems Design and Implementation, Oct. 2010

[12] T. White. Hadoop - The Defenitive Guide. O'Reilly Media, 2nd edition, 2010.

[13] Dean J, Ghemawat S. MapReduce: Simplifed Data Processing on Large Clusters[C]//Proc. of the 6th Symposium on Operating System Design and Implementation, San Francisco. 2004.

[14] J Dean and S Ghemawat. Mapreduce: Simplied data processing on large clusters. In OSDI'04: Sixth Symposium on Operating System Design and Implementation. Google Inc., 2004.

[15] ZHENG Xin-jie, ZHU Cheng-rong, XIONG Qi-bang, "Design and Implementation of Distributed Ray Tracing" . Computer Engineering. November 2007

[16] Eucaluptys Systems, Inc. Eucalyptus Administration Guide (2.0), 2010.