NATIONAL UNIVERSITY OF SINGAPORE, SINGAPORE

HANDLING LARGE SCALE SEARCHING PROBLEMS USING MAPREDUCE PARADIGM

Summer-Internship under the supervision of
Assoc/Prof Bharadwaj Veeravalli

Girish N, 81307132022

Final year, Bachelor of Engineering, Computer Science and Engineering, Oxford Engineering College, Tiruchirapalli, Tamil Nadu, India

ABSTRACT

Large scale searching problems such as searching for the occurrence of a particular word in a set of given documents, or searching for a particular tag in a set of web pages are always challenging tasks. In this work, we address this problem. We consider searching for the occurrence of a particular word in a given set of files. We use MapReduce paradigm to implement our proposed scheme in the popular Open Source Cloud computing platform Hadoop. A simple 3-node Hadoop cluster is setup to demonstrate the capability of our algorithm. Moreover, we calculate the number of occurrence of the given word and analyze the results. We also describe the prospective extensions for the proposed scheme and conclude our work.

ACNOWLEDGEMENTS

I owe my most sincere gratitude to my supervisor, A/Prof. Bharadwaj Veeravalli who provided guidance, support, help during the project. I gained a completely new perspective about approaching problems and devising solutions from him, which helped me to do this exciting project.

I am extremely thankful to Mr. Ganesh Neelakanta Iyer, my guide, teacher and supervisor for this project. Mr. Iyer helped me articulate my ideas, provided direction and helped me achieve my targets. I have become meticulous in my work and have learnt to become more organized. Mr. Iyer provided tremendous motivation in my research phase and was always receptive to my questions, doubts and ideas. I am grateful to him for all the help and guidance for this project.

I would also like to thank Prof Muthu Venkatachalam (HOD, Computer Science and Engineering), for all his support and encouragement throughout my studies. Finally, I would like to thank my parents Mr S Neelakanta Iyer and Mrs P Vasantha for all their love and support for my studies.

Contents

1. INTRODUCTION

Nowadays applications require more resources than that are available in an inexpensive machine [8]. Many organizations find themselves with business processes that no longer fit on a single cost-effective computer. A simple but expensive solution has been to buy speciality machines that have a lot of memory and many CPUs. This solution scales as far as what is supported by the fastest machines available, and usually the only limiting factor is our budget. An alternative solution is to build a high-availability cluster. Such a cluster typically attempts to look like a single machine, and typically requires very specialized installation and administration services. Many high-availability clusters are proprietary and expensive.

A more economical solution for acquiring the necessary computational resources is cloud computing. A common pattern is to have bulk data that needs to be transformed, where the processing of each data item is essentially independent of other data items; which means that using a Single-Instruction Multiple-Data (SIMD) algorithm. Hadoop Core provides an open source framework for cloud computing, as well as a distributed file system.

We are living in the data age. It's not easy to measure the total volume of data stored electronically, but an IDC estimate put the size of the "digital universe" at 0.18 zettabytes in 2006, and is forecasting a tenfold growth by 2011 to 1.8 zettabytes[4]. A zettabyte is 1021 bytes, or equivalently one thousand exabytes, one million petabytes, or one billion terabytes. That's roughly the same order of magnitude as one disk drive for every person in the world. This flood of data is coming from many sources. Consider the following examples [3]:

• The New York Stock Exchange generates about one terabyte of new trade data per day.

• Facebook hosts approximately 10 billion photos, taking up one petabyte of storage.

• Ancestry.com, the genealogy site, stores around 2.5 petabytes of data.

• The Internet Archive stores around 2 petabytes of data, and is growing at a rate of 20 terabytes per month.

• The Large Hadron Collider near Geneva, Switzerland, will produce about 15petabytes of data per year.

The trend is for every individual's data footprint to grow, but perhaps more importantly the amount of data generated by machines will be even greater than that generated by people. Machine logs, RFID readers, sensor networks, vehicle GPS traces, retail transactions—all of these contribute to the growing mountain of data.

The volume of data being made publicly available increases every year too. Organizations no longer have to merely manage their own data: success in the future will be dictated to a large extent by their ability to extract value from other organizations' data. Initiatives such as Public Data Sets on Amazon Web Services, Infochimps.org, and theinfo.org exist to foster the "information commons," where data can be freely (or in the case of AWS, for a modest price) shared for anyone to download and analyze. Mashups between different information sources make for unexpected and hitherto unimaginable applications.

i. MapReduce: The Secret Weapon of Google

MapReduce does what our brains do all the time: It categorizes (Maps) key pieces of information, distributes it across its server farm of PCs, and then eliminates (Reduces) irrelevant data (computers--unlike MapReduce and the brain--soak in everything). Google now uses MapReduce for over 10,000 programs[10], ranging from the processing of satellite imagery, language processing and responding to popular queries. It is now processing roughly 100,000 functions daily and digesting 20 petabytes of data each day.

ii. Problem statement and Motivation

Large scale searching has always been a challenging task. It can be searching for a particular pattern in a given set of files, or searching for a tag word from a set of websites. It is challenging when the input data size is of the order of hundreds of petabytes. In this work, we consider searching a word from the given set of input files using MapReduce paradigm. In addition, we consider counting the number of occurrences of the given word and display the results. We implement our scheme in Hadoop [11] – an open source Cloud platform – by setting up a 3-node multi-cluster Hadoop and analyze our results.

iii. Report Organization

This report is organized as follows:

Chapter 2 gives an overview of Hadoop platform and MapReduce paradigm. We also compare MapReduce with other existing distributed processing strategies such as RDMS and Grid computing.

Chapter 3 describes the process of setting up a single cluster Hadoop system in a step by step manner including installing the Operating System.

Chapter 4 describes the process of setting up a multi-cluster Hadoop system. It tells how a set of Single cluster Hadoop nodes can be changed to make a multi-cluster Hadoop system.

Chapter 5 outlines the details of the implemented large scale searching problem in Hadoop using MapReduce paradigm. It describes the code and analyzes the results.

Chapter 6 outlines the possible future extensions and concludes this work.

2. OVERVIEW OF HADOOP AND MAPREDUCE

i. Overview of Hadoop

Apache Hadoop is a Java software framework that supports data-intensive distributed applications under a free license. It enables applications to work with thousands of nodes and petabytes of data. Hadoop was inspired by Google's MapReduce and GFS (Google file system. Hadoop is a top-level Apache project, being built and used by a community of contributors from all over the world. Yahoo! has been the largest contributor to the project and uses Hadoop extensively across its businesses.

Hadoop was created by Doug Cutting , who named it after his son's stuffed elephant. It was originally developed to support distribution for the Nutch search engine project. The HDFS file system [9] stores large files (an ideal file size is a multiple of 64 MB), across multiple machines. It achieves reliability by replicating the data across multiple hosts, and hence does not require Redundant Array of Inexpensive Disks (RAID) storage on hosts. With the default replication value, 3, data is stored on three nodes: two on the same rack, and one on a different rack.

The file system is built from a cluster of *data nodes*, each of which serves up blocks of data over the network using a block protocol specific to Hadoop Distributed File System (HDFS). They also serve the data over HTTP, allowing access to all content from a web browser or other client. Data nodes can talk to each other to rebalance data, to move copies around, and to keep the replication of data high.

A filesystem requires one unique server, the *name node*. This is a single point of failure for an HDFS installation. If the name node goes down, the filesystem is offline. When it comes back up, the name node must replay all outstanding operations. This replay process can take over half an hour for a big cluster. The filesystem includes what is called a *Secondary namenode*, which misleads some people into thinking that when the primary namenode goes offline, the Secondary namenode takes over. In fact, the Secondary namenode regularly connects with the namenode and downloads a snapshot of the primary namenode's directory information, which is then saved to a directory. This Secondary namenode is used together with the edit log of the primary namenode to create an up-to-date directory structure.

Another limitation of HDFS is that it cannot be directly mounted by an existing operating system. Getting data into and out of the HDFS file system, an action that often needs to be performed before and after executing a job can be inconvenient. A filesystem in user space has been developed to address this problem, at least for Linux and some other Unix systems.

Replicating data three times is costly. To alleviate this cost, recent versions of HDFS have erasure coding support whereby multiple blocks of the same file are combined together to generate a parity block. HDFS creates parity blocks asynchronously and then decreases the replication factor of the file from 3 to 2. Studies have shown that this technique decreases the physical storage requirements from a factor of 3 to a factor of around 2.2.

ii. Overview of MapReduce

MapReduce is a programming model for data processing. The model is simple, yet not too simple to express useful programs in. Hadoop can run MapReduce programs written in various languages; in this chapter, we shall look at the same program expressed in Java, Ruby, Python, and C++. Most importantly, MapReduce programs are inherently parallel, thus putting very large-scale data analysis into the hands of anyone with enough machines at their disposal. MapReduce comes into its own for large datasets, so let’s start by looking at one.

The MapReduce paradigm [5] takes inspiration from the **map** and **reduces** programming constructs common in many programming languages. We introduce map and reduce using Python, then draw out differences between map/reduce in most programming languages and the map/reduce *phases* of Hadoop.

MAP:-Map *applies* a function to each element in a list, returning a list of the results. For example, here is code in Python which uses map to triple each element in a list.

```
def triple(n):
    return n * 3
print map(triple, [1, 2, 3, 4])
```

This code prints the following list:

```
[3, 6, 9, 12]
```

REDUCE:- Reduce also applies a function to elements in a list, but instead of being applied to each element separately, the function is applied to two arguments, the "current result" and the next element in the list. The current result is initialized by calling reduce on the first two elements in the list. This allows you to build a single result (which can be another list but is often a scalar value) from a list. This is best illustrated in another simple Python example:

```
def sum(n1, n2):
    return n1 + n2
print reduce(sum, [1, 2, 3, 4])
```

We can observe that, this function is making three recursive function calls like this:

```
sum (sum (sum (1, 2), 3), 4)
```

iii. Parallelizing Map and Reduce with Hadoop

In the MapReduce programming abstraction, the **map** function takes a single <key, value> pair, and produces **zero or more** <key, value> pairs as a result. This differs slightly from the functional programming construct **map** which always produces one and only one result for each invocation of**map**. The MapReduce style of **map** allows you to produce any *intermediate* pairs which can then be further analyzed with **reduce**.

In MapReduce, the reducer is also more flexible than its functional programming counterpart. While **reduce** is similar in spirit to the reduce described in the Python example above, it is not limited to processing the list of pairs two-at-a-time, but rather is given an iterator over all pairs that have the same key, and that list can be walked over in any way the programmer chooses. Also like the MapReduce **map**, the MapReduce **reduce** can emit an arbitrary number of pairs, although applications often will want to just reduce to a single output pair.

iv. Data Storage and Analysis

The problem is simple: while the storage capacities of hard drives have increased massively over the years, access speeds—the rate at which data can be read from drives—have not kept up. One typical drive from 1990 could store 1370 MB of data and had a transfer speed of 4.4 MB/s [2]. So we could read all the data from a full drive in around five minutes. Almost 20 years later one terabyte drives are the norm, but the transfer speed is around 100 MB/s, so it takes more than two and a half hours to read all the data off the disk.

This is a long time to read all data on a single drive—and writing is even slower. The obvious way to reduce the time is to read from multiple disks at once. Imagine if we had 100 drives, each holding one hundredth of the data. Working ***in parallel***, we could read the data in less than two minutes.

Only using one hundredth of a disk may seem wasteful. But we can store one hundred datasets, each of which is one terabyte, and provide shared access to them. We can imagine that the users of such a system would be happy to share access in return for shorter analysis times, and, statistically, that their analysis jobs would be likely to be spread over time, so they wouldn’t interfere with each other too much. There’s more to being able to read and write data in parallel to or from multiple disks, though.

The first problem to solve is hardware failure: as soon as you start using many pieces of hardware, the chance that one will fail is fairly high. A common way of avoiding data loss is through replication, redundant copies of the data are kept by the system so that in the event of failure, there is another copy available. This is how RAID (Redundant Array of Independent Disks) works, for instance, although Hadoop’s filesystem, the Hadoop Distributed Filesystem (HDFS), takes a slightly different approach.

The second problem is that most analysis tasks need to be able to combine the data in some way; data read from one disk may need to be combined with the data from any of the other 99 disks. Various distributed systems allow data to be combined from multiple sources, but doing this correctly is notoriously challenging. MapReduce provides a programming model that abstracts the problem from disk reads and writes, transforming it into a computation over sets of keys and values. A quick overview of MapReduce is described later in this chapter.

The important point is that there are two parts to the computation, the map, reduce, and it’s interface between the two where the “mixing” occurs. Like HDFS, MapReduce has reliability built-in. Hadoop provides: a reliable shared storage and analysis system. The storage is provided by HDFS, and analysis by MapReduce. There are other parts to Hadoop, but these capabilities are its kernel.

v. Comparison with Other Systems

The approach taken by MapReduce may seem like a brute-force approach. The premise is that the entire dataset—or at least a good portion of it—is processed for each query. But this is its power. MapReduce is a *batch* query processor, and the ability to run an ad hoc query against your whole dataset and get the results in a reasonable time is transformative. It changes the way you think about data, and unlocks data that was previously archived on tape or disk. It gives people the opportunity to innovate with data. Questions that took too long to get answered before can now be answered, which in turn leads to new questions and new insights.

For example, Mailtrust, Rackspace’s mail division, used Hadoop for processing email logs. One ad hoc query they wrote was to find the geographic distribution of their users. In their words: This data was so useful that we’ve scheduled the MapReduce job to run monthly and we will be using this data to help us decide which Rackspace datacenters to place new mail servers in as we grow. By bringing several hundred gigabytes of data together and having the tools to analyze it, the Rackspace engineers were able to gain an understanding of the data that they otherwise would never have had, and, furthermore, they were able to use what they had learned to improve the service for their customers.

RDBMS

The need of MapReduce than using database with lots of disks to do large-scale batch analysis is that, 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 are shown in Table 1-1.) 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.

Table 1-1. RDBMS compared to MapReduce

Term	**Traditional RDBMS**	**MapReduce**
Data size	*Gigabytes*	*Petabytes*
Access	*Interactive and batch*	*Batch*
Updates	*Read and write many times*	*Write once, read many times*
Structure	*Static schema*	*Dynamic schema*
Integrity	*High*	*Low*
Scaling	*Nonlinear*	*Linear*

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.

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 importantly, 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[1].

Grid Computing

The High Performance Computing (HPC) and Grid Computing communities have been doing large-scale data processing for years, using such APIs as Message Passing Interface (MPI). Broadly, the approach in HPC is to distribute the work across a cluster of machines, which access a shared filesystem, hosted by a SAN. This works well for predominantly compute-intensive jobs, but becomes a problem when nodes need to access larger data volumes (hundreds of gigabytes, the point at which MapReduce really starts to shine), since the network bandwidth is the bottleneck, and compute nodes become idle. MapReduce tries to collocate the data with the compute node, so data access is fast since it is local. This feature, known as *data locality*, is at the heart of MapReduce and is the reason for its good performance. Recognizing that network bandwidth is the most precious resource in a datacenter environment (it is easy to saturate network links by copying data around), MapReduce implementations go to great lengths to preserve it by explicitly modelling network topology. Notice that this arrangement does not preclude high-CPU analyses in MapReduce.

MapReduce might sound like quite a restrictive programming model, and in a sense it is: you are limited to key and value types that are related in specified ways and mappers and reducers run with very limited coordination between one another (the mappers pass keys and values to reducers).

3. SETTING UP A SINGLE-CLUSTER HADOOP

i. Introduction

This chapter describes the installation and setting up of a single cluster hadoop system which runs on CentOS Linux operating system. We describe this based on our installation experience. For important terms and definitions in hadoop refer Appendix C.

ii. Installing Cent OS

CentOS is one of the good platforms for setting up a Hadoop cluster (Single node and Multinode).To start, download the OS setup media [12] ,and boot each machine with it. Then follow the instructions to install the OS. Make sure that Firewall is disabled during installation. Note the following points while installing the OS:

- De-select all components, including all GUIs and even "Server"

- On the component selection page, click "Customize now" radio button and click next

- In "Base System" group, De-select "Dialup Networking Support"

It is important (at least in this tutorial) to define the host names in your DNS server (or in the host file, if no DNS server is used). Be careful with the hosts file, one problem was caused by the loop back address (127.0.0.1) being assigned to the name of your host (in this case, centos1). Apparently Java listeners use the hostname to lookup the IP address to listen to, and they picked up the loop back address from hosts file, instead of the network address.

iii. IP Configuration

We have to configure the IP for the system .To find the IP address of the system use the following command.

/sbin/ifconfig

To change the IP and hostname of your node, enter in to root as super user(su) and then using nano editor open /etc/hosts

su - root

nano /etc/hosts

Then the file will look something like this:

# Do not remove the following line, or various programs

# that require network functionality will fail.

#127.0.0.1 localhost.localdomain localhost //We have to command this line.

137.132.153.223 localhost.localdomain master //This line is the modified line containing IP and name for the node. [Master is the alias name of the localhost]

Save and exit the file. As the IP is configured, we have to *reboot* the system.

iv. Disalble IPV6 during network setup

When the OS installation is complete, make sure to turn off and disable firewalls:

service iptables save

service iptables stop

chkconfig iptables off

An alternative to DISABLE IP is as follows. This method is simple and easier to understand. (This is the method I followed).

Edit **/etc/sysconfig/network** and set "**NETWORKING_IPV6**" to "**no**"

Add the following to /etc/modprobe.conf :

alias ipv6 off

alias net-pf-10 off

Run /sbin/chkconfig ip6tables off to disable the IPv6 firewall

Reboot the system

Alternative (which might be easier and works on any release with /etc/modprobe.d):

# touch /etc/modprobe.d/disable-ipv6

# echo "install ipv6 /bin/true" >> /etc/modprobe.d/disable-ipv6

With the 5.4 update symbol/ipv6 module dependency capabilities have been introduced. Therefore, if IPv6 has been previously disabled as above an upgrade to the bonding driver in 5.4 will result in the bonding kernel module failing to load. For the module to load properly use instead:

# touch /etc/modprobe.d/disable-ipv6

# echo "options ipv6 disable=1" >> /etc/modprobe.d/disable-ipv6

v. Installing Java Jdk

We can install JAVA in many ways. Assuming that you are using an SSH client like putty which does a good job with clipboard.

Using a browser (on the machine you run putty on)
visit: http://java.sun.com/javase/downloads/widget/jdk6.jsp

Select Linux as platform and click continue

On the popup click "skip this Step"

Copy the hyperlink address for "jdk-6u18-linux-i586-rpm.bin" to clipboard

Do the following in putty session:

cd /tmp

wget -O jdk-6u18-linux-i586-rpm.bin <the URL you copied from java download site>

chmod a+x jdk-6u18-linux-i586-rpm.bin

Or we can download java directly from the following site and save it default in the location /home/user/Desktop. Move the file to the location /tmp.

http://java.sun.com/javase/downloads/widget/jdk6.jsp

mv /home/user/Desktop /tmp

./jdk-6u18-linux-i586-rpm.bin

To make sure java was installed correctly:

Java –version

If java is installed properly, you should be able to see a folder like /usr/java/jdk-0.18.2

vi. User and SSH Configuration

SET UP HADOOP USER:

-Add a new user hadoop and give its password from

System →Administrator →User and Groups →Then add new user.

-Create a folder for all hadoop operations.

mkdir /hadoop

chown -R hadoop /hadoop

-Switch user to hadoop

Su – hadoop

-Set up pass phrase –less ssh (Only on Master, still logged in as hadoop)

```
ssh-keygen -t dsa
```

(leave pass phrase empty)

-Copy the identity to each machine (Only on master node, after creating hadoop user on each machine)

```
ssh-copy-id  -i  /home/hadoop/.ssh/id_dsa hadoop@master
```

-To test passwordless ssh

```
ssh master
```

vii. Download Hadoop

```
cd /hadoop
wget http://mirror.cloudera.com/apache/hadoop/core/hadoop-0.20.2/hadoop-0.20.2.tar.gz
tar -xvzf hadoop-0.20.2.tar.gz
mv hadoop-0.20.2 hadoop
```

viii. Configure Hadoop

Go to hadoop folder in hadoop directory

```
cd  /hadoop/hadoop
```

Modify the core-site.xml file using the following command

```
nano   conf /core-site.xml
```

Add the following inside the <configuration > tags

```
<property>
  <name>fs.default.name</name>
  <value>hdfs://master:9000/</value>
</property>
```

Modify the hdfs-site.xml: file using the following command

nano conf/hdfs-site.xml

Add the following inside the <configuration > tags

```
<property>
  <name>dfs.name.dir</name>
  <value>/hadoop/hdfs/name</value>
</property>
<property>
  <name>dfs.data.dir</name>
  <value>/hadoop/hdfs/data</value>
</property>
<property>
  <name>dfs.replication</name>
  <value>1</value>    // 1 represents number of slaves---in single cluster we should put only
1.in multi cluster we put n .where n is the number of slaves
</property>
```

Modify the mapred-site.xml: file using the following command

nano conf/ mapred-site.xml

Add the following inside the <configuration > tags

```
<property>
  <name>mapred.job.tracker</name>
  <value>master:9001</value>
</property>
```

Modify the hadoop-env.sh:

nano conf/hadoop-env.sh

Find the following lines and modify them accordingly:

```
export JAVA_HOME=/usr/java/jdk1.6.0_20

export HADOOP_OPTS=-Djava.net.preferIPv4Stack=true
```

Define the masters and slaves (Only on master node):

nano conf/masters

In this case hostname of master is master

master

[By default it will be localhost. We have to change it to master]

Now modify the slaves file:

nano conf/slaves

Slave in our example is master itself.

master

ix. Starting and Stopping Hadoop

Before using the hadoop HDFS, we have to format the namenode as follows

su – hadoop //Entering into the hadoop user as super user

cd /hadoop/hadoop //Go to hadoop folder in hadoop directory

bin/hadoop namenode –format // Format the namenode

If there is no error, you can start all components:

bin/start-all.sh

To stop hadoop,

bin/stop-all.sh

x. Testing Hadoop

For testing the working of hadoop, we download some text files, copy them to HDFS and use a sample MapReduce job to count words in those files

```
cd  /hadoop   // change directory to hadoop

mkdir  gutenberg  // make  a directory named  gutenberg

cd  gutenberg    // enter in  gutenberg

wget http://www.gutenberg.org/files/20417/20417.txt

wget http://www.gutenberg.org/dirs/etext04/7ldvc10.txt

wget http://www.gutenberg.org/files/4300/4300.txt

   // wget is used to download a file from the given website

cd /hadoop/hadoop

bin/hadoop dfs -ls

bin/hadoop dfs -copyFromLocal /hadoop/gutenberg gutenberg

bin/hadoop dfs -ls gutenberg

bin/hadoop dfs -rmr gutenberg-output

bin/hadoop jar hadoop-0.20.2-examples.jar wordcount gutenberg gutenberg-output
```

4. SETTING UP MULTI -CLUSTER HADOOP

i. Introduction

Assume all nodes are setup as described in previous chapter. In this chapter, we will discuss about setting up a Multi –Cluster Hadoop. It can be noted that only few changes are required as described below. The errors that appeared while we configuring hadoop and solutions are described in Appendix A

ii. IP Configuration

In the master and slave node, first log in as super user. Then go to the hosts file as follows

nano/etc/hosts

Add the IP address and hosts name to it.

```
137.132.153.239  master
137.132.153.177  slave 1
137.132.153.189  slave 2
```

Save and Exit and REBOOT the system.

iii. Edit the Configuration Files

Go to hadoop folder in hadoop directory

cd /hadoop/hadoop

Modify the core-site.xml file using the following command **(ONLY IN SLAVES)**

nano   conf /core-site.xml

Add the following inside the <configuration > tags

```
<property>
  <name>fs.default.name</name>
  <value>hdfs://master:9000/</value>
</property>
```

Modify the hdfs-site.xml: file using the following command **(ON MASTER & SLAVES)**

nano conf/hdfs-site.xml

Add the following inside the <configuration > tags

```
<property>
  <name>dfs.name.dir</name>
  <value>/hadoop/hdfs/name</value>
</property>
<property>
  <name>dfs.data.dir</name>
  <value>/hadoop/hdfs/data</value>
</property>
<property>
  <name>dfs.replication</name>
  <value>3</value>    // 3 represents number of slaves--multi cluster we are having 3 slaves
including master so 3.
</property>
```

Modify the mapred-site.xml: file using the following command **(ONLY IN SLAVES)**

nano conf/ mapred-site.xml

Add the following inside the <configuration > tags

```
<property>
  <name>mapred.job.tracker</name>
  <value>master:9001</value>
</property>
```

Define the masters and slaves (In slave node):

nano conf/masters(ON SLAVES)

In this case hostname of master is master

master

Then,

nano conf/slaves (ONLY ON MASTER)

```
master
slave1
slave2
```

Log in as super user as follows:

```
su – hadoop
```

Then do the following, (IN MASTER ONLY)

```
ssh-copy-id  -i  /hadoop/hadoop/.ssh/id_dsa hadoop@slave1

ssh-copy-id  -i  /hadoop/hadoop/.ssh/id_dsa hadoop@slave2
```

To check if the ssh connection is established without password do the following in master.

```
ssh slave1

ssh slave2
```

iv. Formatiing the Name Node

Do the following:

```
[hadoop@smaster hadoop]$bin/hadoop  namenode  -format
```

The output will be as follows

```
... INFO dfs.Storage: Storage directory /usr/local/hadoop-datastore/hadoop-hadoop/dfs/name has been successfully formatted.
hadoop@master:/usr/local/hadoop$
```

v. Do the Following for Starting DFS

```
#bin/start-dfs.sh
```

The output displayed will be as follows:

```
starting namenode, logging to /hadoop/hadoop/bin/../logs/hadoop-hadoop-namenode-master.out
slave1: starting datanode, logging to /hadoop/hadoop/bin/../logs/hadoop-hadoop-datanode-slave1.out
master: starting datanode, logging to /hadoop/hadoop/bin/../logs/hadoop-hadoop-datanode-master.out
slave2: starting datanode, logging to /hadoop/hadoop/bin/../logs/hadoop-hadoop-datanode-slave2.out
master: starting secondarynamenode, logging to /hadoop/hadoop/bin/../logs/hadoop-hadoop-secondarynamenode-master.out
```

To Check Namenode in Master

To check Name node(For Master)

```
2010-06-15 14:52:14,182 INFO org.apache.hadoop.hdfs.StateChange: BLOCK* NameSystem.addToInvalidates:
blk_3760915988618215203 is added to invalidSet of 137.132.153.177:50010
2010-06-15 14:52:14,183 INFO org.apache.hadoop.hdfs.StateChange: BLOCK* NameSystem.addToInvalidates:
blk_3760915988618215203 is added to invalidSet of 137.132.153.88:50010
2010-06-15 14:52:14,183 INFO org.apache.hadoop.hdfs.StateChange: BLOCK* NameSystem.addToInvalidates:
blk_3760915988618215203 is added to invalidSet of 137.132.153.239:50010
2010-06-15 14:52:14,194 INFO org.apache.hadoop.hdfs.server.namenode.FSNamesystem.audit: ugi=hadoop,hadoop
ip=/137.132.153.239     cmd=delete     src=/tmp/hadoop-hadoop/mapred/system     dst=null        perm=null
2010-06-15 14:52:14,204 INFO org.apache.hadoop.hdfs.server.namenode.FSNamesystem.audit: ugi=hadoop,hadoop
ip=/137.132.153.239     cmd=mkdirs     src=/tmp/hadoop-hadoop/mapred/system     dst=null        perm=hadoop:supergroup:rwxr-xr-
x
2010-06-15 14:52:14,208 INFO org.apache.hadoop.hdfs.server.namenode.FSNamesystem.audit: ugi=hadoop,hadoop
ip=/137.132.153.239     cmd=setPermission     src=/tmp/hadoop-hadoop/mapred/system     dst=null
perm=hadoop:supergroup:rwx-wx-wx
2010-06-15 14:52:14,406 INFO org.apache.hadoop.hdfs.server.namenode.FSNamesystem.audit: ugi=hadoop,hadoop
ip=/137.132.153.239     cmd=create     src=/tmp/hadoop-hadoop/mapred/system/jobtracker.info     dst=null
perm=hadoop:supergroup:rw-r--r--
2010-06-15 14:52:14,421 INFO org.apache.hadoop.hdfs.server.namenode.FSNamesystem.audit: ugi=hadoop,hadoop
ip=/137.132.153.239     cmd=setPermission     src=/tmp/hadoop-hadoop/mapred/system/jobtracker.info     dst=null
perm=hadoop:supergroup:rw-------
2010-06-15 14:52:14,430 INFO org.apache.hadoop.hdfs.StateChange: BLOCK* NameSystem.allocateBlock: /tmp/hadoop-
hadoop/mapred/system/jobtracker.info. blk_8460524915705826460_1014
2010-06-15 14:52:14,925 INFO org.apache.hadoop.hdfs.StateChange: BLOCK* NameSystem.addStoredBlock: blockMap updated:
137.132.153.177:50010 is added to blk_8460524915705826460_1014 size 4
2010-06-15 14:52:14,944 INFO org.apache.hadoop.hdfs.StateChange: BLOCK* NameSystem.addStoredBlock: blockMap updated:
137.132.153.88:50010 is added to blk_8460524915705826460_1014 size 4
2010-06-15 14:52:14,954 INFO org.apache.hadoop.hdfs.StateChange: BLOCK* NameSystem.addStoredBlock: blockMap updated:
137.132.153.239:50010 is added to blk_8460524915705826460_1014 size 4
2010-06-15 14:52:14,964 INFO org.apache.hadoop.hdfs.StateChange: DIR* NameSystem.completeFile: file /tmp/hadoop-
hadoop/mapred/system/jobtracker.info is closed by DFSClient_616927149
2010-06-15 14:52:15,622 INFO org.apache.hadoop.hdfs.StateChange: BLOCK* ask 137.132.153.177:50010 to delete
blk_3760915988618215203_1001
2010-06-15 14:52:18,625 INFO org.apache.hadoop.hdfs.StateChange: BLOCK* ask 137.132.153.239:50010 to delete
blk_3760915988618215203_1001
2010-06-15 14:52:21,629 INFO org.apache.hadoop.hdfs.StateChange: BLOCK* ask 137.132.153.88:50010 to delete
blk_3760915988618215203_1001
```

On slave*, you can examine the success or failure of this command by inspecting the log file <HADOOP_INSTALL>/logs/hadoop-hadoop-datanode-slave.log.*

Exemplary output:

```
2010-06-15 14:45:08,861 INFO org.apache.hadoop.hdfs.server.datanode.DataNode: Registered FSDatasetStatusMBean
2010-06-15 14:45:08,864 INFO org.apache.hadoop.hdfs.server.datanode.DataNode: Opened info server at 50010
2010-06-15 14:45:08,868 INFO org.apache.hadoop.hdfs.server.datanode.DataNode: Balancing bandwith is 1048576 bytes/s
2010-06-15 14:45:08,957 INFO org.mortbay.log: Logging to org.slf4j.impl.Log4jLoggerAdapter(org.mortbay.log) via
org.mortbay.log.Slf4jLog
2010-06-15 14:45:09,052 INFO org.apache.hadoop.http.HttpServer: Port returned by webServer.getConnectors()[0].getLocalPort()
before open() is -1. Opening the listener on 50075
2010-06-15 14:45:09,052 INFO org.apache.hadoop.http.HttpServer: listener.getLocalPort() returned 50075
webServer.getConnectors()[0].getLocalPort() returned 50075
2010-06-15 14:45:09,052 INFO org.apache.hadoop.http.HttpServer: Jetty bound to port 50075
2010-06-15 14:45:09,053 INFO org.mortbay.log: jetty-6.1.14
2010-06-15 14:45:09,397 INFO org.mortbay.log: Started SelectChannelConnector@0.0.0.0:50075
2010-06-15 14:45:09,414 INFO org.apache.hadoop.metrics.jvm.JvmMetrics: Initializing JVM Metrics with
processName=DataNode, sessionId=null
2010-06-15 14:45:09,438 INFO org.apache.hadoop.ipc.metrics.RpcMetrics: Initializing RPC Metrics with hostName=DataNode,
port=50020
2010-06-15 14:45:09,441 INFO org.apache.hadoop.ipc.Server: IPC Server Responder: starting
2010-06-15 14:45:09,442 INFO org.apache.hadoop.ipc.Server: IPC Server listener on 50020: starting
2010-06-15 14:45:09,443 INFO org.apache.hadoop.ipc.Server: IPC Server handler 0 on 50020: starting
2010-06-15 14:45:09,444 INFO org.apache.hadoop.ipc.Server: IPC Server handler 1 on 50020: starting
2010-06-15 14:45:09,444 INFO org.apache.hadoop.hdfs.server.datanode.DataNode: dnRegistration =
DatanodeRegistration(slave1:50010, storageID=DS-1089239121-137.132.153.177-50010-1276489276151, infoPort=50075,
ipcPort=50020)
2010-06-15 14:45:09,445 INFO org.apache.hadoop.ipc.Server: IPC Server handler 2 on 50020: starting
2010-06-15 14:45:09,461 INFO org.apache.hadoop.hdfs.server.datanode.DataNode: DatanodeRegistration(137.132.153.177:50010,
storageID=DS-1089239121-137.132.153.177-50010-1276489276151, infoPort=50075, ipcPort=50020)In DataNode.run, data =
FSDataset{dirpath='/hadoop/hdfs/data/current'}
2010-06-15 14:45:09,461 INFO org.apache.hadoop.hdfs.server.datanode.DataNode: using BLOCKREPORT_INTERVAL of
```

```
3600000msec Initial delay: 0msec
2010-06-15 14:45:09,505 INFO org.apache.hadoop.hdfs.server.datanode.DataNode: BlockReport of 10 blocks got processed in 19
msecs
2010-06-15 14:45:09,507 INFO org.apache.hadoop.hdfs.server.datanode.DataNode: Starting Periodic block scanner.
```

At this point, the following Java processes should run on **master...**

```
#/usr/java./jdk/hadoop$ jps
15183 DataNode
15616 Jps
14977 SecondaryNameNode
```

Run the command **<HADOOP_INSTALL>/bin/start-mapred.sh** on the machine you want the jobtracker to run on. This will bring up the MapReduce cluster with the jobtracker running on the machine you ran the previous command on, and tasktrackers on the machines listed in the conf/slaves file.
#bin/start-mapred.sh **(ON MASTER)**
The output obtained will be as follows

```
starting namenode, logging to /hadoop/hadoop/bin/../logs/hadoop-hadoop-namenode-master.out
slave1: starting datanode, logging to /hadoop/hadoop/bin/../logs/hadoop-hadoop-datanode-slave1.out
master: starting datanode, logging to /hadoop/hadoop/bin/../logs/hadoop-hadoop-datanode-master.out
slave2: starting datanode, logging to /hadoop/hadoop/bin/../logs/hadoop-hadoop-datanode-slave2.out
master: starting secondarynamenode, logging to /hadoop/hadoop/bin/../logs/hadoop-hadoop-secondarynamenode-master.out
[hadoop@master hadoop]$ bin/start-mapred.sh
starting jobtracker, logging to /hadoop/hadoop/bin/../logs/hadoop-hadoop-jobtracker-master.out
slave2: starting tasktracker, logging to /hadoop/hadoop/bin/../logs/hadoop-hadoop-tasktracker-slave2.out
master: starting tasktracker, logging to /hadoop/hadoop/bin/../logs/hadoop-hadoop-tasktracker-master.out
slave1: starting tasktracker, logging to /hadoop/hadoop/bin/../logs/hadoop-hadoop-tasktracker-slave1.out
```

On **slave,** you can examine the success or failure of this command by inspecting the log file

<HADOOP_INSTALL>/**logs/hadoop-hadoop-tasktracker-slave.log.**

Exemplary output:

```
************************************************************/
2010-06-15 14:50:41,892 INFO org.apache.hadoop.mapred.TaskTracker: STARTUP_MSG:
/************************************************************
STARTUP_MSG: Starting TaskTracker
STARTUP_MSG:   host = slave1/137.132.153.177
STARTUP_MSG:   args = []
STARTUP_MSG:   version = 0.20.2
STARTUP_MSG:   build = https://svn.apache.org/repos/asf/hadoop/common/branches/branch-0.20 -r 911707; compiled by 'chrisdo'
on Fri Feb 19 08:07:34 UTC 2010
************************************************************/
2010-06-15 14:50:42,148 INFO org.mortbay.log: Logging to org.slf4j.impl.Log4jLoggerAdapter(org.mortbay.log) via
org.mortbay.log.Slf4jLog
2010-06-15 14:50:42,391 INFO org.apache.hadoop.http.HttpServer: Port returned by webServer.getConnectors()[0].getLocalPort()
before open() is -1. Opening the listener on 50060
2010-06-15 14:50:42,404 INFO org.apache.hadoop.http.HttpServer: listener.getLocalPort() returned 50060
webServer.getConnectors()[0].getLocalPort() returned 50060
2010-06-15 14:50:42,404 INFO org.apache.hadoop.http.HttpServer: Jetty bound to port 50060
2010-06-15 14:50:42,404 INFO org.mortbay.log: jetty-6.1.14
2010-06-15 14:50:42,799 INFO org.mortbay.log: Started SelectChannelConnector@0.0.0.0:50060
2010-06-15 14:50:42,805 INFO org.apache.hadoop.metrics.jvm.JvmMetrics: Initializing JVM Metrics with
processName=TaskTracker, sessionId=
2010-06-15 14:50:42,823 INFO org.apache.hadoop.ipc.metrics.RpcMetrics: Initializing RPC Metrics with hostName=TaskTracker,
port=39244
2010-06-15 14:50:42,892 INFO org.apache.hadoop.ipc.Server: IPC Server Responder: starting
```

```
2010-06-15 14:50:42,894 INFO org.apache.hadoop.ipc.Server: IPC Server listener on 39244: starting
2010-06-15 14:50:42,894 INFO org.apache.hadoop.ipc.Server: IPC Server handler 1 on 39244: starting
2010-06-15 14:50:42,894 INFO org.apache.hadoop.ipc.Server: IPC Server handler 0 on 39244: starting
2010-06-15 14:50:42,895 INFO org.apache.hadoop.ipc.Server: IPC Server handler 2 on 39244: starting
2010-06-15 14:50:42,896 INFO org.apache.hadoop.ipc.Server: IPC Server handler 3 on 39244: starting
2010-06-15 14:50:42,897 INFO org.apache.hadoop.mapred.TaskTracker: TaskTracker up at: 127.0.0.1/127.0.0.1:39244
2010-06-15 14:50:42,897 INFO org.apache.hadoop.mapred.TaskTracker: Starting tracker tracker_slave1:127.0.0.1/127.0.0.1:39244
2010-06-15 14:50:42,980 INFO org.apache.hadoop.mapred.TaskTracker:  Using MemoryCalculatorPlugin :
org.apache.hadoop.util.LinuxMemoryCalculatorPlugin@297ffb
2010-06-15 14:50:42,984 WARN org.apache.hadoop.mapred.TaskTracker: TaskTracker's totalMemoryAllottedForTasks is -1.
TaskMemoryManager is disabled.
2010-06-15 14:50:42,986 INFO org.apache.hadoop.mapred.IndexCache: IndexCache created with max memory = 10485760
2010-06-15 14:50:42,988 INFO org.apache.hadoop.mapred.TaskTracker: Starting thread: Map-events fetcher for all reduce tasks
on tracker_slave1:127.0.0.1/127.0.0.1:39244
```

At this point, the following Java processes should run on master...

```
hadoop@master:/usr/local/hadoop$ jps
16017 Jps
14799 NameNode
15686 TaskTracker
14880 DataNode
15596 JobTracker
14977 SecondaryNameNode
**

hadoop@master:/usr/local/hadoop$
```

(The process IDs don't matter of course)

And the following on slave:

```
hadoop@slave:/usr/local/hadoop$ jps
15183 DataNode
15897 TaskTracker
16284 Jps
16689 Secondary Namenode
```

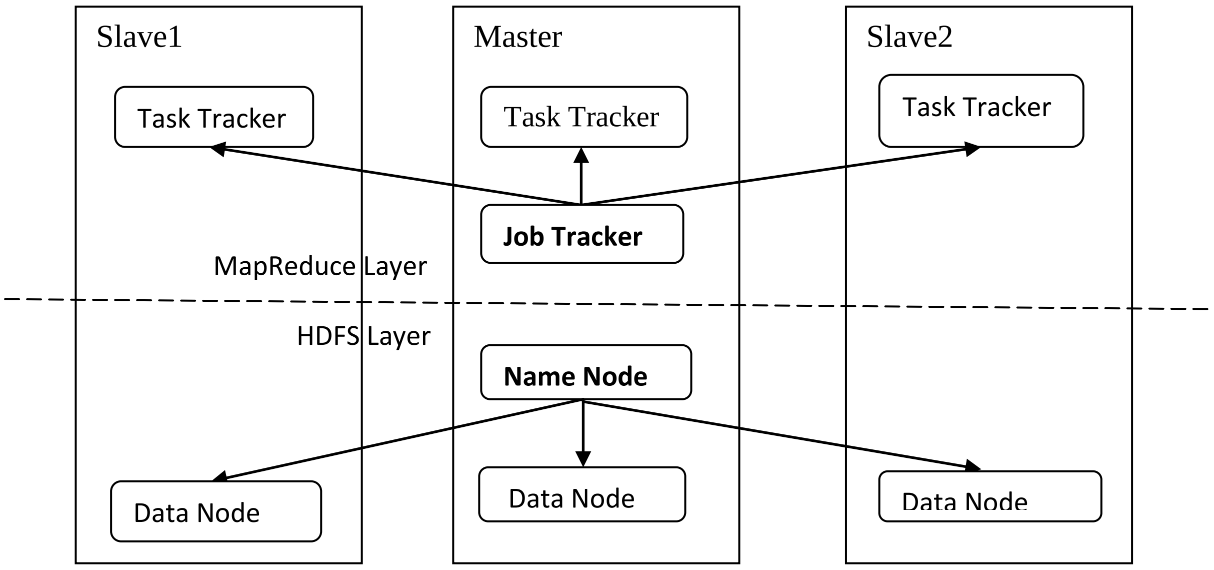


Figure: Illustrating the setup followed for a 3cluster Hadoop setup

vi. Testing Hadoop (In MASTER)

Testing hadoop is similar as done for the single node cluster.

```
bin/hadoop dfs -ls
bin/hadoop dfs -copyFromLocal /hadoop/gutenberg gutenberg
bin/hadoop dfs -ls gutenberg
bin/hadoop dfs -rmr gutenberg-output
bin/hadoop jar hadoop-0.20.2-examples.jar wordcount gutenberg gutenberg-output
```

MAP REDUCE JOB WILL DO IT BY ITS OWN..
In **Slave** we have to check the log files

```
#cat logs/hadoop-hadoop-datanode-slave1.log
```

Then it will display as follows

```
 2010-06-09 15:52:09,873 INFO org.apache.hadoop.mapred.TaskTracker: LaunchTaskAction (registerTask):
attempt_201006091551_0001_m_000002_0 task's state:UNASSIGNED
2010-06-09 15:52:09,878 INFO org.apache.hadoop.mapred.TaskTracker: Trying to launch :
attempt_201006091551_0001_m_000002_0
2010-06-09 15:52:09,878 INFO org.apache.hadoop.mapred.TaskTracker: In TaskLauncher, current free slots : 2 and trying to
launch attempt_201006091551_0001_m_000002_0
2010-06-09 15:52:19,332 INFO org.apache.hadoop.mapred.TaskTracker.clienttrace: src: 137.132.153.177:50060, dest:
137.132.153.177:36085, bytes: 267038, op: MAPRED_SHUFFLE, cliID: attempt_201006091551_0001_m_000002_0
2010-06-09 15:52:20,056 INFO org.apache.hadoop.mapred.TaskTracker: attempt_201006091551_0001_r_000000_0 0.11111112%
reduce > copy (1 of 3 at 0.04 MB/s) >
2010-06-09 15:52:30,988 INFO org.apache.hadoop.mapred.TaskTracker: Received KillTaskAction for task:
attempt_201006091551_0001_r_000000_0
2010-06-09 15:52:30,988 INFO org.apache.hadoop.mapred.TaskTracker: About to purge task:
attempt_201006091551_0001_r_000000_0
2010-06-09 15:52:30,989 INFO org.apache.hadoop.mapred.TaskRunner: attempt_201006091551_0001_r_000000_0 done; removing
files.
2010-06-09 15:52:34,033 INFO org.apache.hadoop.mapred.TaskRunner: attempt_201006091551_0001_m_000002_0 done; removing
files.
2010-06-09 15:52:34,036 INFO org.apache.hadoop.mapred.IndexCache: Map ID attempt_201006091551_0001_m_000003_0 not
found in cache
```

*Then in slave do for **Tasktracker***

```
 INFO org.mortbay.log: Logging to rg.slf4j.impl.Log4jLoggerAdapter(org.mortbay.log) via org.mortbay.log.Slf4jLog

2010-06-09 15:49:50,745 INFO org.apache.hadoop.http.HttpServer: Port returned by webServer.getConnectors()[0].getLocalPort()
before open() is -1. Opening the listener on 50060
2010-06-09 15:49:50,763 INFO org.apache.hadoop.http.HttpServer: listener.getLocalPort() returned 50060
webServer.getConnectors()[0].getLocalPort() returned 50060
2010-06-09 15:49:50,765 INFO org.apache.hadoop.http.HttpServer: Jetty bound to port 50060
2010-06-09 15:49:50,765 INFO org.mortbay.log: jetty-6.1.14
2010-06-09 15:49:51,196 INFO org.mortbay.log: Started SelectChannelConnector@0.0.0.0:50060
2010-06-09 15:49:51,204 INFO org.apache.hadoop.metrics.jvm.JvmMetrics: Initializing JVM Metrics with
processName=TaskTracker, sessionId=
2010-06-09 15:49:51,221 INFO org.apache.hadoop.ipc.metrics.RpcMetrics: Initializing RPC Metrics with hostName=TaskTracker,
port=37444
2010-06-09 15:49:51,287 INFO org.apache.hadoop.ipc.Server: IPC Server Responder: starting

2010-06-09 15:52:13,035 INFO org.apache.hadoop.mapred.JvmManager: In JvmRunner constructed JVM ID:
jvm_201006091551_0001_r_2106942491
2010-06-09 15:52:13,036 INFO org.apache.hadoop.mapred.JvmManager: JVM Runner jvm_201006091551_0001_r_2106942491
spawned.

2010-06-09 15:52:34,032 INFO org.apache.hadoop.mapred.TaskTracker: Received 'KillJobAction' for job: job_201006091551_0001
2010-06-09 15:52:34,033 INFO org.apache.hadoop.mapred.TaskRunner: attempt_201006091551_0001_m_000002_0 done; removing
files.
2010-06-09 15:52:34,034 INFO org.apache.hadoop.mapred.TaskRunner: attempt_201006091551_0001_m_000003_0 done; removing
files.
```

```
2010-06-09 15:52:34,036 INFO org.apache.hadoop.mapred.IndexCache: Map ID attempt_201006091551_0001_m_000003_0 not
found in cache
```

If we want to see the OUTPUT, do the following

bin/hadoop dfs -cat <OUTPUT FILE NAME>

In our example, it is as follows

bin/hadoop dfs -cat gutenberg-output/*

To copy to a specific location in localdisk from hdfs filesystem.

bin/hadoop dfs -copyToLocal <SOURCE> <DESTINATION>

In our example, it is as follows

bin/hadoop dfs -copyToLocal gutenberg-output /hadoop/hadoop/guttenwordcount

For more commands in hdfs refer[6].

5. LARGE SCALE MAPREDUCE SEARCHING ALGORITHM USING HADOOP

i. Introduction

In this chapter, we describe the scheme implemented on hadoop by us for searching the given word and finding the number of occurrences of the same word in given set of documents. Initially the code written for the problem is explained. Then we explain the testing procedure and the results obtained in our case.

ii. The Algorithm

In this work, we consider the problem of searching a particular word from a set of input files. Initially, a Python program is written for word search in Linux platform. In this program, the user can enter a word to search, and the program will display the status based on the search. As an additional feature, we also count the number of occurrences and display the same. The code for this normal word search program is given in Table 4.1. Appendix B gives more details on Pyhton.

Wordsearch.py

```
#!/usr/bin/env python

count = 0

def search(guess,correct):

      global count

      if guess == correct:

            count+=1

searchid=raw_input('Please enter the word to search:')

for line in open('/hadoop/hadoop/20417.txt').xreadlines():
        for word in line.split():
                search(word,searchid)
print "No of occurences of the word "+str(searchid)+" is "+str(count)
```

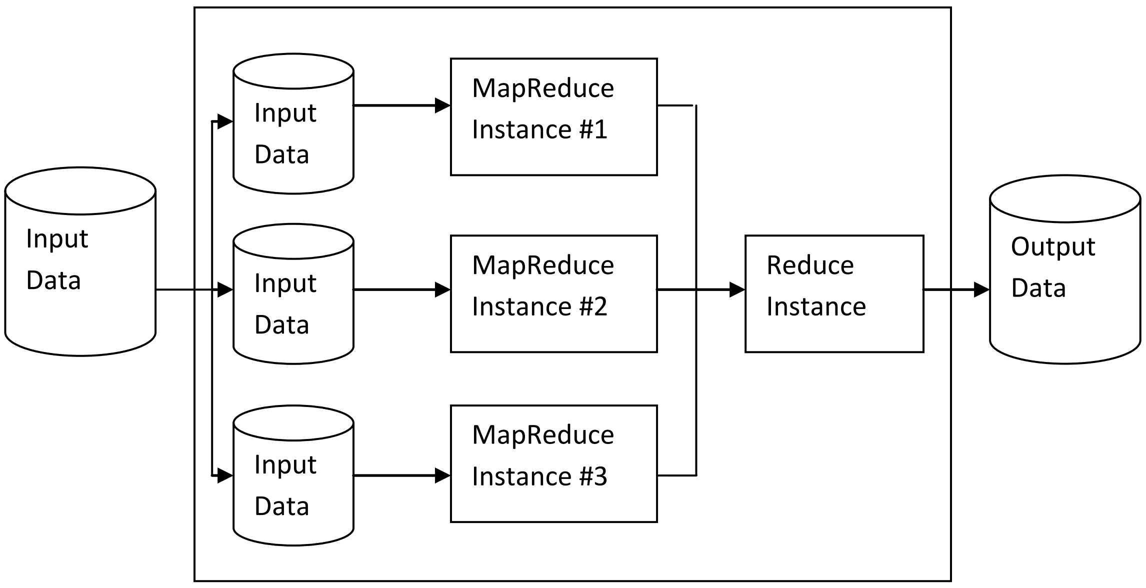
For implementing word search in Hadoop, we need to write separate code for Map and Reduce. During the Map process, the algorithm reads the lines of given input file, split the lines into words and then makes the (key, value) pairs. In this case, the pairs will be (word, “1”). This will be printed onto STDOUT.

```
mapper.py
#!/usr/bin/env python
import sys
# input comes from STDIN (standard input)
for line in sys.stdin:
    # remove leading and trailing whitespace
    line = line.strip()
    # split the line into words
    words = line.split()
    # increase counters
    for word in words:
        # write the results to STDOUT (standard output);
        # what we output here will be the input for the
        # Reduce step, i.e. the input for reducer.py
        #
        # tab-delimited; the trivial word count is 1
        print '%s\t%s' % (word, 1)
```

The reducer accepts this input and combines and generates the new (key`, value`) pairs. It will be (word, count) in our example. Once this process is completed for all words, the given word is searched in the pairs using a function. This output will be displayed in the STDOUT and the program is complete. In our implementation, the word to be searched should be typed in the variable “searchid” in the reducer file.

```
reducer.py
#!/usr/bin/env python
from operator import itemgetter
import sys
searchid = "you"
flag = 0
def search(guess,correct):
        global flag
        if guess == correct:
                flag=1
# maps words to their counts
word2count = {}
# input comes from STDIN
for line in sys.stdin:
    # remove leading and trailing whitespace
    line = line.strip()
    # parse the input we got from mapper.py
    word, count = line.split('\t', 1)
    # convert count (currently a string) to int
    try:
        count = int(count)
word2count[word] = word2count.get(word, 0) + count
    except ValueError:
        # count was not a number, so silently
        # ignore/discard this line
        pass
# sort the words lexigraphically;
#
# this step is NOT required, we just do it so that our
# final output will look more like the official Hadoop
# word count examples
sorted_word2count = sorted(word2count.items(), key=itemgetter(0))
# write the results to STDOUT (standard output)
for word, count in sorted_word2count:
        search(word,searchid)
if flag == 0:
        print "OOPS... The word never appear in this search."
else:
        print "No of occurences of the word "+str(searchid)+" is " + str(word2c$
```

iii. Running the MapReduce Job and Analysis of Results



To run a map reduce job, after copying the Gutenberg files (Refer Chapter 3) and then do the following.

```
##bin/hadoop jar contrib/streaming/hadoop-0.20.2-streaming.jar -file /hadoop/hadoop/mapper.py -mapper

/hadoop/hadoop/mapper.py -file /hadoop/hadoop/reducer.py -reducer /hadoop/hadoop/reducer.py -input

gutenberg/* -output gutenberg-output
```

Then the output will be as follows:

```
hadoop@master:/hadoop/hadoop$ bin/hadoop jar contrib/streaming/hadoop-0.20.2-streaming.jar -mapper
/hadoop/hadoop/mapper.py -reducer /hadoop/hadoop/reducer.py -input gutenberg/* -output gutenberg-output
 additionalConfSpec_:null
 null=@@@userJobConfProps_.get(stream.shipped.hadoopstreaming
 packageJobJar: [/hadoop/hadoop/hadoop-unjar54543/]
 [] /tmp/streamjob54544.jar tmpDir=null
 [...] INFO mapred.FileInputFormat: Total input paths to process : 7
 [...] INFO streaming.StreamJob: getLocalDirs(): [/hadoop /hadoop/mapred/local]
 [...] INFO streaming.StreamJob: Running job: job_200803031615_0021
 [...]
 [...] INFO streaming.StreamJob:  map 0%  reduce 0%
 [...] INFO streaming.StreamJob:  map 43%  reduce 0%
 [...] INFO streaming.StreamJob:  map 86%  reduce 0%
 [...] INFO streaming.StreamJob:  map 100%  reduce 0%
 [...] INFO streaming.StreamJob:  map 100%  reduce 33%
 [...] INFO streaming.StreamJob:  map 100%  reduce 70%
 [...] INFO streaming.StreamJob:  map 100%  reduce 77%
 [...] INFO streaming.StreamJob:  map 100%  reduce 100%
 [...] INFO streaming.StreamJob: Job complete: job_200803031615_0021
 [...] INFO streaming.StreamJob: Output: gutenberg-output  hadoop@master
```

As you can see in the output above, Hadoop also provides a basic web interface for statistics and information. When the Hadoop cluster is running, go to **http://localhost:50030/** and browse around. Here's a screenshot of the Hadoop web interface for the job we just ran.

To see the output, do the following command

#bin/hadoop dfs –cat Gutenberg-output/*

Then it will display as

No of occurrences of the word you is 69

By entering in to the link

http://master:50030/jobdetails.jsp?jobid=job_201006231810_0001&refresh=0

We get the output as follows,

Hadoop job_201006181804_0003 on master

User: hadoop
Job Name: streamjob1742294351578891213.jar
Job File: hdfs://master:5555/tmp/hadoop-hadoop/mapred/system/job_201006181804_0003/job.xml
Job Setup: Successful
Status: Succeeded
Started at: Fri Jun 18 18:25:55 SGT 2010
Finished at: Fri Jun 18 18:26:36 SGT 2010
Finished in: 41sec
Job Cleanup: Successful

Kind	% Complete	Num Tasks	Pending	Running	Complete	Killed	Failed/Killed Task Attempts
map	100.00%	6	0	0	6	0	0 / 0
reduce	100.00	1	0	0	1	0	0 / 0

	Counter	**Map**	**Reduce**	**Total**
Job Counters	Launched reduce tasks	0	0	1
	Launched map tasks	0	0	6
	Data-local map tasks	0	0	6
FileSystemCounters	FILE_BYTES_READ	5,203,570	12,152,540	17,356,110
	HDFS_BYTES_READ	7,279,024	0	7,279,024
	FILE_BYTES_WRITTEN	17,356,332	12,152,540	29,508,872
	HDFS_BYTES_WRITTEN	0	42	42
Map-Reduce Framework	Reduce input groups	0	82,290	82,290
	Combine output records	0	0	0
	Map input records	155,868	0	155,868
	Reduce shuffle bytes	0	9,550,791	9,550,791
	Reduce output records	0	1	1
	Spilled Records	1,794,324	1,258,374	3,052,698
	Map output bytes	9,635,786	0	9,635,786
	Map input bytes	7,279,024	0	7,279,024
	Map output records	1,258,374	0	1,258,374
	Combine input records	0	0	0
	Reduce input records	0	1,258,374	1,258,374

Cluster Summary (Heap Size is 15.5 MB/966.69 MB)

Maps	Reduces	Total Submissions	Nodes	Map Task Capacity	Reduce Task Capacity	Avg. Tasks/Node	Blacklisted Nodes
0	0	2	3	6	6	4.00	0

Completed Jobs

Jobid	Priority	User	Name	Map % Complete	Map Total	Maps Completed	Reduce % Complete	Reduce Total	Reduces Completed	Job Scheduling Information
job_201006181804_0002	NORMAL	hadoop	streamjob2300141973028771637.jar	100%	6	6	100%	1	1	NA
job_201006181804_0003	NORMAL	hadoop	streamjob1742294351578891213.jar	100.00%	6	6	100.00%	1	1	NA

From the above results we can understand that,

- The total number of nodes used is 3.
- Total number of tasks is 6.
- Total size of map input given is 6.94MB.
- Total time taken to obtain the result is 41 sec.
- Number of works failed/killed is 0.
- Map and Reduce percentage completed is 100.
- Map and Reduce task capacity are 6.

When the total size of the input file given is 50MB, we obtain the output as follows

*#bin/hadoop dfs –cat Gutenberg-output/**

Then it will display as

No of occurrences of the word you is 34874

By entering in to the link

http://master:50030/jobdetails.jsp?jobid=job_201006231810_0001&refresh=0

We get the output as follows,

Hadoop job_201006231810_0001 on master

Stage1:

User: hadoop
Job Name: streamjob5578767113618832020.jar
Job File: hdfs://master:5555/tmp/hadoop-hadoop/mapred/system/job_201006231810_0001/job.xml
Job Setup: Successful
Status: Running
Started at: Wed Jun 23 18:29:49 SGT 2010
Running for: 41sec
Job Cleanup: Pending

Kind	% Complete	Num Tasks	Pending	Running	Complete	Killed	Failed/Killed Task Attempts
map	42.85%	42	18	6	18	0	0 / 0
reduce	5.55%	1	0	1	0	0	0 / 0

Stage2:

User: hadoop
Job Name: streamjob5578767113618832020.jar
Job File: hdfs://master:5555/tmp/hadoop-hadoop/mapred/system/job_201006231810_0001/job.xml
Job Setup: Successful
Status: Running
Started at: Wed Jun 23 18:29:49 SGT 2010
Running for: 1mins, 14sec
Job Cleanup: Pending

Kind	% Complete	Num Tasks	Pending	Running	Complete	Killed	Failed/Killed Task Attempts
map	100.00%	42	0	0	42	0	0 / 0
reduce	23.80%	1	0	1	0	0	0 / 0

Stage 3:

User: hadoop
Job Name: streamjob5578767113618832020.jar
Job File: hdfs://master:5555/tmp/hadoop-hadoop/mapred/system/job_201006231810_0001/job.xml
Job Setup: Successful
Status: Succeeded
Started at: Wed Jun 23 18:29:49 SGT 2010
Finished at: Wed Jun 23 18:32:15 SGT 2010
Finished in: 2mins, 26sec
Job Cleanup: Successful

Kind	% Complete	Num Tasks	Pending	Running	Complete	Killed	Failed/Killed Task Attempts
map	100.00%	42	0	0	42	0	0 / 2
reduce	100.00%	1	0	0	1	0	0 / 0

MapReduce Administration

State: RUNNING
Started: Wed Jun 23 18:10:05 SGT 2010

Version: 0.20.2, r911707
Compiled: Fri Feb 19 08:07:34 UTC 2010 by chrisdo
Identifier: 201006231810

Cluster Summary (Heap Size is 15.5 MB/966.69 MB)

Maps	Reduces	Total Submissions	Nodes	Map Task Capacity	Reduce Task Capacity	Avg. Tasks/Node	Blacklisted Nodes
0	0	1	3	6	6	4.00	0

ANALYSIS OF RESULTS

We have executed 2 different cases of word search. In first case the input documents were of size 7MB and the MapReduce job was completed in 41 seconds. In second case, the word search was executed with input files of size about 50MB and it took about 2minutes,26 seconds to complete the same MapReduce job. In both cases, the network setup remained the same (3 slave nodes). We can observe that when the input file size was increased, the time taken was not proportionally increased, but took lesser time than what we expected. This is because, after some time the Mapper and reducer started simultaneously and finish the job in shorter time.

6. CONCLUSIONS AND FUTURE WORK

We implemented large scale searching problem in this work. We considered searching for the number of occurrences of a particular word within a given set of files and output the results. Thus we incorporated finding a given word and also counting the number of times, this word has occurred in the given set of files. We implemented a MapReduce program to accomplish this task. Later we setup a 3-cluster Hadoop system consisting of one master and three slaves (master also acts as a slave). We successfully demonstrated our implementation to search for a particular word in Hadoop system and collected and analyzed the results.

We would like to conduct large scale experiments consisting of larger input files (of the order of Gigabytes) and multi-cluster Hadoop system (consisting of 30-50 nodes in one cluster). Currently we are studying the configuration of multiple Virtual Machines running in the same node. This will enable us to test the scheme in the presence of Virtual Machines. We would also like to extend our algorithm to search a given tag in a set of online web pages dynamically.

7. APPENDIX-A: COMMON PROBLEM FACED FOR HADOOP SETUP AND SOLUTIONS

i. ERROR 1: No Route to Host

I faced an error as given below while running jps in slave1.

```
************************************************************/
2010-06-09 15:47:11,856 INFO org.apache.hadoop.hdfs.server.datanode.DataNode: STARTUP_MSG:
/************************************************************
STARTUP_MSG: Starting DataNode
STARTUP_MSG:   host = slave1/137.132.153.177
STARTUP_MSG:   args = []
STARTUP_MSG:   version = 0.20.2
STARTUP_MSG:   build = https://svn.apache.org/repos/asf/hadoop/common/branches/branch-0.20 -r 911707; compiled by 'chrisdo'
on Fri Feb 19 08:07:34 UTC 2010
************************************************************/
2010-06-09 15:47:13,186 INFO org.apache.hadoop.ipc.Client: Retrying connect to server: master/137.132.153.239:5555. Already
tried 0 time(s).
2010-06-09 15:47:14,190 INFO org.apache.hadoop.ipc.Client: Retrying connect to server: master/137.132.153.239:5555. Already
tried 1 time(s).
2010-06-09 15:47:15,194 INFO org.apache.hadoop.ipc.Client: Retrying connect to server: master/137.132.153.239:5555. Already
tried 2 time(s).
2010-06-09 15:47:16,198 INFO org.apache.hadoop.ipc.Client: Retrying connect to server: master/137.132.153.239:5555. Already
tried 3 time(s).
2010-06-09 15:47:17,202 INFO org.apache.hadoop.ipc.Client: Retrying connect to server: master/137.132.153.239:5555. Already
tried 4 time(s).
2010-06-09 15:47:18,206 INFO org.apache.hadoop.ipc.Client: Retrying connect to server: master/137.132.153.239:5555. Already
tried 5 time(s).
2010-06-09 15:47:19,210 INFO org.apache.hadoop.ipc.Client: Retrying connect to server: master/137.132.153.239:5555. Already
tried 6 time(s).
2010-06-09 15:47:20,214 INFO org.apache.hadoop.ipc.Client: Retrying connect to server: master/137.132.153.239:5555. Already
tried 7 time(s).
2010-06-09 15:47:21,218 INFO org.apache.hadoop.ipc.Client: Retrying connect to server: master/137.132.153.239:5555. Already
tried 8 time(s).
2010-06-09 15:47:22,222 INFO org.apache.hadoop.ipc.Client: Retrying connect to server: master/137.132.153.239:5555. Already
tried 9 time(s).
2010-06-09 15:47:22,225 ERROR org.apache.hadoop.hdfs.server.datanode.DataNode: java.io.IOException: Call to
master/137.132.153.239:5555 failed on local exception: java.net.NoRouteToHostException: No route to host
    at org.apache.hadoop.ipc.Client.wrapException(Client.java:775)
    at org.apache.hadoop.ipc.Client.call(Client.java:743)
    at org.apache.hadoop.ipc.RPC$Invoker.invoke(RPC.java:220)
    at $Proxy4.getProtocolVersion(Unknown Source)
    at org.apache.hadoop.ipc.RPC.getProxy(RPC.java:359)
    at org.apache.hadoop.ipc.RPC.getProxy(RPC.java:346)
    at org.apache.hadoop.ipc.RPC.getProxy(RPC.java:383)
    at org.apache.hadoop.ipc.RPC.waitForProxy(RPC.java:314)
    at org.apache.hadoop.ipc.RPC.waitForProxy(RPC.java:291)
    at org.apache.hadoop.hdfs.server.datanode.DataNode.startDataNode(DataNode.java:269)
    at org.apache.hadoop.hdfs.server.datanode.DataNode.<init>(DataNode.java:216)
    at org.apache.hadoop.hdfs.server.datanode.DataNode.makeInstance(DataNode.java:1283)
    at org.apache.hadoop.hdfs.server.datanode.DataNode.instantiateDataNode(DataNode.java:1238)
    at org.apache.hadoop.hdfs.server.datanode.DataNode.createDataNode(DataNode.java:1246)
    at org.apache.hadoop.hdfs.server.datanode.DataNode.main(DataNode.java:1368)
Caused by: java.net.NoRouteToHostException: No route to host
    at sun.nio.ch.SocketChannelImpl.checkConnect(Native Method)
    at sun.nio.ch.SocketChannelImpl.finishConnect(SocketChannelImpl.java:574)
    at org.apache.hadoop.net.SocketIOWithTimeout.connect(SocketIOWithTimeout.java:206)
```

```
        at org.apache.hadoop.net.NetUtils.connect(NetUtils.java:404)
        at org.apache.hadoop.ipc.Client$Connection.setupIOstreams(Client.java:304)
        at org.apache.hadoop.ipc.Client$Connection.access$1700(Client.java:176)
        at org.apache.hadoop.ipc.Client.getConnection(Client.java:860)
        at org.apache.hadoop.ipc.Client.call(Client.java:720)
        ... 13 more

2010-06-09 15:47:22,226 INFO org.apache.hadoop.hdfs.server.datanode.DataNode: SHUTDOWN_MSG:
/************************************************************
SHUTDOWN_MSG: Shutting down DataNode at slave1/137.132.153.177

************************************************************/
```

Solution
 If such an error occured ,then we should check whether the FIREWALL is DISABLED or not.(For CENTOS this is the path.
System-> Security and Firewall-)

ii. ERROR 2: Datanode not Started While Running jps Command (in Master)

If the data node did not start properly and any error related to occurred in MASTER(By checking in log file)
We have to remove the directory named data which is in /hadoop/hadoop/hdfs
rm -rf data
<u>IMPORTANT</u>

IT IS BETTER TO DELETE */hadoop/hdfs* FILE AND */tmp* FILES IN EVERY NEW CLUSTERING.......

MAY BE DATANODE, NAMENODE AND SECONDARY NAME NODE ISSUES WILL COME BECAUSE OF THIS.........

iii. ERROR 3: Null Pointer Exception

While installing hadoop ...
 "bin/hadoop namenode -format"...it got success
After ,
 "bin/start-all.sh"....if the following error occurs........

Starting namenode, logging to /hadoop/hadoop/bin/../logs/

```
hadoop-hadoop-namenode-localhost.localdomain.out
localhost: ssh: localhost: Name or service not known
hadoop@master's password:
master: starting secondarynamenode, logging to /hadoop/hadoop/bin/../logs/hadoop-hadoop-secondarynamenode-
localhost.localdomain.out
master: Exception in thread "main" java.lang.NullPointerException
master:         at org.apache.hadoop.net.NetUtils.createSocketAddr(NetUtils.java:134)
master:         at org.apache.hadoop.hdfs.server.namenode.NameNode.getAddress(NameNode.java:156)
master:         at org.apache.hadoop.hdfs.server.namenode.NameNode.getAddress(NameNode.java:160)
master:         at org.apache.hadoop.hdfs.server.namenode.SecondaryNameNode.initialize(SecondaryNameNode.java:131)
```

```
master:        at org.apache.hadoop.hdfs.server.namenode.SecondaryNameNode.<init>(SecondaryNameNode.java:115)
master:        at org.apache.hadoop.hdfs.server.namenode.SecondaryNameNode.main(SecondaryNameNode.java:469)
starting jobtracker, logging to /hadoop/hadoop/bin/../logs/hadoop-hadoop-jobtracker-localhost.localdomain.out
localhost: ssh: localhost: Name or service not known
```

SOLUTION

This is because, you may not configured the ssh file properly.

iv. ERROR 4: Encountered While Trying to SSH the Slave

```
[hadoop@master hadoop]$ ssh-copy-id -i /home/hadoop/.ssh/id_dsa hadoop@slave1
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@    WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED!     @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
IT IS POSSIBLE THAT SOMEONE IS DOING SOMETHING NASTY!
Someone could be eavesdropping on you right now (man-in-the-middle attack)!
It is also possible that the RSA host key has just been changed.
The fingerprint for the RSA key sent by the remote host is
ec:59:52:a8:8d:94:5f:25:a9:8b:02:44:c7:df:0b:60.
Please contact your system administrator.
Add correct host key in /home/hadoop/.ssh/known_hosts to get rid of this message.
Offending key in /home/hadoop/.ssh/known_hosts:2
RSA host key for slave1 has changed and you have requested strict checking.
Host key verification failed.
```

For me, I installed centos and set up hadoop on slave 1 and at that time, this ssh error came in master.

SOLUTION:
Go to (.ssh directory where keys are stored)
For me it was /home/hadoop
cd /home/hadoop
rm -rf .ssh
Then repeat the key generation process as described above. It solved this problem for me.
we should give permissions to slave in ***bin*** directory of hadoop folder.by using the following command from the super user(su).

chmod -R 777 *

Then from slave connection in master do the following.....
Log in as hadoop user to each slave and start the datanode and tasktracker:

bin/hadoop-daemon.sh start datanode
bin/hadoop-daemon.sh start tasktracker

v. ERROR 5: java.io.IOException: Incompatible namespaceIDs[7]

If you see the error java.io.IOException: Incompatible namespaceIDs in the logs of a datanode (<HADOOP_INSTALL>/logs/hadoop-hadoop-datanode-<HOSTNAME>.log), chances are you are affected by bug**HADOOP-1212** .I have faced this error many times during my installation.

```
... ERROR org.apache.hadoop.dfs.DataNode: java.io.IOException: Incompatible namespaceIDs in /usr/local/hadoop-
datastore/hadoop-hadoop/dfs/data: namenode namespaceID = 308967713; datanode namespaceID = 113030094
at org.apache.hadoop.dfs.DataStorage.doTransition(DataStorage.java:281)
at org.apache.hadoop.dfs.DataStorage.recoverTransitionRead(DataStorage.java:121)
at org.apache.hadoop.dfs.DataNode.startDataNode(DataNode.java:230)
at org.apache.hadoop.dfs.DataNode.<init>(DataNode.java:199)
at org.apache.hadoop.dfs.DataNode.makeInstance(DataNode.java:1202)
at org.apache.hadoop.dfs.DataNode.run(DataNode.java:1146)
at org.apache.hadoop.dfs.DataNode.createDataNode(DataNode.java:1167)
at org.apache.hadoop.dfs.DataNode.main(DataNode.java:1326)
```

The full error looked like this on my machines:

For more information regarding this issue, read the **bug description**.

At the moment, there seem to be two workarounds as described below.

SOLUTION:

Start from scratch

I can testify that the following steps solve this error, but. The crude workaround I have followed is to:

1. stop the cluster
2. delete the data directory on the problematic datanode: the directory is specified by dfs.data.dir in conf/hdfs-site.xml; if you followed this tutorial, the relevant directory is /hadoop/hadoop/dfs/data
3. reformat the namenode (NOTE: all HDFS data is lost during this process!)
4. restart the cluster

When deleting all the HDFS data and starting from scratch does not sound like a good idea (it might be ok during the initial setup/testing), you might give the second approach a try.

8. APPENDIX- B: PYTHON FOR HADOOP

Python is a general-purpose language. It has been around for quite a while; Guido van Rossum, Python's creator, started developing Python back in 1990.This stable and mature language is very high level, dynamic, object oriented, and cross platform – all characteristics that are attractive to developers. Python runs on all major hardware and operating systems, so it doesn't constrain your platform choices.

Python offers high productivity for all phases of the software life cycles: analysis, design, prototyping, coding, testing, debugging, tuning, documentation, developers, and maintenance. Python provides a unique mix of elegance, simplicity, and power. That's why I used Python for MapReduce job in hadoop. The following is a MapReduce program in hadoop for searching counting the number of occurrences of a particular word in a file.

i. MAP:-Mapper.py

Save the following code in the file /hadoop/hadoop/mapper.py. It will read data from STDIN (standard input), split it into words and output a list of lines mapping words to their (intermediate) counts to STDOUT (standard output). The Map script will not compute an (intermediate) sum of a word's occurrences. Instead, it will output "<word> 1" immediately - even though the <word> might occur multiple times in the input - and just let the subsequent Reduce step do the final sum count.

Make sure the file has execution permission (chmod +x /hadoop/hadoop/mapper.py should do the trick) or you will run into problems.

```
#!/usr/bin/env python

import sys

# input comes from STDIN (standard input)
```

```
for line in sys.stdin:

    # remove leading and trailing whitespace

    line = line.strip()

    # split the line into words

    words = line.split()

    # increase counters

    for word in words:

        # write the results to STDOUT (standard output);

        # what we output here will be the input for the

        # Reduce step, i.e. the input for reducer.py

        #
        # tab-delimited; the trivial word count is 1

        print '%s\t%s' % (word, 1)
```

ii. REDUCE:-Reducer.py

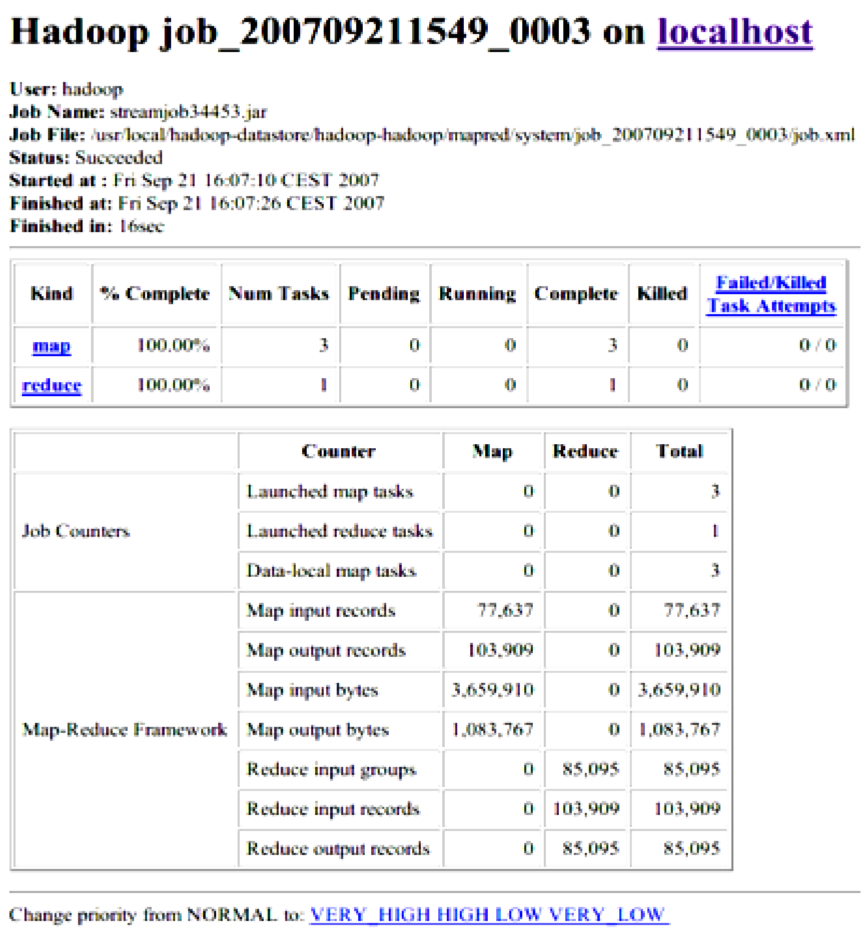
Save the following code in the file /hadoop/hadoop/reducer.py. It will read the results of mapper.py from STDIN (standard input), and sum the occurrences of each word to a final count, and output its results to STDOUT (standard output).

Make sure the file has execution permission (chmod +x /hadoop/hadoop/reducer.py should do the trick) or you will run into problems.

```
#!/usr/bin/env python

 from operator import itemgetter

import sys

# maps words to their counts

word2count = {}
```

```
# input comes from STDIN

for line in sys.stdin:

    # remove leading and trailing whitespace

    line = line.strip()

    # parse the input we got from mapper.py

    word, count = line.split('\t', 1)

    # convert count (currently a string) to int

    try:

        count = int(count)

        word2count[word] = word2count.get(word, 0) + count

    except ValueError:

        # count was not a number, so silently

        # ignore/discard this line

        Pass

# sort the words lexigraphically;

#

# this step is NOT required, we just do it so that our

# final output will look more like the official Hadoop

# word count examples

sorted_word2count = sorted(word2count.items(), key=itemgetter(0))

 # write the results to STDOUT (standard output)

for word, count in sorted_word2count:

    print '%s\t%s'% (word, count)
```

Then output will look something like this.



Hadoop job_200709211549_0003 on localhost

User: hadoop
Job Name: streamjob34453.jar
Job File: /usr/local/hadoop-datastore/hadoop-hadoop/mapred/system/job_200709211549_0003/job.xml
Status: Succeeded
Started at : Fri Sep 21 16:07:10 CEST 2007
Finished at: Fri Sep 21 16:07:26 CEST 2007
Finished in: 16sec

Kind	% Complete	Num Tasks	Pending	Running	Complete	Killed	Failed/Killed Task Attempts
map	100.00%	3	0	0	3	0	0 / 0
reduce	100.00%	1	0	0	1	0	0 / 0

	Counter	Map	Reduce	Total
Job Counters	Launched map tasks	0	0	3
	Launched reduce tasks	0	0	1
	Data-local map tasks	0	0	3
Map-Reduce Framework	Map input records	77,637	0	77,637
	Map output records	103,909	0	103,909
	Map input bytes	3,659,910	0	3,659,910
	Map output bytes	1,083,767	0	1,083,767
	Reduce input groups	0	85,095	85,095
	Reduce input records	0	103,909	103,909
	Reduce output records	0	85,095	85,095

Change priority from NORMAL to: VERY_HIGH HIGH LOW VERY_LOW

9. APPENDIX-C: IMPORTANT TERMS AND DEFINITIONS

i. Namenode and Datanode

A HDFS cluster has two types of node operating in a master-worker pattern: a *namenode* (master) and a number of *datanodes* (workers). The namenode manages the filesystem namespace. It maintains the filesystem tree and the metadata for all the files and directories in the tree. This information is stored persistently on the local disk in the form of two files: the namespace image and the edit log. The namenode also knows the datanodes on which all the blocks for a given file are located, however, it does not store block locations persistently, since this information is reconstructed from datanodes when the system starts.

A *client* accesses the filesystem on behalf of the user by communicating with the namenode and datanodes. The client presents a POSIX-like file system interface, so the user code does not need to know about the namenode and datanode to function. Datanodes are the work horses of the file system. They store and retrieve blocks when they are told to (by clients or the namenode), and they report back to the namenode periodically with lists of blocks that they are storing.

Without the namenode, the file system cannot be used. In fact, if the machine running the namenode was obliterated, all the files on the filesystem would be lost since there would be no way of knowing how to reconstruct the files from the blocks on the datanodes. For this reason, it is important to make the namenode resilient to failure, and Hadoop provides two mechanisms for this.

The first way is to back up the files that make up the persistent state of the filesystem metadata. Hadoop can be configured so that the namenode writes its persistent state to multiple file systems. These writes are synchronous and atomic. The usual configuration choice is to write to local disk as well as a remote NFS mount.

ii. Secondary Namenode

It is also possible to run a *secondary namenode*, which despite its name does not act as a namenode. Its main role is to periodically merge the namespace image with the edit log to prevent the edit log from becoming too large. The secondary namenode usually runs on a separate physical machine, since it requires plenty of CPU and as much memory as the namenode to perform the merge. It keeps a copy of the merged namespace image, which can be used in the event of the namenode failing. However, the state of the secondary namenode lags that of the primary, so in the event of total failure of the primary data, loss is almost guaranteed. The usual course of action in this case is to copy the namenode's metadata files that are on NFS to the secondary and run it as the new primary.

iii. HDFS

The Hadoop Distributed File System (HDFS) is a distributed file system designed to run on commodity hardware. It has many similarities with existing distributed file systems. HDFS is a filesystem designed for storing very large files with streaming data access patterns, running on clusters on commodity hardware. However, the differences from other distributed file systems are significant. HDFS is highly fault-tolerant and is designed to be deployed on low-cost hardware. HDFS provides high throughput access to application data and is suitable for applications that have large data sets. HDFS relaxes a few POSIX requirements to enable streaming access to file system data. HDFS was originally built as infrastructure for the Apache Nutch web search engine project. HDFS is part of the Apache Hadoop Core project. The project URL is http://hadoop.apache.org/core/.

iv. Jobtracker and Tasktracker

The JobTracker is the service within Hadoop that farms out MapReduce tasks to specific nodes in the cluster, ideally the nodes that have the data, or at least are in the same rack.

1. Client applications submit jobs to the Job tracker.
2. The JobTracker talks to the NameNode to determine the location of the data
3. The JobTracker locates TaskTracker nodes with available slots at or near the data
4. The JobTracker submits the work to the chosen TaskTracker nodes.
5. The TaskTracker nodes are monitored. If they do not submit heartbeat signals often enough, they are deemed to have failed and the work is scheduled on a different TaskTracker.
6. The TaskTrackers notify the JobTracker when a task fails. The JobTracker decides what to do then: it may resubmit the job elsewhere, it may mark that specific record as something to avoid, and it may even blacklist the TaskTracker as unreliable.
7. When the work is completed, the JobTracker updates its status.
8. Client applications can poll the JobTracker for information.

The JobTracker is a point of failure for the Map/Reduce infrastructure. If it goes down, all running jobs are lost. The file system remains live.

10. APPENDIX- D: ABBREVIATIONS

AWS	Amazon Web Services
GFS	Graphics Interchange Format
GPS	Global Positioning System
HDFS	Hadoop Distributed File System
HPC	High Performance Computing
HTTP	Hypertext Transfer Protocol
IDC	Internet Database Connector
MPI	Message Passing Interface
OS	Operating System
POSIX	Portable Operating System Interface (for UNIX)
RAID	Redundant Array of Inexpensive Disks
RDBMS	Relational Database Management System
RFID	Radio Frequency Identification
SIMD	Single Instruction Multiple Data
SQL	Structure Query Language
XML	Extensible Mark-up Language

11. REFERENCES

[1] # In January 2007, David J. DeWitt and Michael Stonebraker caused a stir by publishing "MapReduce: A major step backwards" *http://www.databasecolumn.com/2008/01/mapreduce-a-major-step-back.html*)

[2] These specifications are for the Seagate ST-41600n

[3] *http://www.intelligententerprise.com/showArticle.jhtml?articleID=207800705*, *http://blog.familytreemagazine.com/insider/Inside+Ancestrycoms+TopSecret+Data+Center.aspx*, *http://www.archive.org/about/faqs.php*, *http://www.interactions.org/cms/?pid=1027032*.

[4]From Gantz et al., "The Diverse and Exploding Digital Universe," March 2008 (*http://www.emc.com/collateral/analyst-reports/diverse-exploding-digital-universe.pdf*).

[5] http://www.cs.brandeis.edu/~cs147a/lab/hadoop-intro

[6] ***http://hadoop.apache.org/common/docs/r0.17.1/hdfs_shell.html#DFShell***

[7]http://www.michael-noll.com/wiki/Running_Hadoop_On_Ubuntu_Linux_(Multi-Node_Cluster)

[8] Tom White, "Hadoop- the definitive guide", O'Reilly Media,1005,2009

[9] http://en.wikipedia.org/wiki/Hadoop#Hadoop_Distributed_File_System

[10] http://burtonator.files.wordpress.com/2008/01/p107-dean.pdf

[11] http://hadoop.apache.org

[12] http://centos.org/, version 5.0