Programming Assignment 2: MapReduce on Hadoop

In this assignment, you will program the Hadoop parallel data processing platform, developed by Apache. This will allow you gain practical experience on MapReduce programming and learn the performance implication of parallel data processing.

Step 1: Hadoop System Structure:

A Hadoop cluster consists of two parallel systems: the Hadoop Distributed File System (HDFS) and the MapReduce framework. Each system performs a different task, but they work together to process large amounts of data. HDFS stores files in a cluster, spread them so they can contain files that are larger than any individual node could store. Hadoop has programs written in Java and supports scheduling the tasks in such a way that parallel execution of programs is made possible.

To get started, you need to read the tutorial of MapReduce, available at https://hadoop.apache.org/docs/r1.2.1/mapred_tutorial.html, to get an idea on how the technical details are pieced together. The tutorial provides an excellent starting point for understanding the general structure of a program and MapReduce as a template to start your own programs. Please note the code "WordCount v2.0" at the bottom contains many useful features that you want to take a look at.

Hadoop API (http://hadoop.apache.org/docs/stable/api/) is the place to find detailed information on the functions. Other information is also available from the official website of Hadoop (http://hadoop.apache.org/).

Step 2: Deploying Hadoop on Hydra Lab:

Hydra lab is one of the most frequently used lab in the EECS department. The computer's hardware is powerful enough to deploy a small Hadoop cluster. A typical cluster storage capacity (30 Hydra machines) is around 7TB, far more enough for our programming needs. The following steps provide the details on how you can deploy the Hadoop system on Hydra machines for your programming needs. Here are a few points to take care of when get started.

Note: please read the entire Step 2, especially the debugging and ssh configuration at the end, if you meet problems following some of the detailed procedures.

First, when you log into the Hydra machines, the /home/netid directory is actually a networked file system directory. In other words, if you copy a file to /home/netid/ directory via Hydra2 machine, then every other machine can access the data from the same directory path.

Second, the /home/netid directory has a quota limitation that the maximum storage is usually 2GB per user. So your home directory is not enough to support a Hadoop cluster which may contain data in a larger size. However, /local_scratch is the per- machine directory that you can use as much as you want. Apparently, this directory is shared with others, but you can reach 500GB disk size if necessary. We hope that by using this directory as needed, you can develop programs using Hadoop more easily.

Please note that the /local_scratch directory is not shared, and will be deleted within two weeks if there is no data change. So please do not store your important files under this directory.

Fourth, the DNSs of the Hydra lab's machines are very straightforward. For instance, the name of the Hydra machines are hydra1 - hydra30, so the DNSs are hydra1.eecs.utk.edu - hydra30.eecs.utk.edu.

Step-by-step Deployment

After we completely understand the features of Hydra lab, we can take advantages of them to quickly set up a Hadoop cluster.

SSH Permission Problem

During this experiment you may encounter some of the errors from the SSH access permission problem. Specifically, you may not be able to SSH directly from one hydra machine to any other hydra machines, in other words, it requires your netid and password to do so.

If this issue happens, you may need to create an SSH public key without encrypted phrases from any Hydra machine, say hydra1, and copy this public key to any other machine. Note that, your /home/netid directory is in networked file system, so this one time key copy resolve all the issues between any two machines inside the department's infrastructure. The following commands should help.

$ ssh-keygen -t rsa -C ""

When it asks for the filename to save the key, you can either use the default one by clicking enter key or give a new name. I use the default name /home/netid/.ssh/id_rsa here.

Next, you copy key to any other Hydra machine, say hydra7.eecs.utk.edu. Use the following commands.

$ ssh-copy-id -i /home/netid/.ssh/id_rsa.pub hydra7.eecs.utk.edu

You may need to enter your password, but it should be the last time. After this, you should be able to connect from one hydra machine to another without requiring you to use passwords. If you still have problems, read the debugging section later.

Step1: Download Hadoop

To obtain a latest version of Hadoop software stack, you may want to use this link (http://www.apache.org/dyn/closer.cgi/hadoop/common/). There are multiple versions, but I would recommend you to use a stable version, for instance, Hadoop-1.2.1. In this article, I would use Hadoop-1.2.1 as the example.

You may download the Hadoop package into your /home/netid directory and I would recommend you to do so, because the /home/netid directory can be accessed by any Hydra machine under the same location, meaning that you do not need to copy the Hadoop package to every single machine in Hydra lab.

Step 2: Configure Hadoop Environment

The hadoop-env.sh file contains the Hadoop environment parameters, among which the most important parameter is the JAVA_HOME parameter. So edit the ./conf/hadoop-env.sh file by adding the following information.

export JAVA_HOME=/usr/lib/jvm/java-6-openjdk

Besides this parameter, where to store the log file should also be in consideration. Hadoop will generate lots of logs, but our /home/netid directory has a quota limitation. So we have to relocate the log files into the /local_scratch/. Add the following line into the hadoop-env.sh will change it.

export HADOOP_LOG_DIR=/local_scratch/netid/hadoop/logs

Note that here, as well as the following, you need to change netid to your own unique netid assigned to you by the department.

Step 3: Configure Core Site

The core site defines some important parameters related to the whole system's work. For instance the hadoop.tmp.dir and the fs.default.name, which defines the temporary directory and master or namenode for the whole system. The core site configuration file is the ./conf/core-site.xml, and you can use the following XML block.

<configuration>

<property>

<name>hadoop.tmp.dir</name>

<value>/local_scratch/netid/hadoop</value>

</property>

<property>

<name>fs.default.name</name>

<value>hdfs://hydra21.eecs.utk.edu:PORT</value>

</property>

</configuration>

To edit such files under Linux, you may use a text editor such as vi for your needs. Note that you need to replace the PORT with a numerical value. You should choose one value is not conflicting with others, that is, as unique as possible, so a high value > 50000 is appropriate.

Step 4. Configure HDFS Site

As to the HDFS, three very important parameters need to be set in order to make sure the cluster in Hydra lab works properly. The three parameter are the following:

dfs.replication: the number of replications.

dfs.name.dir: the place for the namenode to maintain all the files in the distributed file system.

dfs.data.dir: the place to store the data content on the slave nodes.

The HDFS site configuration is via the ./conf/hdfs-site.xml file. So add the following XML block will apply the changes.

<configuration>

<property>

<name>dfs.replication</name>

<value>3</value>

</property>

<property>

<name>dfs.name.dir</name>

<value>/local_scratch/netid/hadoop/</value>

</property>

<property>

<name>dfs.data.dir</name>

<value>/local_scratch/netid/hadoop/</value>

</property>

</configuration>

Step 5. Configure the Map Reduce Site:

The MapReduce Site information is mainly about the job tracker and its TCP port. So the configuration is simply edit the ./conf/mapred-site.xml through the following XML block. You can change hydra21 to other hosts depending on the machine you use as the master node.

<configuration>

<property>

<name>mapred.job.tracker</name>

<value>hydra21.eecs.utk.edu:PORT+1</value>

</property>

</configuration>

Step 6. Configure Master and Slaves:

Typically, one master and several slave nodes are used in the Hadoop system. So we need to tell the Hadoop system which one is master and which nodes are slaves. In ./conf/masters and ./conf/slaves files, you can list the master and slaves accordingly. For instance, the ./conf/masters can be the following.

hydra21.eecs.utk.edu

And the ./conf/slaves file can be (three machine example):

hydra22.eecs.utk.edu

hydra23.eecs.utk.edu

hydra24.eecs.utk.edu

After finishing the above five steps, your Hadoop cluster in Hydra lab is ready to run.

Step 7. Start the Cluster:

If you are using the cluster for the first time, you will need to format the namenode by the following command.

$ ./bin/hadoop namenode -format

Two steps should be followed to start the cluster. The first one is to start DFS through the following command.

$ ./bin/start-dfs.sh

Then you will see your terminal screen keeps popping message to write log into your HADOOP_LOG_DIR.

Then, you need to start the Map Reduce framework by executing the following command.

$ ./bin/start-mapred.sh

Similarly, you will see the terminal screen keeps saying to write log to your HADOOP_LOG_DIR.

Step 8: try out some applications

By default, Hadoop-1.2.1 provides a jar package which contains some default examples of Map/Reduce applications. Therefore, we can use one of those to validate if the whole system is working correctly.

In this example, we use a simple application that calculates the value of π. You can simply execute the following command to initiate the task.

$./bin/hadoop jar hadoop-examples.jar pi 4 10000

If you don't encounter any errors, you have configured Hadoop successfully. If you meet errors, read the following to see if you can debug them.

HDFS Basic Commands

HDFS is a distributed file system and its shell-like commands are close to the typical Linux shell command. The following basic commands are from the official site (http://hadoop.apache.org/docs/r0.18.3/hdfs_shell.html), which is quite useful.

ls, to list a directory, usage: ./bin/hadoop fs -ls args

mkdir, to create a new directory or file, usage: ./bin/hadoop fs -mkdir args

mv, to move a directory or file from one place to another, usage: ./bin/hadoop fs - mv args

rm, to remove a directory or file, usage: ./bin/hadoop fs -rm args

cp, to copy a directory or file from one place to another, usage: ./bin/hadoop fs -cp args

put, to copy from local to HDFS, usage: ./bin/hadoop fs -put args

get, to copy from HDFS to local, usage: ./bin/hadoop fs -get args

Debugging

Since your particular settings on Hydra machines may be different, here are some tips for handling possible errors that arise during your debugging phase.

- You may meet problems using Hydra machines with lower IDs, such as hydra1 to hydra20. If you meet such problem, it may be safe you can choose to use hydra machines with ID of 21 or larger only.
- If you have problems with setting up SSH properly, you may want to check your directory permissions. For example, in some cases, SSH does not work properly if you allow reading/writing privileges to SSH/Hadoop directories by other users (i.e., you should set the directory rights to be rwx------ under Linux environments to secure the ssh directories and Hadoop directories, so that ssh won't complain errors).
- If you meet an error saying the namenode is running properly, you may check out this link: http://stackoverflow.com/questions/16713011/hadoop-namenode-is-not-starting-up
- If you meet the error: **Error: INFO ipc.Client: Retrying connect to server**, check out this link: http://stackoverflow.com/questions/13295646/hadoop-dfs-error-info-ipc-client-retrying-connect-to-server-localhost
- You may want to use start-all and stop-all under the bin directory to control the services start and stop.
- You may reformat the namenode under some cases to re-initialize the Hadoop file system, if you are trying to restart the cluster.
- You may use the command **jps** to see the current services running for Hadoop on a node.

Step 3: Building the Reverse-indexer

In this programming assignment, your job is to design and implement an important component of an online search engine - the capability for users to search full-text from a document. This assignment involves, for the first step, creating a full inverted index of the contents of the document used as input. Here you are required to use the complete works of Shakespeare as testing input, which is available at:

http://www.gutenberg.org/ebooks/author/65

Your program should support indexing of one or more txt files as input. A full inverted index is a mapping of words to their location (document file and offset) in a set of documents. Most modern search engines utilize some form of a full inverted index to process user-submitted queries. In its most basic form, an inverted index is a simple table which maps words in the documents to some sort of document identifier, while a full inverted index maps words in the documents to some sort of document identifier and offset within the document. For example, if given the following two documents:

Doc1:

Hello

World

You could construct the following inverted file index:

Hello -> (Doc1, 1)

World -> (Doc1, 2)

Here the offset is assumed to be the line number (starting with 1). To use Hadoop, you write two classes: a Mapper and a Reducer. The Mapper class contains a map function, which is called once for each input and outputs any number of intermediate <key, value> pairs. What code you put in the map function depends on the problem you are trying to solve. Let's start with a short example. Suppose the goal is to create an "index" of a body of text -- we are given a text file, and we want to output a list of words annotated with the line-number at which each word appears. For that problem, an appropriate Map strategy is: for each word in the input, output the pair <word, line-number>. For now we can think of the <key, value> pairs as a nice linear list, but in reality, the Hadoop process runs in parallel on many machines. Each process has a little part of the overall Map input (called a map shard), and maintains its own local cache of the Map output. For a description of how it really works, you should read the Google paper: "MapReduce: Simplified Data Processing on Large Clusters", which we discuss in the class. After the Map phase produces the intermediate <key, value> pairs they are efficiently and automatically grouped by key by the Hadoop system in preparation for the Reduce phase (this grouping is known as the Shuffle phase of a map-reduce). For the above example, that means all pairs with the same keys are grouped together.

The Reducer class contains a reduce function, which is then called once for each key. Each reduce looks at all the values for that key and outputs a "summary" value for that key in the final output. Suppose reduce computes a summary value string made of the line numbers sorted into increasing order, then the output of the Reduce phase on the above pairs will produce the pairs that contain the line numbers for each key. Like Map, Reduce is also run in parallel on a group of machines. Each machine is assigned a subset of the keys to work on (known as a reduce shard), and outputs its results into a separate file.

In the indexer assignment, given an input text, we naturally want to create the Map code to extract one word at a time from the input, and the Reduce code to combine all the data for one word. Specifically, you need to program the Hadoop system to divide the (large) input data set into logical "records" and then calls map() once for each record. It is up to you to decide how to implement the details, but since in this example we want to output <word, offset> pairs, the types will both be Text (a basic string wrapper, with UTF8 support).

For the line indexer problem, the map code takes in a line of text and for each word in the line outputs a string key/value pair <word, offset:line>. When run on many machines, each mapper gets part of the input -- so for example with 100 Mbytes of data on 20 mappers, each mapper would get roughly its own 5 Megabytes of data to go through. On a single mapper, map() is called going through the data in its natural order, from start to finish. The Map phase outputs <key, value> pairs, but what data makes up the key and value is totally up to the Mapper code. In this case, the Mapper uses each word as a key, so the reduction below ends up with pairs grouped by word.

The reduce() method is called once for each key; the values parameter contains all of the values for that key. The Reduce code looks at all the values and then outputs a single "summary" value. Given all the values for the key, the Reduce code typically iterates over all the values and either concats the values together in some way to make a large summary object, or combines and reduces the values in some way to yield a short summary value. In this assignment, the summary should include all locations for a word in documents used as input.

Deliverables

Your deliverables include a README file, your source code, and testing result document with some inputs from the works of Shakespeare. Your code should achieve the following:

(30%) Identifying and removing stop words – One issue is that some words are so common that their presence in an inverted index is "noise," that is they can obfuscate the more interesting properties of a document. Such words are called “stop words.” For this part of the assignment, write a word count MapReduce function to perform a word count over a corpus of text files and to identify stop words. It is up to you to choose a reasonable threshold (word count frequency) for stop words, but make sure you provide adequate justification and explanation of your choice. A parser will group words by attributes which are not relevant to their meaning (e.g., "hello", "Hello", and "HELLO" are all the same word), so it is up to you to define how to group such words however you wish; some suggestions include case-insensitivity, etc. It is not required that you treat “run” and “ran” as the same word, but your parser should handle case insensitivity at least. Once you have written your code, then run your code and collect the word counts for submission with all your Mapper and Reducer Java files. As a starting point, you can find the source code for a simple word count map-reduce function in:

hadoop-0.14.2\src\examples\org\apache\hadoop\examples\WordCount.java

(40%) Building the Inverted Index -- For this portion of the assignment, you will design a MapReduce-based algorithm to calculate the inverted index. To this end, you are to create a full inverted index, which maps words to their document ID + line number in the document. Note that your final inverted index should not contain the words identified in Step 1. How you choose to remove those words is up to you; one possibility is to create multiple MapReduce passes, but there are many possible ways to do this. Your submitted indexer should be able to run successfully on one or multiple input txt files, where "successfully" means it should run to completion without errors or exceptions, and generate the correct mappings. You are required to submit all relevant Mapper and Reducer Java files, in addition to any supporting code or utilities.

(30%) Query the Inverted Index -- Write a query program on top of your full inverted file index that accepts a user-specified query (one or more words) and returns not only the document IDs but also the locations in the form of line numbers. The query program also returns a text "snippet" from each document showing where the query term appears in the document, by extracting such snippet from the original text.

(Extra credit: 20%) Improve the Query: Your query format should support "and", "or", and "not" operations for text in the same line. Your query should also support multiple words, like “fare well”, if used as a query, should return only lines where these two words appear consecutively. Note that to achieve this extra credit, you need to change your earlier implementation on building the inverted table such that the precise word locations in addition to the line numbers are included so that you can find whether two words are consecutive or not.

Notes

Note that this programming assignment, just like the previous one, does not have a single correct solution. You need to make design decisions. So please document them in the README file, and also include detailed instructions on how to use your code. You need to put the sample input and output results into the testing report for the grading purposes.