Lab 2: MapReduce Programming

1. Objectives

- Understand the MapReduce concept.
- Get familiar with the Hadoop framework.
- Experience working with small Hadoop cluster using VMs.

2. Equipment Needs

- Computers
- Internet

3. Experiments

3.1 Basics

a) Go through the Apache Hadoop introduction to get the general idea about Hadoop: http://hadoop.apache.org/
b) Go through the Apache Hadoop release notes to understand the evolution of Hadoop: http://hadoop.apache.org/releases.html

3.2 Hadoop Single Node Mode.

a) Follow the instructions on http://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-common/SingleCluster.html to set up Hadoop environment on your own Linux machine.
b) Follow the instructions/tutorials from the above link to run simple practice with single node mode.

3.3 Hadoop "cluster"

Create two VMs (one master and one slave) on your own computer (ex: Virtualbox) and construct a small hadoop cluster for running the word count program. You can also use Docker containers to perform this assignment.

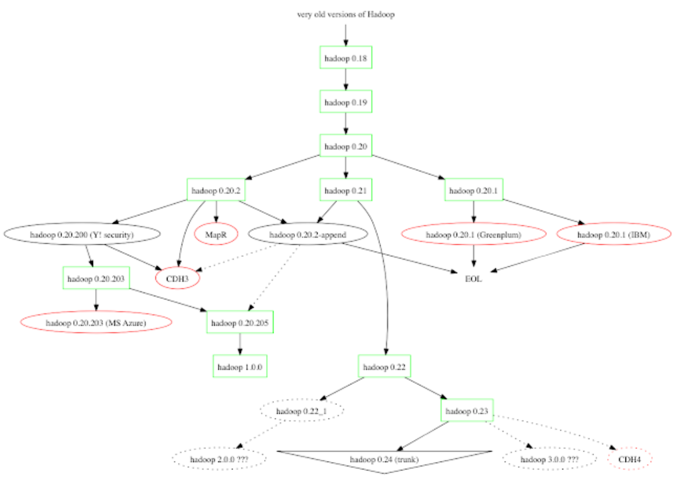
a) Create two VMs.
b) Configure the VM network so that the VMs can ping and communicate with each other.
c) Download and install hadoop and all required tools.
d) Configure hadoop configure files for your two-VM cluster
e) Run the WordCount Hadoop job on the file (wordCountText.txt) provided by the TA.

4. Reports

(a) What are the differences between Hadoop 0.X, 1.X, 2.X and 3.X?

First of all, the major different between 1.X and 2.X is YARN. The NameNode and JobTracker are replaced by ResourceManager and NodeManager in YARN. The design of YARN is to make the structure more scalable, since it has multi NameNode. (1.X can only support upto 4000 Nodes per cluster) Therefore, Hadoop 1.X is sutable for single purpose system, since it can only support the framework of MapReduce; Hadoop 2.X solve the scalability problem with multiple programming models.

According to the roadmap (https://wiki.apache.org/hadoop/Roadmap), we can learn that 2.X is actually not an improvement of 1.X. Instead, they are developed in parallel. 1.X is a branch of 0.20, after 0.21 is released; 2.X is a branch of 0.21. A graph from this website shows the diagram of the version branching. (https://blogs.apache.org/bigtop/entry/all_you_wanted_to_know)



The structure of Hadoop 3.X is similar with Hadoop 2.X. The main novel feature of 3.X is "Classpath isolation on by default". Other plans of Hadoop 3.X is still not very clear.

Back to 0.X, from the diagram, we can see that it is the main branching of these versions. When a branch is slower than the next 0.2X, it becomes an independent branch. (eg. 1.X, 2.X)

(b) What is YARN? Why do we need YARN?

YARN is short for "Yet Another Resource Negotiator". It provides a more flexible framework for Hadoop to run more than just MapReduce applications by introducing higher-level cluster management.

Firstly, YARN can support more kinds of application. In Hadoop 1.X, users can only submit MapReduce based applications to Job Tracker. The program should follow the framework of MapReduce; however, YARN supports various applications.

Next, YARN provide higher-level cluster management that dynamically improves the utilization of clusters. In Hadoop 1.X, it relies on a single node NameNode to manage the state of an application, whose setting is static; however, in YARN, application manager can be in one of the nodes, managed by NodeMenager. Moreover, with ResourceManager, who focus on scheduling and resource management, YARN has a better scalability.

(c) What is Hadoop streaming?

Hadoop streaming is a utility that allows users to run Map/Reduce jobs with any external executable program/script. By assigning a mapper and reducer with "-mapper <mapper> -reducer <reducer>", users can run Hadoop with the external mapper and reducer. With Hadoop streaming, users can program their own mapper/reducer independently with different programming languages.

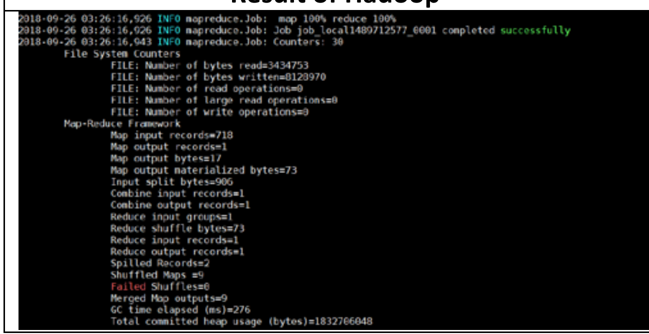
(d) Screenshots of the practice of single node mode Hadoop on your own computer. The screenshot should show the output and result of Hadoop execution as well as the files in HDFS.
(ref: http://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-common/SingleCluster.html)
"Follow the instructions/tutorials from the above link to run simple practice with single node mode."
This doesn't specific which practice, so we include every example.

1. Standalone Operation

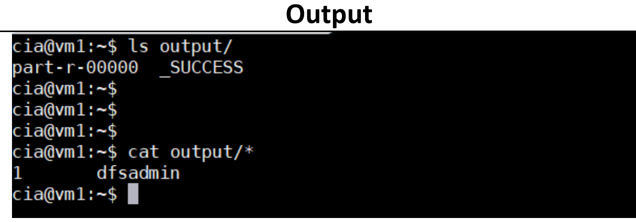
The example copies the unpacked conf directory to use as input and then finds and displays every match of the given regular expression.

```
$ mkdir input
$ cp etc/hadoop/*.xml input
$ bin/hadoop jar share/hadoop/mapreduce/hadoop-mapreduce-examples-2.9.1.jar grep input output 'dfs[a-z.]+'
$ cat output/*
```

Result of Hadoop



Output



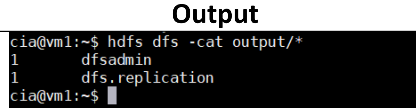
Successfully find dfs*.

2. Pseudo-Distributed Operation

$ bin/hdfs dfs -mkdir input
$ bin/hdfs dfs -put etc/hadoop/*xml input
$ bin/hadoop jar share/hadoop/mapreduce/hadoop-mapreduce-examples-2.9.1.jar grep input output 'dfs[a-z.]+'
$ bin/hdfs dfs -cat output/*

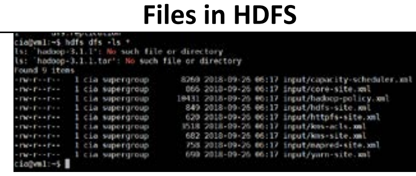
Result of Hadoop

Output



Instead of copy whole "hadoop" folder, I copied "*.xml" to get the result. (It returns nothing if I copy whole folder.)

Files in HDFS



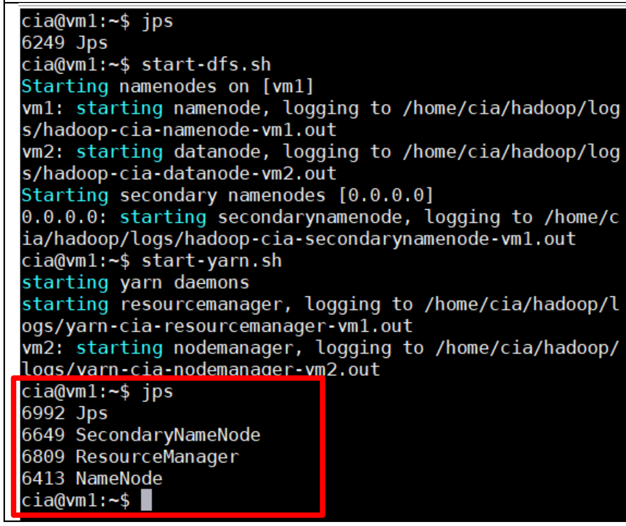
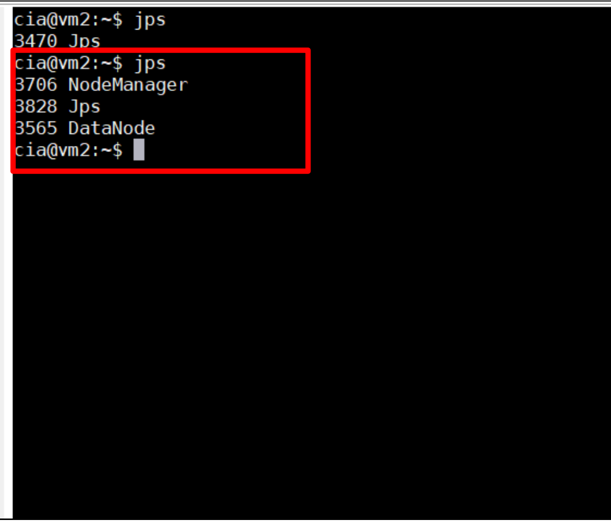
$ bin/hdfs dfs -ls *
("bin" is added to PATH, so can be removed in command.)

(e) Run WordCount on the Hadoop cluster with 2 VMs, what are the top 5 most frequent word in the provided txt file?

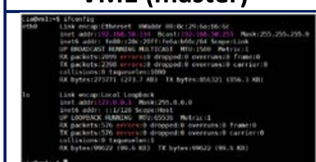
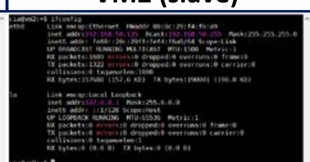
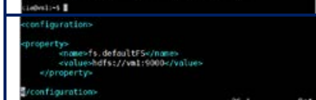
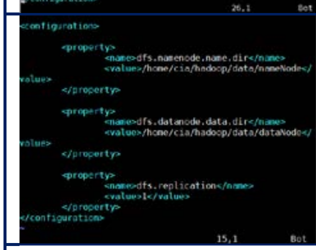
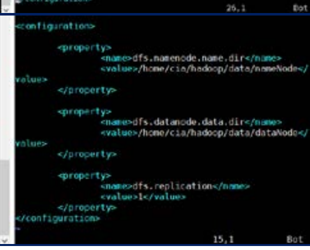
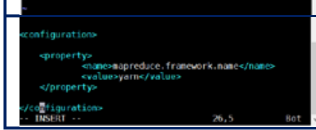
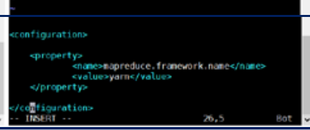
Word	Appear times
ashkan	106228
soheil	67768

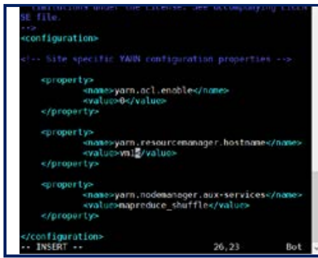
I	55057
frank	47661
in	27613

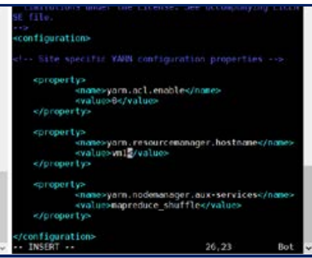
(f) Use jps commands on both VMs to show running Hadoop daemons and provide and screenshots.

VM1 (master)	VM2 (slave)
	

(g) Screenshots of configuration files and IP addresses for Master node and Slave node of your small cluster as well as the MapReduce execution result. For each configuration file, please also briefly explain what it does.

VM1 (master)	VM2 (slave)	Description
		**IP addresses:** Master: 192.168.50.134 Slave: 192.168.50.135
		**core-site.xml:** Mainly configure the URI of NameNode, which is the master.
		**hdfs-site.xml:** Configure the nameNode and dataNode. Here I assign the directory for nameNode and dataNode for further observation.
		**Slaves:** Configure the hosts of the slaves.
		**mapred-site.xml:** Configure the framework for MapReduce.





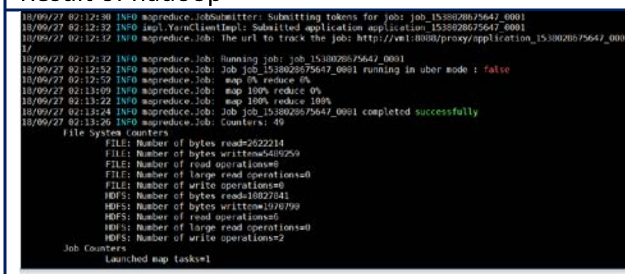
yarn-site.xml: Configure the URI/host of yarn.

Result

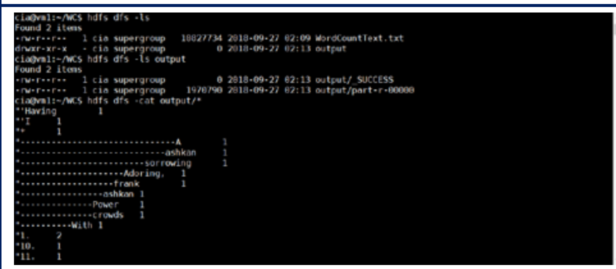
$ hadoop jar WordCount.jar WordCountText.txt output

$ hdfs dfs -cat output/*

Result of hadoop



Cat the results



(h) What are the differences between Hadoop master and slave nodes? Also name what functionalities are performed on each node.

In our experiment setting, our master node runs: **NameNode, SecondaryNameNode, ResourceManager**; our slave node runs: **DataNode, NodeManager**. In brief, master node mainly allocates the resource and assigns jobs, while slave node mainly execute the jobs.

From HDFS's point of view, the master node manages HDFS storage with NameNode, and the slave node have to handle the HDFS service to let NameNode store files. Moreover, the slave node will enable TaskTracker for managing the map/reduce taskes if a task is assigned to the node in Hadoop 1.X.

From YARN's point of view, the master node handle the scheduling of tasks and management of the resources in clusters with ResourceManager; on the other hand, the slave node manages its own resource and reports back to ResourceManager. Moreover, the slave nodes hold containers that can be Application Master or run tasks. When an application is assigned to YARN, the ApplicationMaster on the slave node will track the progress on the cluster, and the tasks will be executed on slave nodes.

(i) Write a pseudo code to multiply large matrices using Hadoop. Also explain the function of your Mappers and Reducers.

Firstly, the problem does not provide the input of the large matrices, so we chose the one that is commonly used in the similar problem in Hadoop. The reason we need to specific the input format is that the mapping in Hadoop is divided with the new line in the file. Different kinds of input format will affect how the data is divided, and affect how mapping function should be designed according to the format. For example, in the following table, Case1 and Case2 has different data formats. In Case1, the mapper function receives "M,0,0,10", while in Case2, the mapper function receives "1 2 3 4". The former one only receives one element of a matrix, and the latter one receives four element of the matrix. Therefore, in this answer, we take Case1 as our input format.

Case1. Input (Matrix M)	Case2. Input (Matrix M)
M,0,0,10 <Matrix ID, row, col, data>	1 2 3 4 <data data data data> (row 1)
M,0,2,9	5 6 7 8
M,0,3,9	9 10 11 12
M,0,5,9	13 14 15 16
M,0,6,9	17 18 19 20
…	…

Here is the pseudo code of our matrix multiplying function:

High level design:

$$MN = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \begin{bmatrix} a & e \\ b & f \\ c & g \end{bmatrix} = \begin{bmatrix} 1a+2b+3c & 1e+2f+3g \\ 4a+5b+6c & 4e+5f+6g \end{bmatrix}$$

From the example, we find out that first row of matrix M repeats two times for multiplying the columns of matrix N. Therefore, we can divide the multiplications with MapReduce.
First, in map stage, we need to duplicate the rows of M and columns of N for the multiplication, and then shuffle the elements according to the position of answer matrix (i, k). After the stage, we can get key-value pairs with the (full or part) list of vectors that ready for inner product at position (i, k).
Then, in reduce stage, we perform the inner product for each key (i, k), and get the result.

Notations:
M, N: ID of first matrix (Our goal is MN)
M: i x j, N: j x k (where i, j, k are size of the matrices)
Mij, Njk: elements in M, N matrix
ANS_{ik}: elements of answer matrix

Mapping Function

(input: [M, i, j, Mij] or [N, j, k, Njk])

1. **If** input[ID] equals to 'M':
2. **for** *count* from 1 to k:
3. Set (key, value pairs) as ((i, *count*), ('M', j, Mij))
4. **else if** input[ID] equals to 'N':
5. **for** *count* from 1 to i:
6. Set (key, value pairs) as ((*count*, k), ('N', j, Njk))
7. **return** Set of (key, value pairs)

Reduce Function

input: Set of shuffled (key, value pairs):
(**(i,k)**, [('M',1,M_{i1}), ('M',2,M_{i2}),… ('M',j,M_{ij}), ('N',1,N_{1k}), ('N',2,N_{2k}),… ('N',j,N_{jk})])
value pair: (ID, j, data)

1. Sort the values with value pair[ID] equals to 'M' by value pair[j] and get *arrayM* //row i of M
2. Sort the values with value pair[ID] equals to 'N' by value pair[j] and get *arrayN* //col i of M

3. $ANS_{ik} \leftarrow 0$
4. **for** *index* from 1 to j:
5. $ANS_{ik} \leftarrow ANS_{ik}$ + *arraryM*[*index*] * *arrayN*[*index*]
6. **return** (key, element): ((i,k), ANS_{ik})

(j) What is a combiner? Add a combiner to the last question and explain its function.

Combiner is a "sub-reducer" between mapper and reducer. It performs the pre-processing of the data and reduces the amount of data that needs to be sent to the reducer. The combiner runs on each node, so the input of the combiner is the output of the node instead of whole data. By doing the pre-processing, combiner can reduce the size of the output, and reduce the transition cost on the network.

To add the combiner, we need to first observe that in each node, what can be pre-processed. From the pseudo-code in (i), we can see that the multiplication is handled on reducer, so we can move some of the multiplication to combiner to avoid sending whole list of value pair to the reducer.

Before we start, we have to know that the list of might value pair not be fully gathered, so we need to first check if the data is gathered in the node. We defined a new ID: "S" to note that the multiplication of (i, k) is done in combiner. Thus on reducer, we can identify which are solved and which are not.

Mapping Function

(input: [M, i, j, Mij] or [N, j, k, Njk])

1. **If** input[ID] equals to 'M':
2.   **for** *count* from 1 to k:
3.   Set (key, value pairs) as ((i, *count*), ('M', j, Mij))
4. **else if** input[ID] equals to 'N':
5.   **for** *count* from 1 to i:
6.   Set (key, value pairs) as ((*count*, k), ('N', j, Njk))
7. **return** Set of (key, value pairs)

Combiner Function

input: Set of shuffled (key, value pairs):

(**(i,k)**, [('M',1,M_{i1}), ('M',2,M_{i2}),… ('M',j,M_{ij}), ('N',1,N_{1k}), ('N',2,N_{2k}),… ('N',j,N_{jk})])

value pair: (ID, j, data)

1. **if** size of (value pairs) equals to 2*j: **// All elements of this pair are gathered, then do "reduce"**
2.   Sort the values with value pair[ID] equals to 'M' by value pair[j] and get *arrayM* //row i of M
3.   Sort the values with value pair[ID] equals to 'N' by value pair[j] and get *arrayN* //col i of M
4. $ANS_{ik} \leftarrow 0$
5.   **for** *index* from 1 to j:
6. $ANS_{ik} \leftarrow ANS_{ik}$ + *arraryM*[*index*] * *arrayN*[*index*]
7.   Set (key, value pairs) as ((i, k), ('S', j, ANS_{ik}))
8. **else**: // If elements are not gathered, then keep the key-value pair
9.   Set (key, value pairs) as ((i, k), (ID, j, data))
10. **return** Set of (key, value pairs)

Reduce Function

input: Set of shuffled (key, value pairs):

(**(i,k)**, [('M',1,M_{i1}), ('M',2,M_{i2}),… ('M',j,M_{ij}), ('N',1,N_{1k}), ('N',2,N_{2k}),… ('N',j,N_{jk})])

value pair: (ID, j, data)

if value pairs [first][ID] equals to ‘S’: **// Means that the multiplication is done in combiner**

Set (key, value pairs) as ((i, k), ANS_{ik})

else:

Sort the values with value pair[ID] equals to ‘M’ by value pair[j] and get *arrayM* //row i of M

Sort the values with value pair[ID] equals to ‘N’ by value pair[j] and get *arrayN* //col i of M

$ANS_{ik} \leftarrow 0$

for *index* from 1 to j:

$ANS_{ik} \leftarrow ANS_{ik}$ + *arraryM*[*index*] * *arrayN*[*index*]

Set (key, value pairs) as ((i, k), ANS_{ik})

return Set of (key, element): ((i,k), ANS_{ik})

Ref: Compile Java to Jar: http://www.skylit.com/javamethods/faqs/createjar.html

Install: https://www.linode.com/docs/databases/hadoop/how-to-install-and-set-up-hadoop-cluster/

We have zero tolerance to forged or fabricated data!! A single piece of forged/fabricated data would bring the total score down to zero.