Lab 3: MapReduce Programming

Name: Ranran Lyu ID: rl3783 Date: 03/13/2020

Name: Lingfeng Zhao ID: lz1973 Date: 03/13/2020

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

1. Hadoop 0.X has only HDFS. Hadoop 1.X has two major components: HDFS(HDFS V1), MapReduce(MR V1). Hadoop 2.X has three major components: HDFS V.2, YARN(MR V2), MapReduce(MR V1)

2. Hadoop 1.x supports only MapReduce (MR) processing. It does not support non-MR tools. Hadoop 2.x allows to work in MR as well as other distributed computing models like Spark, Hama, Giraph, Message Passing Interface) MPI & HBase coprocessors.

3. MR of Hadoop 1.X does both processing and cluster resource management. YARN (Yet Another Resource Negotiator) of Hadoop 2.X does cluster resource management and processing is done using different processing models.

4. Hadoop 1.x has limited scaling of nodes. Limited to 4000 nodes per cluster. 2.x has better scalability. Scalable up to 10000 nodes per cluster.

5. Hadoop 1.X works on concepts of slots – slots can run either a Map task or a Reduce task only. 2.X works on concepts of containers. Using containers can run generic tasks.

6. Hadoop 1.X just has a single Namenode to manage the entire namespace. 2.X has multiple Namenode servers manage multiple namespace.

7. Hadoop 1.x has Single-Point-of-Failure (SPOF) – because of single Namenode- and in case of Namenode failure, needs manual intervention to overcome. 2.x has feature to overcome SPOF with a standby Namenode and in case of Namenode failure, it is configured for automatic recovery.

8. MR API is compatible with Hadoop 1x. A program written in Hadoop1 executes in Hadoop1x without any additional files. MR API requires additional files for a program written in Hadoop1x to execute in Hadoop2x.

9. Hadoop 1.x has a limitation to serve as a platform for event processing, streaming and real-time operations. 2.X can serve as a platform for a wide variety of data analytics-possible to run event processing, streaming and real time operations.

10. Hadoop 1.X does not support for Microsoft windows, but Hadoop 2.X can be deployed on windows.

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

YARN is one of the core components of the open-source Apache Hadoop distributed processing frameworks which helps in job scheduling of various applications and resource management in the cluster. YARN was initially called 'MapReduce 2' since it took the original MapReduce to another level by giving new and better approaches for decoupling MapReduce resource management for scheduling capabilities from the data processing unit.

YARN is being extensively used for writing applications by Hadoop Developers. It lets them create applications, work with huge amounts of data, and manipulate them in an efficient manner. YARN is much more effective and versatile than Hadoop MapReduce, and this is exactly what is required in a world inundated with big data. However, it will remain the most sought-after tool until the perennial search—for a tool that works well in the challenging environment of Big Data Hadoop—comes up with a new befitting tool.

There is a few reasons why we need YARN:

Despite being thoroughly proficient at data processing and computations, Hadoop had some shortcomings like delays in batch processing, scalability issues, etc. as it relied on MapReduce for processing big datasets. With YARN, Hadoop is now able to support a variety of processing approaches and has a larger array of applications. Hadoop YARN clusters are now able to run stream data processing and interactive querying side by side with MapReduce batch jobs. YARN framework runs even the non-MapReduce applications, thus overcoming the shortcomings of Hadoop 1.0.

The architecture of YARN ensures that the Hadoop cluster can be enhanced in the following ways:

- Multi-tenancy

YARN lets you access various proprietary and open-source engines for deploying Hadoop as a standard for real-time, interactive, and batch processing tasks that are able to access the same dataset and parse it.

- Cluster Utilization

YARN lets you use the Hadoop cluster in a dynamic way, rather than in a static manner by which MapReduce applications were using it, and this is a better and optimized way of utilizing the cluster.

- Scalability

YARN gives the power of scalability to the Hadoop cluster. YARN Resource Manager (RM) service is the central controlling authority for resource management and it makes allocation decisions.

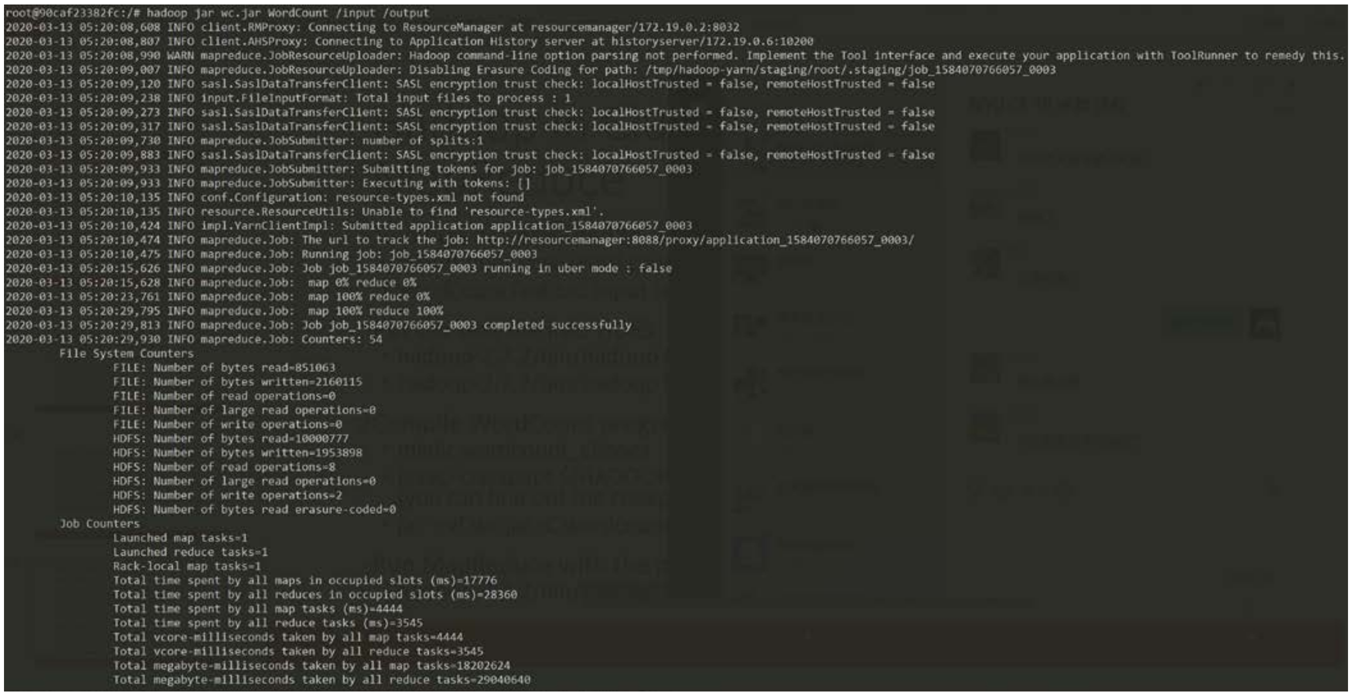
- Compatibility

YARN tool is highly compatible with the existing Hadoop MapReduce applications, and thus those projects that are working with MapReduce in Hadoop 1.0 can easily move on to Hadoop 2.0 with YARN without any difficulty, ensuring complete compatibility.

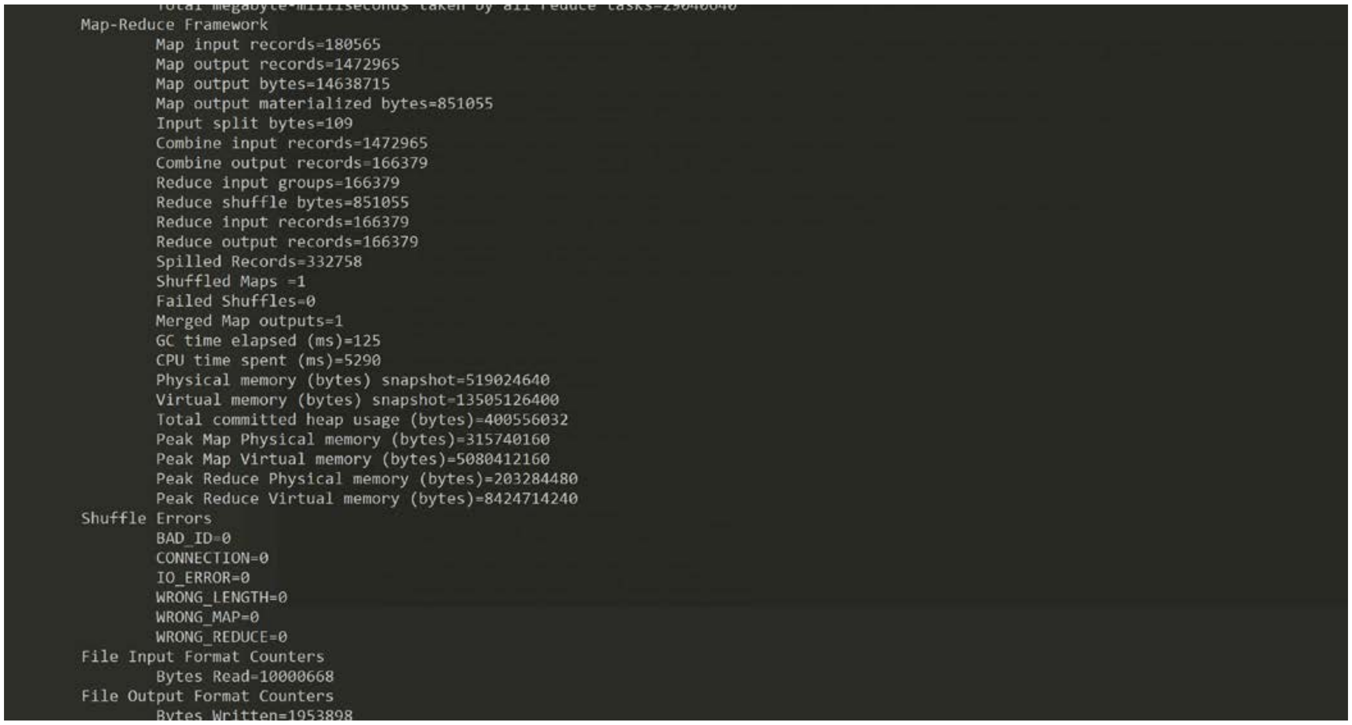
(c) What is Hadoop streaming?

Hadoop Streaming is a generic API which allows writing Mappers and Reduces in any language. But the basic concept remains the same. Mappers and Reducers receive their input and output on stdin and stdout as (key, value) pairs. Apache Hadoop uses streams as per UNIX standard between your application and Hadoop system.

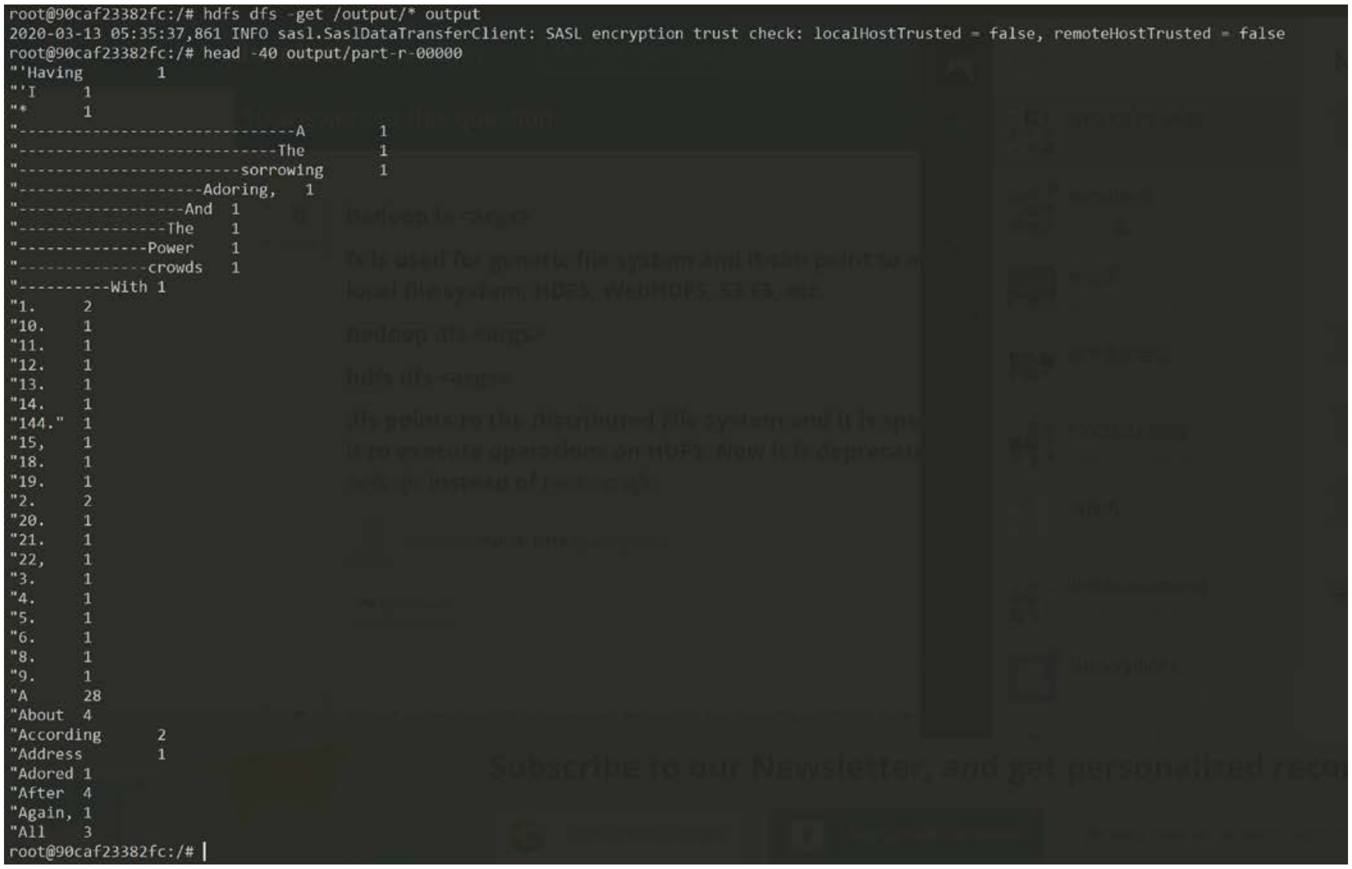
(d) Screenshots of the practice of single node mode Hadoop on your own computer. The screenshot should show the output and result of Hadoop execution as well as the files in HDFS. (ref: http://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-common/SingleCluster.html)



```
root@90caf23382fc:/# hadoop jar wc.jar WordCount /input /output
2020-03-13 05:20:08,608 INFO client.RMProxy: Connecting to ResourceManager at resourcemanager/172.19.0.2:8032
2020-03-13 05:20:08,807 INFO client.AHSProxy: Connecting to Application History server at historyserver/172.19.0.6:10200
2020-03-13 05:20:08,990 WARN mapreduce.JobResourceUploader: Hadoop command-line option parsing not performed. Implement the Tool interface and execute your application with ToolRunner to remedy this.
2020-03-13 05:20:09,007 INFO mapreduce.JobResourceUploader: Disabling Erasure Coding for path: /tmp/hadoop-yarn/staging/root/.staging/job_1584070766057_0003
2020-03-13 05:20:09,120 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localHostTrusted = false, remoteHostTrusted = false
2020-03-13 05:20:09,238 INFO input.FileInputFormat: Total input files to process : 1
2020-03-13 05:20:09,273 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localHostTrusted = false, remoteHostTrusted = false
2020-03-13 05:20:09,317 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localHostTrusted = false, remoteHostTrusted = false
2020-03-13 05:20:09,730 INFO mapreduce.JobSubmitter: number of splits:1
2020-03-13 05:20:09,883 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localHostTrusted = false, remoteHostTrusted = false
2020-03-13 05:20:09,933 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1584070766057_0003
2020-03-13 05:20:09,933 INFO mapreduce.JobSubmitter: Executing with tokens: []
2020-03-13 05:20:10,135 INFO conf.Configuration: resource-types.xml not found
2020-03-13 05:20:10,135 INFO resource.ResourceUtils: Unable to find 'resource-types.xml'.
2020-03-13 05:20:10,424 INFO impl.YarnClientImpl: Submitted application application_1584070766057_0003
2020-03-13 05:20:10,474 INFO mapreduce.Job: The url to track the job: http://resourcemanager:8088/proxy/application_1584070766057_0003/
2020-03-13 05:20:10,475 INFO mapreduce.Job: Running job: job_1584070766057_0003
2020-03-13 05:20:15,626 INFO mapreduce.Job: Job job_1584070766057_0003 running in uber mode : false
2020-03-13 05:20:15,628 INFO mapreduce.Job:  map 0% reduce 0%
2020-03-13 05:20:23,761 INFO mapreduce.Job:  map 100% reduce 0%
2020-03-13 05:20:29,795 INFO mapreduce.Job:  map 100% reduce 100%
2020-03-13 05:20:29,813 INFO mapreduce.Job: Job job_1584070766057_0003 completed successfully
2020-03-13 05:20:29,930 INFO mapreduce.Job: Counters: 54
        File System Counters
                FILE: Number of bytes read=851063
                FILE: Number of bytes written=2160115
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=10000777
                HDFS: Number of bytes written=1953898
                HDFS: Number of read operations=8
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=2
                HDFS: Number of bytes read erasure-coded=0
        Job Counters
                Launched map tasks=1
                Launched reduce tasks=1
                Rack-local map tasks=1
                Total time spent by all maps in occupied slots (ms)=17776
                Total time spent by all reduces in occupied slots (ms)=28360
                Total time spent by all map tasks (ms)=4444
                Total time spent by all reduce tasks (ms)=3545
                Total vcore-milliseconds taken by all map tasks=4444
                Total vcore-milliseconds taken by all reduce tasks=3545
                Total megabyte-milliseconds taken by all map tasks=18202624
                Total megabyte-milliseconds taken by all reduce tasks=29040640
```

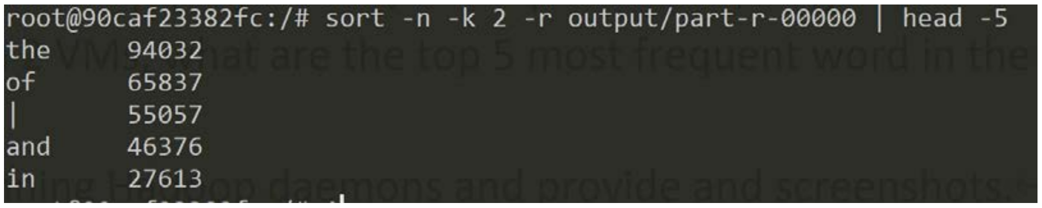


```
                Total megabyte-milliseconds taken by all reduce tasks=29040640
        Map-Reduce Framework
                Map input records=180565
                Map output records=1472965
                Map output bytes=14638715
                Map output materialized bytes=851055
                Input split bytes=109
                Combine input records=1472965
                Combine output records=166379
                Reduce input groups=166379
                Reduce shuffle bytes=851055
                Reduce input records=166379
                Reduce output records=166379
                Spilled Records=332758
                Shuffled Maps =1
                Failed Shuffles=0
                Merged Map outputs=1
                GC time elapsed (ms)=125
                CPU time spent (ms)=5290
                Physical memory (bytes) snapshot=519024640
                Virtual memory (bytes) snapshot=13505126400
                Total committed heap usage (bytes)=400556032
                Peak Map Physical memory (bytes)=315740160
                Peak Map Virtual memory (bytes)=5080412160
                Peak Reduce Physical memory (bytes)=203284480
                Peak Reduce Virtual memory (bytes)=8424714240
        Shuffle Errors
                BAD_ID=0
                CONNECTION=0
                IO_ERROR=0
                WRONG_LENGTH=0
                WRONG_MAP=0
                WRONG_REDUCE=0
        File Input Format Counters
                Bytes Read=10000668
        File Output Format Counters
                Bytes Written=1953898
```



```
root@90caf23382fc:/# hdfs dfs -get /output/* output
2020-03-13 05:35:37,861 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localHostTrusted = false, remoteHostTrusted = false
root@90caf23382fc:/# head -40 output/part-r-00000
"'Having        1
"'I     1
"*      1
"------------------------------A        1
"----------------------------The        1
"------------------------sorrowing      1
"--------------------Adoring,   1
"------------------And  1
"----------------The    1
"--------------Power    1
"--------------crowds   1
"----------With 1
"1.     2
"10.    1
"11.    1
"12.    1
"13.    1
"14.    1
"144."  1
"15,    1
"18.    1
"19.    1
"2.     2
"20.    1
"21.    1
"22,    1
"3.     1
"4.     1
"5.     1
"6.     1
"8.     1
"9.     1
"A      28
"About  4
"According      2
"Address        1
"Adored 1
"After  4
"Again, 1
"All    3
root@90caf23382fc:/# |
```

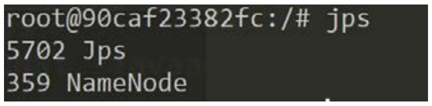
(e) Run WordCount on the Hadoop cluster with 2 VMs, what are the top 5 most frequent word in the provided txt file?



```
root@90caf23382fc:/# sort -n -k 2 -r output/part-r-00000 | head -5
the     94032
of      65837
|       55057
and     46376
in      27613
```

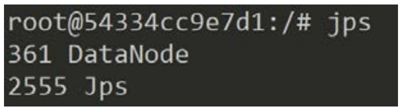
(f) Use jps commands on both VMs to show running Hadoop daemons and provide and screenshots.

For master:



```
root@90caf23382fc:/# jps
5702 Jps
359 NameNode
```

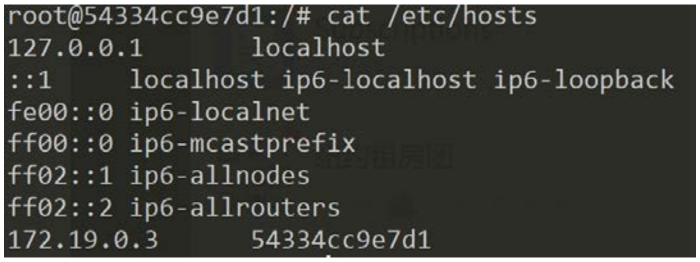
For slave:



```
root@54334cc9e7d1:/# jps
361 DataNode
2555 Jps
```

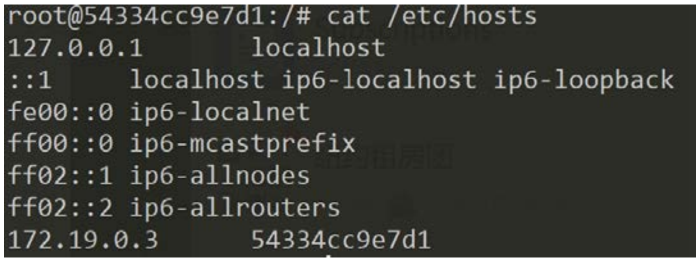
(g) Screenshots of configuration files and IP addresses for Master node and Slave node of your small cluster as well as the MapReduce execution result. For each configuration file, please also briefly explain what it does.

For master:



```
root@54334cc9e7d1:/# cat /etc/hosts
127.0.0.1       localhost
::1     localhost ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
172.19.0.3      54334cc9e7d1
```

For slave:

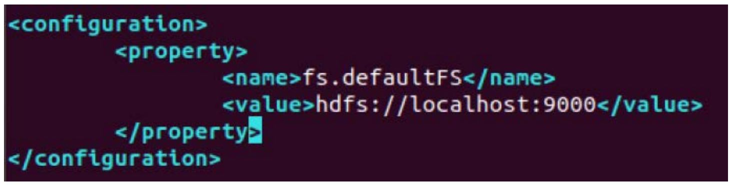


```
root@54334cc9e7d1:/# cat /etc/hosts
127.0.0.1       localhost
::1     localhost ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
172.19.0.3      54334cc9e7d1
```

Configuration files:

core-site.xml:

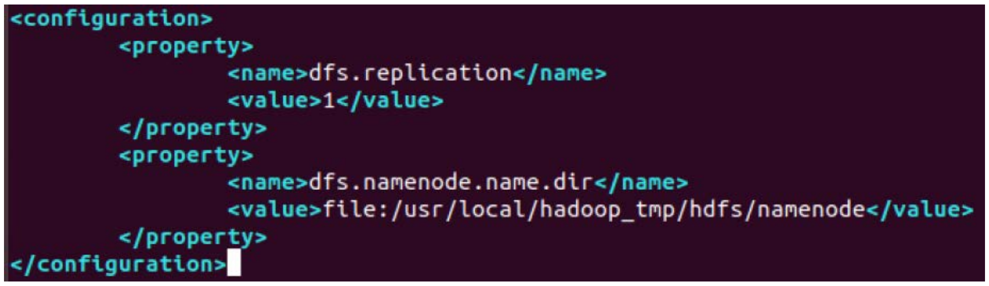
Configure the address and port number of the HDFS master.



```
<configuration>
        <property>
                <name>fs.defaultFS</name>
                <value>hdfs://localhost:9000</value>
        </property>
</configuration>
```

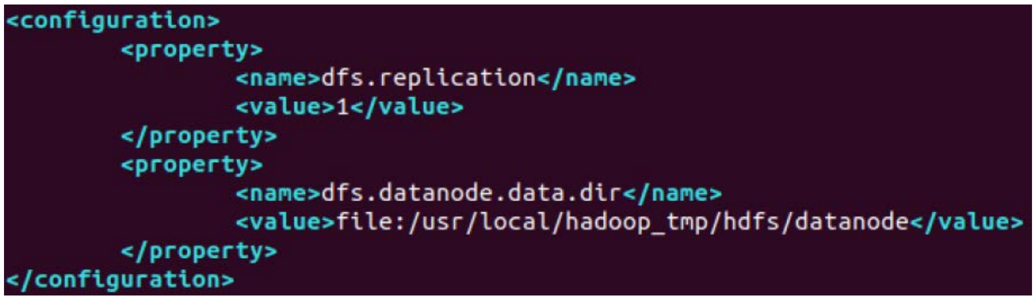
hdfs-site.xml(HadoopMaster):

Modify the configuration of HDFS in Hadoop, and the default backup mode of configuration is 3.



```
<configuration>
        <property>
                <name>dfs.replication</name>
                <value>1</value>
        </property>
        <property>
                <name>dfs.namenode.name.dir</name>
                <value>file:/usr/local/hadoop_tmp/hdfs/namenode</value>
        </property>
</configuration>
```

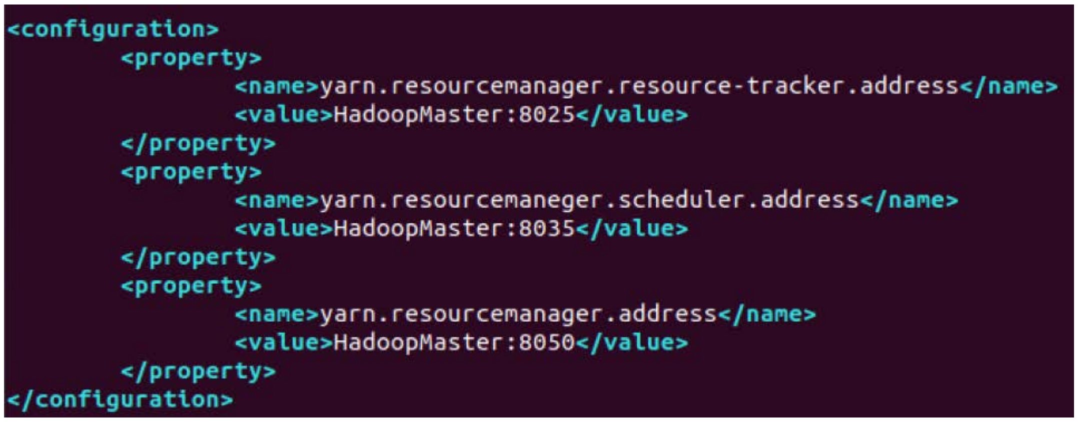
hdfs-site.xml(HadoopSlave):



```
<configuration>
        <property>
                <name>dfs.replication</name>
                <value>1</value>
        </property>
        <property>
                <name>dfs.datanode.data.dir</name>
                <value>file:/usr/local/hadoop_tmp/hdfs/datanode</value>
        </property>
</configuration>
```

yarn-site.xml

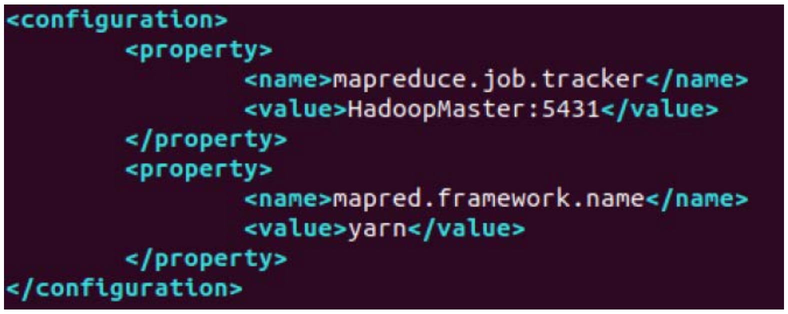
Update hostname from localhost to HadoopMaster.



```
<configuration>
        <property>
                <name>yarn.resourcemanager.resource-tracker.address</name>
                <value>HadoopMaster:8025</value>
        </property>
        <property>
                <name>yarn.resourcemaneger.scheduler.address</name>
                <value>HadoopMaster:8035</value>
        </property>
        <property>
                <name>yarn.resourcemanager.address</name>
                <value>HadoopMaster:8050</value>
        </property>
</configuration>
```

mapred-site.xml

Modify the MapReduce configuration file in Hadoop to configure the address and port of JobTracker.



```
<configuration>
        <property>
                <name>mapreduce.job.tracker</name>
                <value>HadoopMaster:5431</value>
        </property>
        <property>
                <name>mapred.framework.name</name>
                <value>yarn</value>
        </property>
</configuration>
```

MapReduce execution result：

```
"Come,  1
"Confined       1
"Continuing 1
"Could  1
"Damned 1
"Deeply 1
"Defects,"      3
"Desirous       1
"Dies   1
"Disturbing 1
"Do 1
"Dr.    1
"During 3
"E. 1
"Eat    1
"Elastic        1
"Elephants      1
"Elixir 2
"Ere    1
"Ever   1
"Every  3
"Examined       1
"Experience 1
"Experiments    1
"Fahrenheit's   1
"Farther        1
"Father 1
"First, 1
"Flitch 1
"Fly    1
"For    1
"From   5
"Further        2
"Give   1
```

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

Master Node – Master node in a hadoop cluster is responsible for storing data in HDFS and executing parallel computation the stored data using MapReduce. Master Node has 3 nodes – NameNode, Secondary NameNode and JobTracker. JobTracker monitors the parallel processing of data using MapReduce while the NameNode handles the data storage function with HDFS. NameNode keeps a track of all the information on files (i.e. the metadata on files) such as the access time of the file, which user is accessing a file on current time and which file is saved in which hadoop cluster. The secondary NameNode keeps a backup of the NameNode data.

Slave/Worker Node- This component in a hadoop cluster is responsible for storing the data and performing computations. Every slave/worker node runs both a TaskTracker and a DataNode service to communicate with the Master node in the cluster. The DataNode service is secondary to the NameNode and the TaskTracker service is secondary to the JobTracker.

NameNode works as Master in Hadoop cluster. Below listed are the main function performed by NameNode:

1. Stores metadata of actual data. E.g. Filename, Path, No. of Data Blocks, Block IDs, Block Location, No. of Replicas, Slave related configuration
2. Manages File system namespace.
3. Regulates client access request for actual file data file.
4. Assign work to Slaves(DataNode).
5. Executes file system name space operation like opening/closing files, renaming files and directories.
6. As Name node keep metadata in memory for fast retrieval, the huge amount of memory is required for its operation. This should be hosted on reliable hardware.

DataNode works as Slave in Hadoop cluster . Below listed are the main function performed by DataNode:

1. Actually stores Business data.
2. This is actual worker node were Read/Write/Data processing is handled.
3. Upon instruction from Master, it performs creation/replication/deletion of data blocks.
4. As all the Business data is stored on DataNode, the huge amount of storage is required for its operation. Commodity hardware can be used for hosting DataNode.

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

Suppose we have a pxq matrix M, whose element in row i and column j will be denoted m_ij and a qxr matrix N whose element in row j and column k is donated by n_jk then the product P = MN will be pxr matrix P whose element in row i and column k will be donated by p_ik, where P(i,k) = m_ij * n_jk.

The Map function:

```
for each element m_ij of M do:
        produce (key, value) pairs as ((i,k),(M,j,m_ij)), for k = 1,2,3, … up to the number of columns of N

for each element n_jk of N do:
        produce (key,value) pairs as ((i,k),(N,j,n_jk)), for i = 1,2,3, … up to the number of rows of M

return Set of (key,value) pairs that each key, (i,k), has a list with values (M,j,m_ij) and (N,j,n_jk) for all possible values of j
```

i.e.

input:

((M,1,1),m11)

((M,1,2),m12)

((M,1,3),m13)

...

((N,1,1),n11)

((N,1,2),n12)

((N,1,3),n13)

...

After map():

(M,i,j),x)---> ((i,1),x) ((i,2),x).... ((i,Q),x)

(N,i,j),x)---> ((1,j),x) ((2,j),x).... ((Q,j),x)

Where Q = # rows in matrix M (= # columns in matrix N)

The Reduce function:

```
for each key (i,k) do:
    sort values begin with M by j in listM
    sort values begin with N by j in listN
    multiply m_ij and n_jk for jth value of each list
    sum up m_ij * n_jk
return (i,k) $\sum_{j=1} m_{ij} * n_{jk}$
```

Pseudo code:

```
map(key,value):
    if value[0] == "M":
        i = value[1]
        j = value[2]
        m_ij = value[3]
        for k = 1 to r:
            emit((i,k),(M,j,m_ij))
    else:
        j = value[1]
        k = value[2]
        n_jk = value[3]
        for i = 1 to p:
            emit((i,k),(N,j,n_jk))
reduce(key,values):
    hash_M = {j:m_ij for(x,j,m_ij) in values if x == M}
    hash_N = {j:n_jk for(x,j,n_jk) in values if x == N}
    result = 0
    for j = 1 to q:
        result += hash_M[j] * hash_N[j]
    emit(key,result)
```

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

A Combiner, also known as a semi-reducer, is an optional class that operates by accepting the inputs from the Map class and thereafter passing the output key-value pairs to the Reducer class. The main function of a Combiner is to summarize the map output records with the same key.

Add a combiner after the map function and before the reduce function. The main function of a Combiner is to summarize the map output records with the same key, which can shuffle on the first key of the map( ) output and generate the inputs to reducers.