Cloud Networking and Computing
Lab 1: HDFS and Benchmarking Map-Reduce

Andrzej Duda

Preliminaries

When you need to recall the basic facts on Hadoop clusters or better understand what happens during the Map-Reduce execution, use this support:

Brad Hedlund. *Understanding Hadoop Clusters and the Network*,
Map-Reduce: *Introduction to Map-Reduce.*

1 Getting started

You need to connect first to the following server: `ssh liku@delos.imag.fr`. On `delos.imag.fr`, you will connect to the cluster deployed for this Lab with a single user name `duda`. The connection is to the master node "NameNode", which is also a traditional Linux system.

Please do not run any program or perform any harmful operation that may jeopardize basic security rules on these servers!

So, the steps to follow:

☞ **1** – Open a shell on delos with `ssh liku@delos.imag.fr`.

☞ **2** – Open a shell on the master node with `ssh duda@152.77.78.100`.

☞ **3** – Execute `hdfs dfs` to list available commands.

☞ **4** – Execute `hdfs dfs -ls /` to view the contents of the root of HDFS.

☞ **5** – Execute `hdfs dfs -ls` to view the contents of a specific HDFS directory.

☞ **6** – Explore the content of HDFS on the cluster.

Question: (1 point) What is the HDFS path to your account directory?
Answer:

```
/user/duda
```

■

☞ **7** – Look at the HDFS configuration file:
`/usr/local/hadoop/share/doc/hadoop/hadoop-project-dist/hadoop-hdfs/hdfs-default.xml`

Question: (1 point) What is the HDFS default block size?
Answer:

```
134217728 for 128 MB
```

■

Question: (1 point) Find the available disk space.
Answer:

```
duda@NameNode:~ hdfs dfs -df
Filesystem                      Size          Used      Available  Use%
hdfs://NameNode:9000  356324343808  104847302804  232153948160    29%
```

■

Question: (1 point) Find the replication factor of the HDFS file system on the cluster.
Answer:

```
duda@NameNode:~$ hdfs fsck /user/duda
Connecting to namenode via http://NameNode:50070
FSCK started by duda (auth:SIMPLE) from /152.77.78.100 for path /user/duda at
Sun Apr 10 11:12:59 CEST 2016
.......................................Status: HEALTHY
 Total size: 17777217961 B
 Total dirs: 7
 Total files: 39
 Total symlinks: 0
 Total blocks (validated): 154 (avg. block size 115436480 B)
 Minimally replicated blocks: 154 (100.0 %)
 Over-replicated blocks: 0 (0.0 %)
 Under-replicated blocks: 0 (0.0 %)
 Mis-replicated blocks: 0 (0.0 %)
 Default replication factor: 2
 Average block replication: 2.0
 Corrupt blocks: 0
 Missing replicas: 0 (0.0 %)
 Number of data-nodes: 4
 Number of racks: 1
FSCK ended at Sun Apr 10 11:12:59 CEST 2016 in 9 milliseconds

The filesystem under path '/user/duda' is HEALTHY

Default replication factor: 2
```

■

☞

8 – Look at the site HDFS configuration file:
`/usr/local/hadoop/etc/hadoop/hdfs-site.xml`

Question: (1 point) Why the replication factor is not 3 as specified in the default configuration file?
Answer:

```
hdfs-site.xml overrides the default configuration file:

<property>
   <name>dfs.replication</name>
   <value>2</value>
 </property>
```

■

☞

9 – Look at `/etc/hosts` file.

Question: **(1 point)** What are the IP addresses of datanodes? Find which ones are operational.
Answer:

```
152.77.78.101    Worker01
152.77.78.102    Worker02
152.77.78.103    Worker03
152.77.78.104    Worker04
152.77.78.105    Worker05
152.77.78.106    Worker06
152.77.78.107    Worker07
152.77.78.108    Worker08
152.77.78.109    Worker09

152.77.78.100    NameNode# NIC <eth0>

Only 4 are operational:

duda@NameNode:~$ ping 152.77.78.101
PING 152.77.78.101 (152.77.78.101) 56(84) bytes of data.
64 bytes from 152.77.78.101: icmp_req=1 ttl=64 time=0.204 ms
^C
--- 152.77.78.101 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.204/0.204/0.204/0.000 ms
duda@NameNode:~$ ping 152.77.78.104
PING 152.77.78.104 (152.77.78.104) 56(84) bytes of data.
64 bytes from 152.77.78.104: icmp_req=1 ttl=64 time=0.237 ms
^C
--- 152.77.78.104 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.237/0.237/0.237/0.000 ms
duda@NameNode:~$ ping 152.77.78.105
PING 152.77.78.105 (152.77.78.105) 56(84) bytes of data.
^C
--- 152.77.78.105 ping statistics ---
2 packets transmitted, 0 received, 100% packet loss, time 1008ms
```

■

☞

10 – Look at `/usr/local/hadoop/etc/hadoop/slaves` file.

Question: **(1 point)** How many datanodes are defined?
Answer:

```
Worker01
Worker02
Worker03
Worker04
```

■

☞ **11** – Use this command to find how many CPUs are there on a node:

```
nproc
```

Question: (1 point) How many CPUs are there on a node? How many CPUs are there on the cluster?
Answer:

```
2

8
```

■

2 Run TestDFSIO benchmark on the cluster

Hadoop includes an HDFS benchmark application called TestDFSIO. It is a read and write test for HDFS: it writes into or reads from a specified number of files. The file size is specified as a parameter to the test. Each file is accessed in a separate map task.

The reducer collects the following statistics:

- number of tasks completed,
- number of bytes written/read,
- execution time,
- io rate,
- io rate squared.

To run this test, we will use MapReduce 2.0 (MRv2) or YARN. The fundamental idea of MRv2 is to split up the two major functionalities of the JobTracker, resource management and job scheduling/monitoring, into separate daemons. The idea is to have a global ResourceManager (RM) and per-application ApplicationMaster (AM). An application is either a single job in the classical sense of Map-Reduce jobs or a DAG of jobs.

☞ **12** – Check whether there are MapReduce jobs running. You may need to run benchmarks without any other jobs running.

```
mapred job -list
```

Question: (1 point) What information do you obtain?
Answer:

```
duda@NameNode:~$ mapred job -list
16/04/13 06:16:41 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platfo
16/04/13 06:16:41 INFO client.RMProxy: Connecting to ResourceManager at NameNode/152.77.78.100:803
Total jobs:0
                  JobId      State      StartTime     UserName        Queue   Priority  UsedContai
RsvdContainers  UsedMem  RsvdMem  NeededMem    AM info
```

No other jobs. ■

☞ **13** – Run TestDFSIO in write mode and create data—it generates 16 output files of size 1GB for a total of 16GB:

```
yarn jar /usr/local/hadoop/share/hadoop/mapreduce/
hadoop-mapreduce-client-jobclient-2.2.0.jar TestDFSIO
-Dtest.build.data=/user/duda/hdfs/ -write -nrFiles 16 -fileSize 1000
```

☞ **14** – When your application is running (it would take several minutes), open another ssh connection to 152.77.78.100 and run:
`yarn application -list`
to discover all applications running and use the application id to run:
`yarn application -status <your_app_id>`

Question: (1 point) What information do you obtain?
Answer:

```
duda@NameNode:~$ yarn application -status application_1444991602904_0682
16/04/10 12:40:16 INFO client.RMProxy: Connecting to ResourceManager
at NameNode/152.77.78.100:8032
Application Report :
Application-Id : application_1444991602904_0682
Application-Name : hadoop-mapreduce-client-jobclient-2.2.0-tests.jar
Application-Type : MAPREDUCE
User : duda
Queue : default
Start-Time : 1460284590604
Finish-Time : 0
Progress : 32.19%
State : RUNNING
Final-State : UNDEFINED
Tracking-URL : http://Worker03:37166
RPC Port : 57440
AM Host : Worker03
Diagnostics :
```

■

☞ **15** – Use the command `yarn node` to discover all running nodes.

Question: (1 point) How many workers are running?
Answer:

```
duda@NameNode:~$ yarn node -all -list
16/04/10 12:41:22 INFO client.RMProxy: Connecting to ResourceManager
at NameNode/152.77.78.100:8032
Total Nodes:4
         Node-Id      Node-State Node-Http-Address Number-of-Running-Containers
  Worker03:48450         RUNNING     Worker03:8042                            2
  Worker04:40820         RUNNING     Worker04:8042                            2
  Worker01:50843         RUNNING     Worker01:8042                            2
  Worker02:50121         RUNNING     Worker02:8042                            2
```

■

When the TestDFSIO application is done, answer the following questions:

Question: (1 point) What is the throughput in MB/s? How it is defined?
Answer:

```
16/04/10 09:25:43 INFO fs.TestDFSIO:      Throughput mb/sec: 7.049538872038533
```

$$Throughput(N) = \frac{\sum_{i=0}^{N} filesize_i}{\sum_{i=0}^{N} time_i},$$

where $filesize_i$ is the file size written (or read) by the individual map task and $time_i$ is the elapsed time to do so. ■

Question: (1 point) What is the Average IO rate in MB/s? How it is defined?
Answer:

```
16/04/10 09:25:43 INFO fs.TestDFSIO: Average IO rate mb/sec: 8.03139591217041
```

$$AverageIOrate(N) = \frac{\sum_{i=0}^{N} rate_i}{N} = \frac{\sum_{i=0}^{N} \frac{filesize_i}{time_i}}{N}$$

where $filesize_i$ is the file size written (or read) by the individual map task and $time_i$ is the elapsed time to do so. ■

☞ **16** – Run TestDFSIO in read mode:

```
yarn jar /usr/local/hadoop/share/hadoop/mapreduce/
hadoop-mapreduce-client-jobclient-2.2.0.jar TestDFSIO
-Dtest.build.data=/user/duda/hdfs/ -read -nrFiles 16 -fileSize 1000
```

Question: (1 point) What is the throughput in MB/s?
Answer:

```
16/04/10 09:30:38 INFO fs.TestDFSIO:      Throughput mb/sec: 14.58819796549344
```

■

Question: (1 point) What is the average IO rate in MB/s?
Answer:

```
16/04/10 09:30:38 INFO fs.TestDFSIO: Average IO rate mb/sec: 15.580423355102539
```

■

☞ **17** – Rerun TestDFSIO in write mode for replication factor = 1:

```
yarn jar /usr/local/hadoop/share/hadoop/mapreduce/
hadoop-mapreduce-client-jobclient-2.2.0.jar TestDFSIO
-Dtest.build.data=/user/duda/hdfs/ -D dfs.replication=1
-write -nrFiles 16 -fileSize 1000
```

Question: (1 point) What is the throughput now? Why?
Answer:

```
16/04/10 12:12:57 INFO fs.TestDFSIO:      Throughput mb/sec: 15.778797044236844
```

■

Question: (1 point) What is the average IO rate now? Why?
Answer:

```
16/04/10 12:12:57 INFO fs.TestDFSIO: Average IO rate mb/sec: 19.16858673095703
```

The preformance indices are roughly 2 times better because instead of writing 2 blocks, HDFS writes only 1 block. ■

☞ **18** – Clean the file system with: `yarn jar /usr/local/hadoop/share/hadoop/mapreduce/ hadoop-mapreduce-client-jobclient-2.2.0.jar TestDFSIO -Dtest.build.data=/user/duda/ -clean`

3 Run TeraSort benchmark on the cluster

The TeraSort benchmark sorts 1TB of data (or any other amount of data you want) as fast as possible. It is a benchmark that combines testing the HDFS and MapReduce layers of an Hadoop cluster. As such, it is not surprising that the TeraSort benchmark suite is often used in practice, which has the added benefit that it allows us – among other things – to compare the results of our own cluster with the clusters of other people.

You can use the TeraSort benchmark, for instance, to iron out your Hadoop configuration after your cluster passed a convincing TestDFSIO benchmark first. Typical areas where TeraSort is helpful is to determine whether your map and reduce slot assignments are sound (as they depend on the variables such as the number of cores per TaskTracker node and the available RAM), whether other MapReduce-related parameters are set to proper values, or whether the FairScheduler configuration you came up with really behaves as expected.

The TeraSort benchmark run consists of the following three steps:

1. Generating the input data via TeraGen.
2. Running the actual TeraSort on the input data.
3. Validating the sorted output data via TeraValidate.

TeraGen generates random data that can be conveniently used as input data for a subsequent TeraSort run. The first parameter specifies the number of rows of the input data to generate, each of which having the size of 100 bytes.

The actual TeraGen data format per row is:

```
<10 bytes key><10 bytes row_id><78 bytes filler>\r\n
```

☞

19 – Run TeraGen to generate a 1G file:

```
time yarn jar /usr/local/hadoop/share/hadoop/mapreduce/
hadoop-mapreduce-examples-2.2.0.jar teragen
10000000 /user/duda/hdfs/TeraGen-1GB
```

Question: **(1 point)** How much time did the job take? How many tasks were involved?
Answer:

```
real 0m43.897s

Job Counters
Launched map tasks=2
Other local map tasks=2
Total time spent by all maps in occupied slots (ms)=431056
Total time spent by all reduces in occupied slots (ms)=0
```

■

TeraSort is implemented as a standard map/reduce sort, except for a custom partitioner that uses a sorted list of $N - 1$ sampled keys that define the key range for each reduce. In particular, all keys such that $sample[i-1] <= key < sample[i]$ are sent to reduce i. This guarantees that the output of reduce i are all less than the output of reduce $i+1$. To speed up the partitioning, the partitioner builds a two level *trie* that quickly indexes into the list of sample keys based on the first two bytes of the key. TeraSort generates the sample keys by sampling the input before the job is submitted and writing the list of keys into HDFS. The output of the reduce has replication set to 1, instead of the default 3 (or 2 in our cluster), because the contest does not require the output data be replicated onto multiple nodes.

- TeraSort is essentially **distributed sort (DS)**

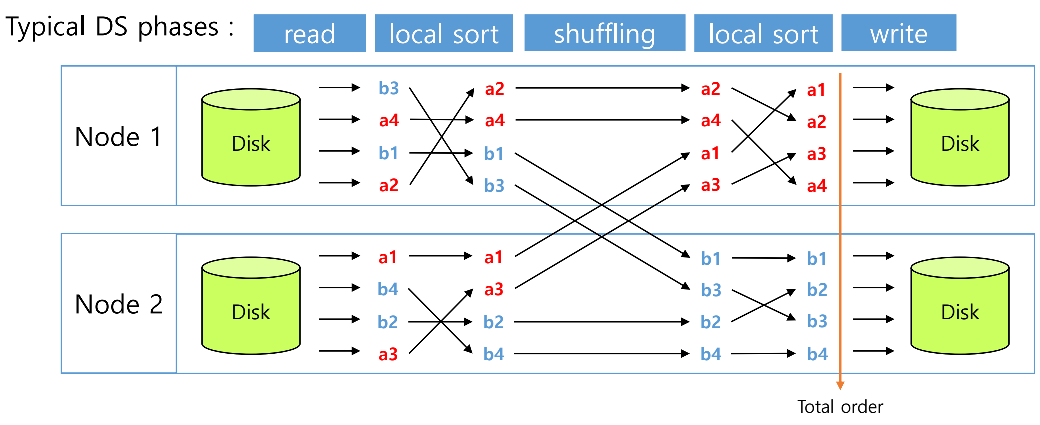

Figure 1: Distributed sort in TeraSort.

Yahoo! achieved 62 seconds to sort 1 TB of data on a cluster of 3800 nodes.

☞ **20** – Run TeraSort on the generated file:

```
time yarn jar /usr/local/hadoop/share/hadoop/mapreduce/
hadoop-mapreduce-examples-2.2.0.jar terasort
/user/duda/hdfs/TeraGen-1GB /user/duda/hdfs/TeraSort-1GB
```

Question: (1 point) How much time did the job take? How many tasks were involved?
Answer:

```
real 1m30.911s

Job Counters
Killed map tasks=1
Launched map tasks=9
Launched reduce tasks=1
Data-local map tasks=8
Rack-local map tasks=1
Total time spent by all maps in occupied slots (ms)=1458168
Total time spent by all reduces in occupied slots (ms)=431000
```

☞ **21** – Run TeraSort with an increased number of Reduce tasks:

```
time yarn jar /usr/local/hadoop/share/hadoop/mapreduce/
hadoop-mapreduce-examples-2.2.0.jar terasort -Dmapred.reduce.tasks=4
/user/duda/hdfs/TeraGen-1GB /user/duda/hdfs/TeraSort-1GB
```

Question: (1 point) How many tasks were involved? How much time did the job take? Why?
Answer:

```
real 1m18.578s

Job Counters
Launched map tasks=8
Launched reduce tasks=4
Data-local map tasks=8
Total time spent by all maps in occupied slots (ms)=1503064
Total time spent by all reduces in occupied slots (ms)=868160
```

A good starting point for the number of reduce tasks is the number of CPU cores in the cluster divided by 2. This will allow all reduce tasks to begin shuffling data across the network when the last wave of map tasks begin processing.

In our case, it is 4.

TeraSort is:

CPU bound for the Map stage

I/O bound for the Reduce stage

As there were more Reduce tasks, the time is shorter than previously, because 4 Reduce tasks can work in parallel.

☞ **22** – Clean the file system with: `hdfs dfs -rm -r -skipTrash /user/duda/Tera*`