TP3: Configuring and optimizing *Hadoop*

Shadi Ibrahim
(March 2nd, 2017)

MapReduce has emerged as a leading programming model for data-intensive computing. It was originally proposed by Google to simplify development of web search applications on a large number of machines.

Hadoop is a java open source implementation of MapReduce sponsored by Yahoo! The Hadoop project is a collection of various subprojects for reliable, scalable distributed computing. The two fundamental subprojects are the Hadoop MapReduce framework and the HDFS. HDFS is a distributed file system that provides high throughput access to application data. It is inspired by the GFS. HDFS has master/slave architecture. The master server, called NameNode, splits files into blocks and distributes them across the cluster with replications for fault tolerance. It holds all metadata information about stored files. The HDFS slaves, the actual store of the data blocks called DataNodes, serve read/write requests from clients and propagate replication tasks as directed by the NameNode.

The Hadoop MapReduce is a software framework for distributed processing of large data sets on compute clusters. It runs on the top of the HDFS. Thus data processing is collocated with data storage. It also has master/slave architecture. The master, called Job Tracker (JT), is responsible of : (a) Querying the NameNode for the block locations, (b) considering the information retrieved by the NameNode, JT schedule the tasks on the slaves, called Task Trackers (TT), and (c) monitoring the success and failures of the tasks.

The goal of this TP is to study how to configure the platform to expose different important features of Hadoop (e.g., data replication, speculation, etc). Then, we shall examine how this tuning impacts on the overall performance (e.g., execution time). We suppose you have an instance of Hadoop running on your local machine, as seen in the previous TP.

Exercise 1: Configuring your Hadoop platform

Question 1.1

Setting replication factor to 2 and 3. Replication is Hadoop is essential to achieve (1) fault-tolerance and (2) data locality when executing map tasks. Which file in the conf folder we should modify and how? Please set the replication factor to 3.

Use data set of 10GB and run wordcount benchmark using different replication factor (i.e., 1 ,2 and 3). Use the default block size (128MB)
See the execution times and the achieved data locality, what can you observe?

Question 1.2

Use data set of 10GB and run wordcount benchmark using different block size (i.e., 10MB, 64MB, 128MB, and 512MB). Use the default replication factor (3)
See the execution times and the achieved data locality, what can you observe?

Question 1.3

MapReduce masks slow nodes by launching a speculative copy of its task on another machine. To enable this feature (here for map task), in `conf/mapred-site.xml` add:

```
<configuration>
  <property>
    <name>mapred.map.tasks.speculative.execution</name>
      <value>true</value>
    </property>
</configuration>
```

To disable this feature set the `value` to `false`. Try to enable speculation for both map and reduce tasks.

Use data set of 10GB and run wordcount benchmark using block size (i.e., 128MB) and the default replication factor (3) with speculation enabled and disabled (try to log-in one node and write a program to overload the CPU or the disk).
See the execution times, the number of speculative copies, and average (maximum) task runtimes for both map and reduce tasks, what can you observe?

Use data set of 10GB and run sort (to generate the input data, please refer to the course site) benchmark using block size (i.e., 128MB) and the default replication factor (3) with speculation enabled and disabled (try to log-in one node and write a program to overload the CPU or the disk).
See the execution times, the number of speculative copies, and average (maximum) task

runtimes for both map and reduce tasks, what can you observe?

Question 1.4

Each TaskTracker sends a heartbeat to the JobTracker every 3 seconds to say that they are alive. However if the TaskTracher didn't receive any heartbeat for certain time (by default 10 minutes), the TaskTracker will be declared as failed node. This value can be changed in the `conf/mapred-site.xml`.

```
<configuration>
   <property>
      <name>mapred.tasktracker.expiry.interval</name>
         <value>600000</value>
      </property>
</configuration>
```

Use data set of 10GB and run sort benchmark using block size (i.e., 128MB) and the default replication factor (3). For each run, set different expiry interval (30seconds, 1 minutes and 5 minutes). Try to stop the tasktracker daemon on one node (before the completion of the map tasks and after the completion of map tasks).
See the execution times, the number of killed tasks, what can you observe?

Question 1.5

Try to stop the datanode daemon of one node and run randomwriter to generate 10GB of data. Re-start Hadoop and use *balancer* command to rebalance the cluster.

Exercise 2: Hadoop optimization

Question 2.1

By default, each Hadoop node runs 2 mapper tasks and 2 reducer tasks concurrently. To change the maximum number of map/reduce tasks that will be run simultaneously by a TaskTracker (i.e., the number of available map and reduce slots per node), reset the value in `conf/mapred-site.xml`:

```
<configuration>
  <property>
    <name>mapred.tasktracker.map.tasks.maximum</name>
      <value>X</value>
    </property>
</configuration>
```

and

```
<configuration>
  <property>
    <name>mapred.tasktracker.reduce.tasks.maximum</name>
      <value>X</value>
    </property>
</configuration>
```

Don't forget to re-start Hadoop! Set the Block size to 128MB, and use the Data set 10GB. Now, run the two aforementioned examples (wordcount, Pi estimator, sort) using different map slots number(i.e., 1, 4, 16).
See the execution times, what can you observe?

Question 2.2

JVM reuse is a Hadoop tuning parameter that is very relevant for performance, especially in scenarios where it's hard to avoid small files and scenarios with lots of tasks, most which have short execution times. The default configuration of Hadoop will typically launch map or reduce tasks in a forked JVM; that is: one JVM per map/reduce task. The JVM start-up may create significant overhead, especially when launching jobs with hundreds or thousands of tasks.
Instead, reuse allows a JVM instance to be reused up to N times for the same job. If you have very small tasks that are definitely running after each other, it is useful to set this property to -1 (meaning that a spawned JVM will be reused unlimited times). In long running jobs the percentage of the runtime in comparison to setup a new JVM is very low, so it doesn't give you a huge performance boost. This value is set in Hadoop's `mapred-site.xml`:

```
<configuration>
  <property>
    <name>mapred.job.reuse.jvm.num.tasks</name>
      <value>1</value>
    </property>
</configuration>
```

Set the Block size to 128MB, and use the Data set 5GB. Now, run the two aforementioned examples (wordcount, Pi estimator, sort) with 8 map slots and change the number of tasks per jvm to (i.e., 1, 2, and 4).
See the execution times, what can you observe?

Question 2.3

To set the fraction of the number of maps in the job which should be complete before reduces are scheduled for the job.

```
<configuration>
  <property>
    <name>mapred.reduce.slowstart.completed.maps</name>
      <value>0.05</value>
    </property>
</configuration>
```

Use the Data set 10GB and run wordcount and sort benchmarks using different values (i.e., 0.05, 0.5 and 1). See the execution time, what can you observe?