Report

Jian Zhu
A20362556

1. Description

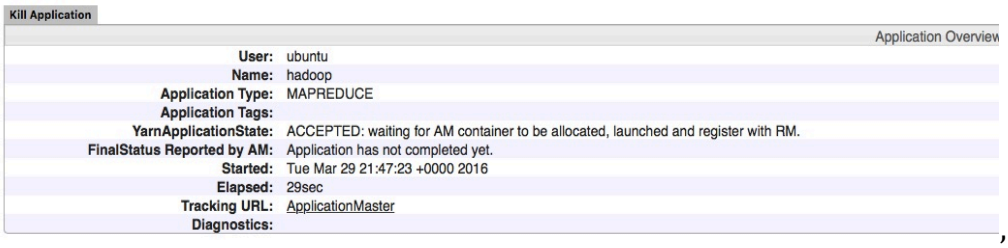
1. Problem
   a) Share-memory
      We need to implement a sort application to sort datasets that are larger than memory, use multiple threads to process large files concurrently and measure the execution time and throughout.
   b) Hadoop
      Use hadoop in HDFS distributed file system to sort different sizes of datasets on 1 node and 16 nodes, then measure the execution time and throughout.
   c) Spark
      Use Spark in HDFS distributed file system to sort different sizes of datasets on 1 node and 16 nodes, then measure the execution time and throughout.

2. Runtime environment setting
   a) Share-memory
      I develop my code on Mac System with Eclipse, 1.7 JAVA version.
      When executing my source code, we need to compile it on Eclipse environment then get the class file. I set 5 parameters when executing the code. In linux system, we run the program like: 'java, Terasort, datasize, size of buffer, datadirectory, number of buffer, thread number'. For example, 'java Terasort 10731520 16384 /desktop 3000 2'.

   b) Hadoop&Spark
      For each instance, I set up raid to guarantee enough disk space.
      sudo umount /mnt
      sudo mdadm --create --verbose /dev/md0 --level=0 --name=my_raid --raid-devices=2 /dev/xvda /dev/xvdf
      sudo umount /home/ubuntu/hadoop_dir
      sudo mkfs.ext4 -L my_raid /dev/md0
      sudo mount LABEL=my_raid ~/hadoop_dir
      sudo chown -R ubuntu ~/hadoop_dir
      sudo chgrp -R ubuntu ~/hadoop_dir

3. Installation step
   Install 1 instance c3.large, use scp method to upload the hadoop and spark configuration file to this instance, run on Ubuntu system. Then change configuration files one by one. For 16 nodes, at start I create image AMI using existed master node in order to decrease the work of modifying configuration file.
4. Difficulty

When I first create c3.large, the existed disk space is enough to execute the 1G file, so I did not attach extra volume. However, the reality is that when the program is executing, the condition that 'disk is out of space' always happen. Also, it took me a lot of time to modify and delete the data in 16 slaves. The biggest problem I meet is that the yarn application always keeps the status:



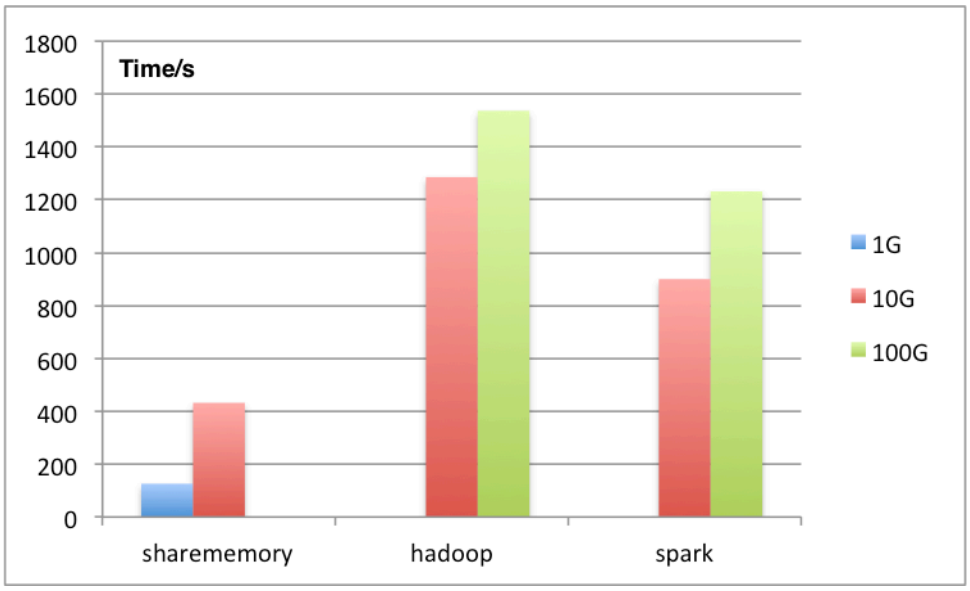
,

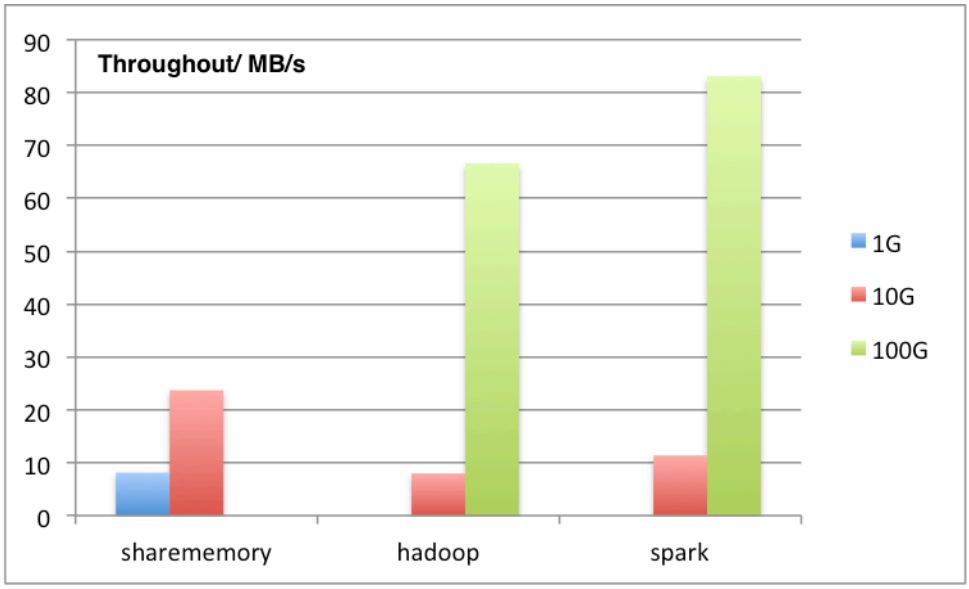
This never happened when I run on 1 node so that I did not notice my yarn-site.xml has error, which took me long time to check.

5. Version
   a) Java-SE 1.7
   b) Hadoop 2.7.2
   c) Spark 1.6.1

2. Data Analysis &Chart

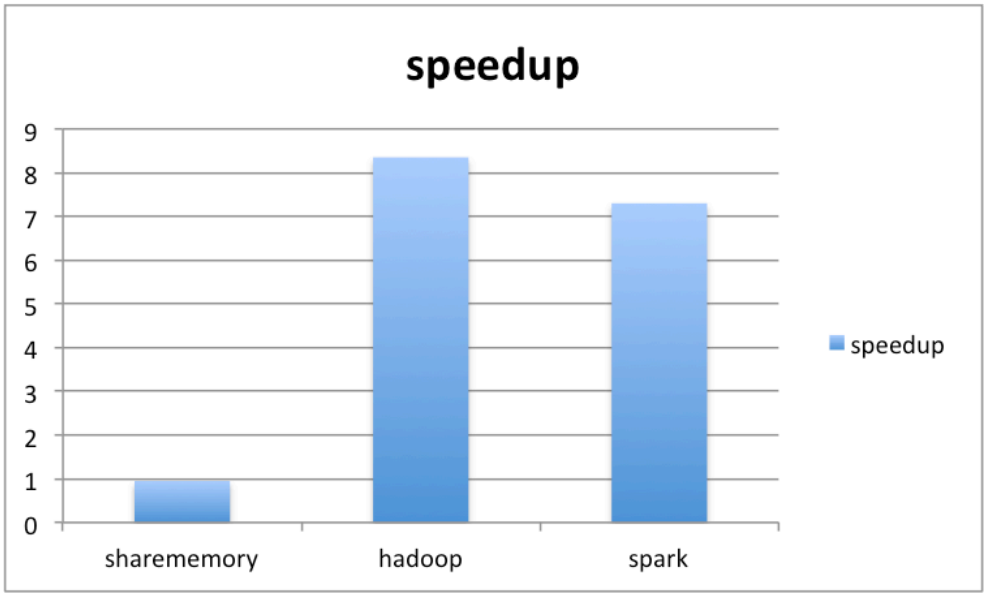
1. Execution time chart





2. Speed-up chart

Share-Memory:
1G: 126s 8.13MB/s
10G: 432s 23.70MB/s
Hadoop:
10G: 1286s 7.97MB/s
100G: 1537s 66.62MB/s
Spark:
10G: 900s 11.38MB/s
100G: 1232s 83.12MB/s



According to the chart,

1. at the same dataset 10G or 100G, spark spent less time to finish, we can know that Spark is faster than hadoop and has better throughout than hadoop.
2. Comparing to run on 1 node, running on 16 nodes will not increase the speed to 16 times.
3. I predict that spark will perform best on 100 or 1000nodes.
4. In one node scale, Share-memory is the best. In 16 nodes scale, spark is the best. Also, even on 100 nodes, the speed will not increase to 100 times because of speed limitation.
5. Compared 1 node to 16 nodes, the hadoop has best speedup.

3. Question-Answer

1. conf/master: Contain DNC address of master.
   conf/slaves: Contain all the DNC address of slaves
   conf/core-site.xml: Contain the configuration files: Hdfs,Yarn and MapReduce
   conf/hdfs-site.xml: Contain hdfs and how many replicates
   conf/mapred-site.xml: set the maximum map and reduce task number here.

2. Question:
   a) What is a master node and what is a slaves node?
      i. Master node is to manager resource of slaves and control the slaves to compute and store
      ii. Slaves node is to compute and store data.
   b) Because every configuration file needs to have a port, if we use only one port, it will cause the conflicts.
   c) In the mapred-site.xml, we can modify the this configuration file through change mapred.tasktracker.map.tasks.maximum and mapred.tasktracker.reduce.tasks.maximum .
3. Cloud-sort:

   Through this, we can get the minimum cost for sorting a fixed number if records on any public cloud. This benchmark will not only point out the best platforms for building data pumps, but also find the most efficient sort implementations from a total-cost-of-ownership perspective.

   Therefore, in the future, we may use this to improve the efficiency of work and save money.