**Cluster Benchmark Document**

As a Hadoop cluster is brought online and optimized for peak performance, it is important to periodically benchmark the performance of the cluster to ensure any changes made were beneficial to the performance of the cluster.

To assist in this task, we'll use the Ganglia distributed system monitoring software. We'll use the one node we installed the GUI to be our main monitoring node. To easily install the software we need, it is necessary to add the RPMForge repository:

- `wget .` http://apt.sw.be/redhat/el6/en/x86_64/rpmforge/RPMS/rpmforge-release-0.5.3-1.el6.rf.x86_64.rpm
- `rpm -ivh` rpmforge-release-0.5.3-1.el6.rf.x86_64.rpm

Now that we have access to the RPMForge repository, lets install the necessary packages on the main monitoring node:

- `sudo yum install rrdtool ganglia ganglia-metad ganglia-gmond ganglia-web httpd`
- `sudo service gmond start`
- `sudo service gmetad start`
- `sudo service httpd start`

If everything went successfully, a Ganglia monitor should be visible by going to http://localhost/ganglia.

To get Ganglia up and running on our other nodes, SCP the RPMForge rpm to those machines and install it as we did above. Then:

- `sudo yum install ganglia ganglia-gmond`
- `sudo service gmond start`

By default, Ganglia will monitor processor load and memory, which is all that we really need at the moment.

The most straightforward initial test for our cluster is to measure the performance of the HDFS filesystem on reads and writes. To accomplish this, we'll use the TestDFSIO program that comes bundled with Hadoop (/usr/lib/hadoop-mapreduce/hadoop-mapreduce-*test*.jar).

- For these initial read/write tests, all meaningful work is done during the map stage and thus this stage is the only important one. A small subsequent reduce step merely aggregates the results from the various map tasks.
- The number of maps that will launch corresponds to the number of files we want to write/read (indicated by the '-nrFiles' flag).
- `time hadoop jar /usr/lib/hadoop-mapreduce/hadoop-mapreduce-*test*.jar TestDFSIO -write -nrFiles 10 -fileSize 1000`
- `time hadoop jar /usr/lib/hadoop-mapreduce/hadoop-mapreduce-*test*.jar TestDFSIO -read -nrFiles 10 -fileSize 1000`

- `hadoop jar /usr/lib/hadoop-mapreduce/hadoop-mapreduce-*test*.jar TestDFSIO -clean`

The next logical step is testing the performance of MapReduce on YARN. To do this, we'll generate one gigabyte of random data and then sort it. To generate the random data, we'll use the randomwriter utility found in /usr/lib/hadoop-mapreduce/hadoop-mapreduce-examples-2.0.6-alpha.jar.

- To set the number of maps to be used, pass argument for test.randomwriter.maps_per_host:
  - `hadoop jar /usr/lib/hadoop-mapreduce/hadoop-mapreduce-examples-2.0.6-alpha.jar randomwriter -Dtest.randomwriter.maps_per_host=10 randomdata`

Now to sort the data we just generated we can use another bundled sort program found in the examples JAR used above. The number of reducers by default is 1.0 x configured capacity and may run faster with more.

- `time hadoop jar /usr/lib/hadoop-mapreduce/hadoop-mapreduce-examples-2.0.6-alpha.jar sort randomdata sorteddata`