**Engineering Excellence J1**

**DATA LAKE** **BENCHMARK TESTING**

[PREREQUISITES 3](#_Toc458010978)

[CONFIGURATIONS 3](#_Toc458010979)

[HDFS TESTS 4](#_Toc458010980)

[1. TestDFSIO 4](#_Toc458010981)

[2. TERASORT BENCHMARK SUITE 6](#_Toc458010982)

[3. NAMENODE BENCHMARK (NNBENCH) 7](#_Toc458010983)

[4. MAP REDUCE TESTS 11](#_Toc458010984)

[ MapReduce Benchmark (MRBench) 11](#_Toc458010985)

[CONCLUSION 12](#_Toc458010986)

**Overview**

This document provides overview of one of the Hadoop cluster evaluation process.

Hadoop is a framework which enables applications to work on large data on clusters which can have n numbers of nodes built of commodity hardware. It provides a distributed file system (HDFS) that stores data on the computed nodes, providing very high aggregate bandwidth across the cluster. In addition, Hadoop implements a parallel computational paradigm named MapReduce which divides the application into many small fragments of work, each of which may be executed or re executed on any node in the cluster.

The benchmarking of the cluster which means assessing its performances by running a variety of jobs each focused on a specific field (indexing, querying, predictive statistics, machine learning …).

This benchmark measures response on a handful of relational queries: scans, aggregations, joins, and UDF's, across different data sizes. Keep in mind that these systems have very different sets of capabilities. MapReduce-like systems (Shark/Hive) target flexible and large-scale computation, supporting complex User Defined Functions (UDF's), tolerating failures, and scaling to thousands of nodes. Traditional MPP databases are strictly SQL compliant and heavily optimized for relational queries. The workload here is simply one set of queries that most of these systems these can complete.

The Hadoop distribution comes with a number of benchmarks, which are bundled in hadoop-\*test\*.jar and hadoop-\*examples\*.jar. The four benchmarks we will be looking at in more details are TestDFSIO, nnbench, mrbench (in hadoop-\*test\*.jar) and TeraGen / TeraSort / TeraValidate (in hadoop-\*examples\*.jar).

## 

## PREREQUISITES

Run the following command before executing benchmark tests.

1. Create directory /benchmarks in hdfs

sudo -u hdfs hdfs dfs -mkdir /benchmarks

1. Provide ownership on newly created directory to user who is going to execute benchmark tests.

sudo -u hdfs hdfs dfs -chown -R hdfs:hdfs /benchmarks​

## 

## CONFIGURATIONS

|  |  |  |  |
| --- | --- | --- | --- |
| **Group** | **Property** | **Property Name** | **Property Value** |
| HDFS Block size |  | dfs.block.size | 134 MB |
| Java Heap Size of NameNode |  |  | 1GB |
| Java Heap Size of DataNode |  |  | 1GB |
| ResourceManager | Container Memory Minimum | yarn.scheduler.minimum-allocation-mb | 512 MB |
| ResourceManager | Container Memory Maximum | yarn.scheduler.maximum-allocation-mb | 1024MB |
| NodeManager | Container Memory | yarn.nodemanager.resource.memory-mb | 4096MB |
| Gateway | Map Task Memory | mapreduce.map.memory.mb |  |
| Gateway | Map Task Java Opts Base | mapreduce.map.java.opts |  |
| Gateway | Reduce Task Memory | mapreduce.reduce.memory.mb |  |
| Gateway | Reduce Task Java Opts Base | mapreduce.reduce.java.opts |  |
| Gateway | ApplicationMaster Memory | yarn.app.mapreduce.am.resource.mb |  |
| Gateway | ApplicationMaster Java Opts Base | yarn.app.mapreduce.am.command-opts |  |
| Gateway | I/O Sort Memory Buffer (MiB) | mapreduce.task.io.sort.mb |  |
| Gateway | Mapreduce Task out | mapreduce.task. out |  |
| Gateway | I/O Sort Factor | mapreduce.task.io.sort.factor |  |
| Gateway | Default Number of Parallel Transfers During Shuffle | mapreduce.reduce.shuffle.parallelcopies |  |
| ResourceManager | Java Heap Size of ResourceManager in Bytes |  | 1024MB |

## HDFS TESTS

The following tests are used for HDFS benchmark testing.

### TestDFSIO

The TestDFSIO benchmark is a read and write test for HDFS. It is helpful for tasks such as stress testing HDFS, to discover performance bottlenecks in your network, to shake out the hardware, OS and Hadoop setup of your cluster machines (particularly the NameNode and the DataNodes) and to give you a first impression of how fast your cluster is in terms of I/O.

**Scenarios 1**: Run the TestDFSIO on 10 Files of 10MB each.

Write 10 Files of 10MB each = total 100MB

hadoop jar /usr/hdp/2.4.2.0-258/hadoop-mapreduce/hadoop-mapreduce-client-jobclient-tests.jarTestDFSIO -write -nrFiles 10 -fileSize 10

Read 10 Files of 10MB each = total 100MB

hadoop jar /usr/hdp/2.4.2.0-258/hadoop-mapreduce/hadoop-mapreduce-client-jobclient-tests.jarTestDFSIO -read -nrFiles 10 -fileSize 10

Clean up and remove test data

hadoop jar /usr/hdp/2.4.2.0-258/hadoop-mapreduce/hadoop-mapreduce-client-jobclient-tests.jarTestDFSIO –clean

**Scenarios 2:** Run the TestDFSIO on 100 Files of 10MB each.

Write 100 Files of 10MB each = total 100MB

hadoop jar /usr/hdp/2.4.2.0-258/hadoop-mapreduce/hadoop-mapreduce-client-jobclient-tests.jarTestDFSIO -write -nrFiles 100 -fileSize 10

Read 100 Files of 1GB each = total 100GB

hadoop jar /usr/hdp/2.4.2.0-258/hadoop-mapreduce/hadoop-mapreduce-client-jobclient-tests.jarTestDFSIO -read -nrFiles 100 -fileSize 10

Clean up and remove test data

hadoop jar /usr/hdp/2.4.2.0-258/hadoop-mapreduce/hadoop-mapreduce-client-jobclient-tests.jarTestDFSIO –clean

Results:

**USECASES**

* 1. Write 10 Files of 10MB each = total 100MB
  2. Read 10 Files of 10MB each = total 100MB
  3. Write 100 Files of 10MB each = total 1GB
  4. Read 100 Files of 10MB each = total 1GB

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Use case | Throughput [mp/sec] | Avg I/O Rate [mb/sec] | IO rate std deviation | Total exec [sec] |
| Write 10 Files of 10 MB each = total 100MB | 71.67739438335937 | 72.71614074707031 | 8.591936390942998 | 217.23 |
| Read 10 Files of 10MB each = total 100MB | 100.03000900270081 | 100.32747650146484 | 5.30274820593243 | 153.679 |
| Write 100 Files of 10MB each = total 1GB | 74.53753188342927 | 78.69841003417969 | 26.081204025209914 | 1829.841 |
| Read 100 Files of 10MB each = total 1GB | 90.67029831434849 | 91.14476776123047 | 6.575542187824743 | 1521.729 |

**REMARKS**

The TestDFSIO executed successfully on the cluster on both the scenarios, no performance issues were observed during the execution.

### TERASORT BENCHMARK SUITE

TeraSort is to sort 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 a Hadoop cluster.

A full 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.

To run the Terasort Benchmark test, execute run\_terasort.sh



**Scenario 1:** Run Terasort benchmark test with mapreduce.task.io.sort.mb to 10MB

**Scenario 2**: Run Terasort benchmark test with mapreduce.task.io.sort.mb to 100MB

|  |  |  |
| --- | --- | --- |
| **Operation** | **Data** | **Total exec (ms)** |
| TeraGen | 10MB | 7294 |
| TeraSort | 10MB | 7002 |
| TeraValidate | 10MB | 8576 |
| TeraGen | 100MB | 8566 |
| TeraSort | 100MB | 7127 |
| TeraValidate | 100MB | 10999 |

### 

### 3. NAMENODE BENCHMARK (NNBENCH)

NNBench is useful for load testing the NameNode hardware and configuration. It generates a lot of HDFS-related requests with normally very small “payloads” for the sole purpose of putting a high HDFS management stress on the NameNode. The benchmark can simulate requests for creating, reading, renaming and deleting files on HDFS.

**Scenarios 1:** Run NNBench Create\_Write and Open\_Read scenarios with 1 file

1. **Create\_Write with 1 File**

hadoop jar /usr/hdp/2.4.2.0-258/hadoop-mapreduce/hadoop-mapreduce-client-jobclient-tests.jar nnbench -operation create\_write -blockSize 1048576

1. **Open\_Read with 1 File**

hadoop jar /usr/hdp/2.4.2.0-258/hadoop-mapreduce/hadoop-mapreduce-client-jobclient-tests.jar nnbench -operation open\_read -blockSize 1048576

**Results:**

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Scenarios** | **Parameter** | **TPS** | **Avg exec (ms)** | **Avg Lat(ms)** | **Avg Lat (ms)** | **AL Total #1** | **AL Total #2** | **TPS Total (ms)** | **Longest Map (ms)** | **Late maps** | **# of exceptions** | **Remarks** |
| Create\_Write with 1 File | nnbench  -operation create\_write  -maps 1  -reduces 1  -blockSize 1048576  -bytesToWrite 0  -numberOfFiles 1  -replicationFactorPerFile 1  -readFileAfterOpen false  -baseDir /benchmarks/NNBench | (Create/Write)90 | 22.0 | (Create/Write)18.0 | (Close)4.0 | 18 | 4 | 22 | 22.0 | 0 | 0 | real 2m7.150s  user 0m4.011s  sys 0m0.287s |
| Open\_Read with 1 File | nnbench  -operation open\_read  -maps 1  -reduces 1  -blockSize 1048576  -bytesToWrite 0  -numberOfFiles 1  -replicationFactorPerFile 1  -readFileAfterOpen true  -baseDir /benchmarks/NNBench | (Open/Read) 333 | (Open/Read) 3.0 | (Open) 3.0 | (Read) 0.0 | 3 | 0 | 3 | 3.0 | 0 | 0 | real 2m7.177s  user 0m4.003s  sys 0m0.280s |

**Scenarios 2: Run NNBench Create\_Write and Open\_Read scenarios with 1000 file**

1. **Create\_Write with 1000 File**

hadoop jar /usr/hdp/2.4.2.0-258/hadoop-mapreduce/hadoop-mapreduce-client-jobclient-tests.jar nnbench -operation create\_write -numberOfFiles 1000 -blockSize 1048576

1. **Open\_Read with 1000 File**

hadoop /usr/hdp/2.4.2.0-258/hadoop-mapreduce/hadoop-mapreduce-client-jobclient-tests.jar nnbench -operation open\_read -numberOfFiles 1000 -blockSize 1048576

**Results:**

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Scenario** | **Parameter** | **TPS** | **Avg exec (ms)** | **Avg Lat(ms)** | **Avg Lat (ms)** | **AL Total #1** | **AL Total #2** | **TPS Total (ms)** | **Longest Map (ms)** | **Late maps** | **# of exceptions** | **Remarks** |
| Create\_Write with 1000 File | nnbench  -operation create\_write  -maps 1  -reduces 1  -blockSize 1048576  -bytesToWrite 0  -numberOfFiles 1000  -replicationFactorPerFile 1  -readFileAfterOpen false  -baseDir /benchmarks/NNBench | (Create/Write) 978 | 2.043 | (Create/Write) 1.108 | (Close) 0.872 | 1108 | 872 | 2043 | 2043.0 | 0 | 0 | real 2m10.385s  user 0m4.376s  sys 0m0.289s |
| Open\_Read with 1000 File | nnbench  -operation open\_read  -maps 1  -reduces 1  -blockSize 1048576  -bytesToWrite 0  -numberOfFiles 1000  -replicationFactorPerFile 1  -readFileAfterOpen true  -baseDir /benchmarks/NNBench | (Open/Read) 1968 | (Open/Read) 0.508 | (Open) 0.453 | (Read) 0.0 | 453 | 0 | 508 | 508.0 | 0 | 0 | real 2m7.151s  user 0m4.011s  sys 0m0.236s |

**Remarks:**

The NNBench benchmark executed successfully on the cluster, no read/write exceptions found in both the scenarios.

## 4. MAP REDUCE TESTS

### MapReduce Benchmark (MRBench)

MRBench loops a small job a number of times. As such it is a very complimentary benchmark to the “large-scale” TeraSort benchmark suite because MRBench checks whether small job runs are responsive and running efficiently on your cluster. It puts its focus on the MapReduce layer as its impact on the HDFS layer is very limited.

hadoop jar /usr/hdp/2.4.2.0-258/hadoop-mapreduce/hadoop-mapreduce-client-jobclient-2.7.1.2.4.2.0-258-tests.jar mrbench -numRuns 50

**Results:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Parameters** | **DataLines** | **Maps** | **Reduces** | **Avg (ms)** | **Remarks** |
| numRuns 50 | 1 | 2 | 1 | 13568 | real    11m20.226s  user    0m10.888s  sys     0m0.789s |

**Remarks:**

The MRBench executed successfully with number of runs 50 and average execution 13.56 sec.

## CONCLUSION

The benchmark tests executed successfully, we haven’t observed any issue.

**REFERENCES**

* <http://www.michael-noll.com/blog/2011/04/09/benchmarking-and-stress-testing-an-hadoop-cluster-with-terasort-testdfsio-nnbench-mrbench/#namenode-benchmark-nnbench>
* <http://epaulson.github.io/HadoopInternals/benchmarks.html#nnbench>