POLITECNICO DI MILANO

FACOLTÁ DI INGEGNERIA DELL'INFORMAZIONE

POLO REGIONALE DI COMO

MASTER OF SCIENCE IN COMPUTER ENGINEERING

REVIEW OF PERFORMANCE EVALUATION BENCHMARKS OF APACHE HADOOP

SUPERVISOR: PROF. MARCO GRIBAUDO

MASTER GRADUATION THESIS BY:

BLENDI PUSTINA

ID NUMBER: 749598

ACADEMIC YEAR: 2013/2014

Preface

To my family who supported me unconditionally and believed in me.

To Italy, for giving me this opportunity and unforgettable time.

Acknowledgement

I would like to express my sincerest gratitude to Prof Marco Gribaudo and Pietro Piazzolla for guiding and helping me throughout the entire process of writing this thesis. Their expertise in this field motivated and encouraged me to take into consideration big data and cloud computing for my future career goals.

Abstract

With the Internet and data growth increasing trends, big data is becoming an extremely important and challenging problem for Data Centers. Many platforms and frameworks are working to bring a cutting edge technology to this problem.

Apache Hadoop is a software framework addressing the big-data processing and storing on clusters, providing reliability, scalability and distributed computing.

Hadoop has a distributed file system to store vast amount of data in distributed environments, and uses MapReduce algorithm to perform the computations and process large amount of data, by parallelizing the workload and storage. In comparison to other relational database systems, Hadoop works well with unstructured data.

Our work is focused on performance evaluation of benchmarks of Hadoop, which are crucial for testing the infrastructure of the clusters. Taking into consideration the sensitiveness and importance of data, it's inevitable testing the clusters and distributed systems before deploying. The benchmark results can lead to optimizing the parameters for an enhanced performance tuning of the cluster.

This thesis covers the necessary related topics of Hadoop and a comprehensive listing of benchmarks used to test Hadoop, while providing detailed information for their appliance and procedures to run them.

We tested benchmarks in a virtual environment, with different parameters and options which yielded results that led to the conclusion of this thesis.

Keywords: hadoop, big data, mapreduce, benchmark, distributed filesystem

Table of Contents

1. Introduction

Apache Hadoop initially derived from a white paper published by Google, for Google file System (GFS), that was built to provide them efficiency and reliability to their clusters.

Hadoop is a software used for distributed infrastructures with particular focus on big data, ensuring scalability and reliability for distributed data centers. Hadoop has its own filesystem, Hadoop Filesystem or HDFS that differs from other filesystem due its large block size, which was designed to be robust to many problems that other Distributed Filesystems couldn't handle such as fault tolerance, reliability and scalability. Distributed systems have existed before, but with Hadoop the data and work are automatically distributed across machines and the CPU usage are parallelized. As most distributed filesystem, HDFS is based in an architecture where data is separated from the namespace.

HDFS is a best effort solution to fault tolerance in a very large data center. Its purpose is to distribute large amount of data across many machines (nodes). As an input Hadoop receives a very large file and divides it into chunks called blocks, which then are stored and replicated across different machines. So when a machine in the distributed environment will fail due to some problem, HDFS will provide the missing blocks which were replicated in some other machines. In this way HDFS ensures reliability so end users won't be affected by a single machine failure since data is distributed.

Although Hadoop is a recently developed framework, there are great expectations in its quick development and spreading in the near future. According to Philip Russom, as revealed in his research on 'Integrating Hadoop into Business Intelligence and Data Warehousing', first hand users of Hadoop have confessed that there are still many areas of improvement to the software in regards to security, administration, availability and others. However, the open source community is continuously addressing these issues and is also steadily contributing to improving Hadoop products which leads to anticipating greater capabilities and usage of the software in the next years. In addition, a survey of organizations conducted under the umbrella of this research, 10% of the organizations stated that they are already using Hadoop in their production, while a surprising 51% are expected to use one in the next three years. Russom also states that: *"If this trend pans out, Hadoop will impact at least half of BI/DW environments soon. Hence, users need to prepare for Hadoop usage now."* (Russom, 2013)

Hadoop has gained considerable attention, but it still remains a new software out there and many professionals of the field are not sure on what it is and what is does. Russom explains

10 facts in regards to Hadoop that are currently misunderstood or unknown to many users of the software:

1. Contrary to the opinion of many, Hadoop is not one single software. It is rather a group of open source products which are provided by Apache Software Foundation. Hadoops library includes: HDFS, MapReduce, Pig, Hive and others.

2. Hadoop's open source software are initially and mainly offered from Apache. However there also a few other vendors that offer a more complete package that includes more tools and better ways to use it.

3. Hadoop represents a growing ecosystem of products that are provided by vendors other than Apache too. These products from other vendors blend with Hadoop technologies and are used to expand them.

4. Hadoop is not a database management system but its main purpose is to serve as a distributed file system. As such, Hadoop does not have capacities of a DBMS (i.e. indexing, query optimization, random access to data, etc.). However, on the other hand, the advantages of Hadoop distributed file system is to manage and process large masses of files and data which is unstructured.

5. Although Hive, a Hadoop product, resembles SQL a lot, there are many issues that arise when it comes to compatibility with SQL based tools.

6. Hadoop is related to MapReduce, however they can work independently. There are users who deploy HDFS using Hive and Hbase, without the use of MapReduce.

7. MapReduce is not only used for analytics, but is an execution engine used for general purposes which is capable to process complexities of network communication.

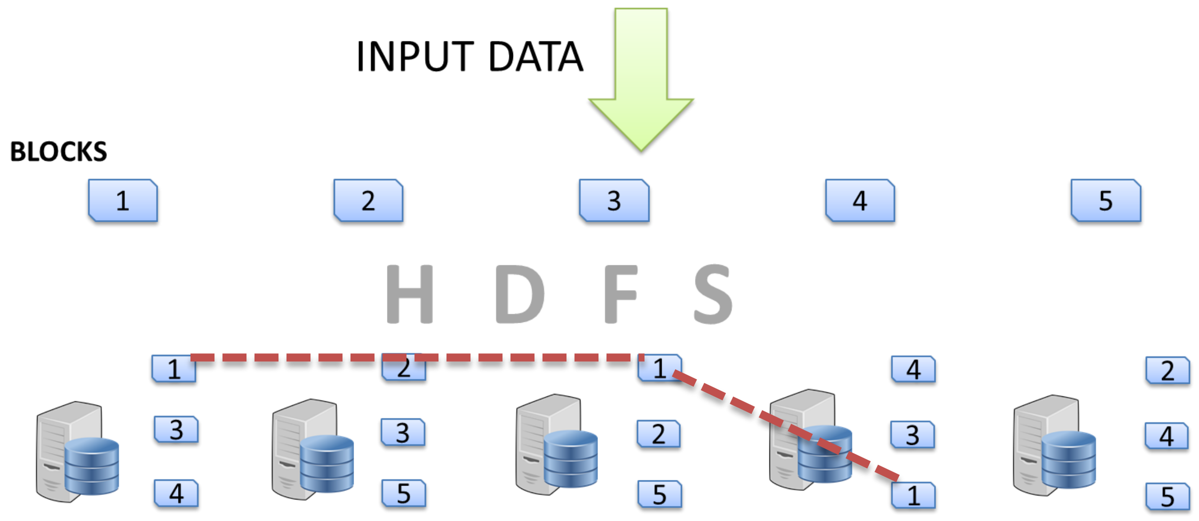
8. Hadoop not only processes large amounts of data but also has capabilities of processing many different types of data given that these data can be put in a file and copied into Hadoop.

9. Hadoop serves as a complement to Data Warehousing, however it cannot be considered a replacement for it.

10. Although Hadoop receives most of attention because of the way companies use it to perform web analytics, it has capabilities to perform analytics of other types of large piles of data such as those coming from sensory devices and robotics. *(Philip Russom,* "Integrating Hadoop into Business Intelligence and DataWarehousing"

1.1. Hadoop Architecture

HDFS architecture consists of nodes or machines which are grouped in a rack, whereas several racks form a HDFS cluster. HDFS cluster has several slave nodes or DataNodes and a Master node or NameNode. HDFS stores filesystem metadata on NameNode dedicated server and application data on another dedicated server called DataNodes. In Hadoop datanodes do not rely on RAID protected mechanism, instead the data is replicated in multiple datanodes for reliability.



All servers are fully connected to each other and communicate using TCP based protocol.

The HDFS file replication starts by a request submitted by the client in NameNode, which checks if the file does already exist, and if the client has the permission to write the file. The File is chunked into blocks (default size 64MB) which are then used to be placed into DataNodes.

NameNode then determines the DataNode to which to write the first block to. If client is running on DataNode, it will first try to place it there, otherwise it chooses a DataNode at random. By default DataNode is replicated at two other nodes in the cluster, preferring different rack from the first block.

A pipeline is built between DataNodes where blocks are stored. When the blocks are written an acknowledgement is sent back to HDFS client and NameNode completes the process if the minimum blocks replicas are placed.

Apache Hadoop YARN is a sub-project of Hadoop at the Apache Software Foundation introduced in Hadoop 2.0 that separates the resource management and processing components. YARN was born of a need to enable a broader array of interaction patterns for data stored in HDFS beyond MapReduce. The YARN-based architecture of Hadoop 2.0 provides a more general processing platform that is not constrained to MapReduce. (Horton Works)

1.2. Map Reduce

Hadoop MapReduce is a network-based parallel programming paradigm for implementing data computation over big data using a cluster of community computer systems (Moody & Jobs)

MapReduce, is divided into two major separate tasks, Mapping and Reducing. In the first phase, the input data is converted to another type of data by using filtering and sorting techniques, where the elements of the data are chunked and grouped in a key/value pairs.

The Reduce task, collects the intermediate data and merges them into a single output, by showing their occurrence in intermediate phase.

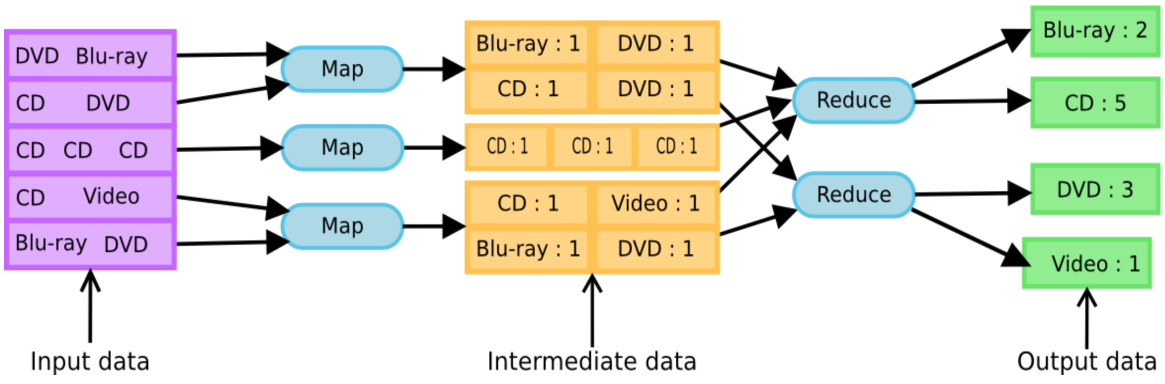


Figure 1 Map Reduce

MapReduce Foundation: Typing of MapReduce.

http://www.infosun.fim.uni-passau.de/cl/MapReduceFoundations

A good example in the figure by MapReduce Foundation, shows the input data as it is received by Mapping and transferred in an intermediate phase to be then proceeded by Reduce task.

The reason why MapReduce algorithm is applied to distributed filesystem, is its ability to spread the data processing into parallel computing. By applying MapReduce, a large amount of processing will be done by several processing units, and by doing so, it's capable of handling vast amount of data in Hadoop.

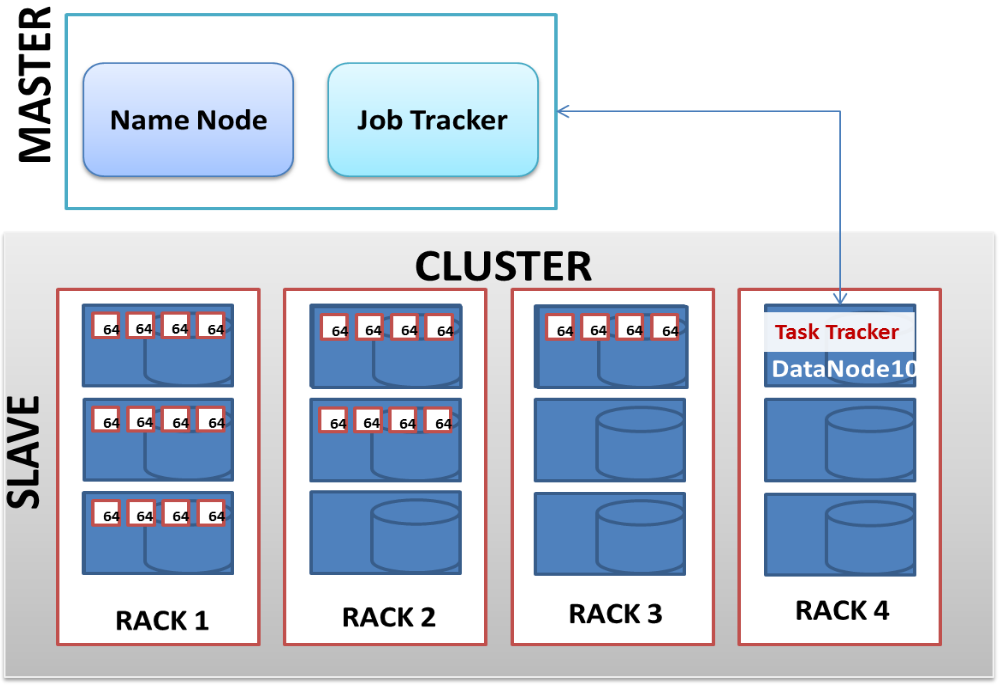
One key feature of MapReduce that differentiates it from previous models of parallel computation is that it interleaves sequential and parallel computation.
(Karloff, Siddharth, & Vassilvitskii‡)

1.3. Master Slave model

Hadoop Architecture is based on Master/Slave model which allows it to perform distributed data storage and distributed or parallel computing.

Master/slave is a model of communication where one device or process has unidirectional control over one or more other devices. In some systems a master is elected from a group of eligible devices, with the other devices acting in the role of slaves (Obaidat & Misra, 2011)

In Hadoop Clusters, NameNode has the role of the Master and DataNodes have the role of Slaves. Normally there is one NameNode per Cluster and several Racks consisting of several DataNodes.

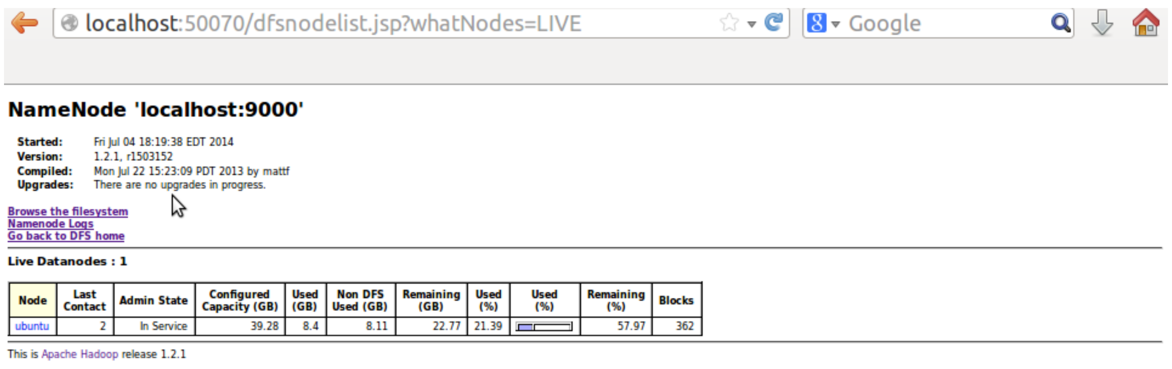


1.4. Hadoop Daemons

1.4.1. NameNode

NameNode distributes the workload and job executions across Slaves and stores filesystem metadata or the location where the data blocks are stored in DataNodes. NameNode's hardware performance should be taken into consideration since entire clusters information is stored in the Metadata files of NameNode. This is the core node of HDFS file system, and clients communicate first with namenode to locate a file or perform any operation on it.

In the following screenshot, you can the information that Namenode keep track of.



NameNode 'localhost:9000'

Started:	Fri Jul 04 18:19:38 EDT 2014
Version:	1.2.1, r1503152
Compiled:	Mon Jul 22 15:23:09 PDT 2013 by mattf
Upgrades:	There are no upgrades in progress.

Browse the filesystem
Namenode Logs
Go back to DFS home

Live Datanodes : 1

Node	Last Contact	Admin State	Configured Capacity (GB)	Used (GB)	Non DFS Used (GB)	Remaining (GB)	Used (%)	Used (%)	Remaining (%)	Blocks
ubuntu	2	In Service	39.28	8.4	8.11	22.77	21.39	▨	57.97	362

This is Apache Hadoop release 1.2.1

1.4.2. DataNode

DataNode Daemons are the storage nodes where the fileblocks of HDFS are stored. Basically it performs read and writes into storage from the client, by asking NameNode for the location of data in the cluster, saved on its metadata. DataNode continuously keeps NameNode informed for the actions and changes.

DataNode's send signals to NameNode indicating that they're alive usually referred as heartbeat of DataNode. If NameNode doesn't receive the heartbeat from DataNode in certain time, it considers it dead and replicates another copy of it, somewhere in the cluster.

Datanodes communicate with each other, to rebalance data and make sure the replication of blocks is done. A pipeline is built between datanodes, connecting the replicas of the same blocks in datanodes . If a block is damaged or is not sending heartbeat, with the command of namenode, the datanodes replicate it through the pipeline.

1.4.3. Secondary NameNode

Hadoop is very flexible and fault tolerant in DataNode's, whereas in NameNode is the critical part where all the information of Meta data is stored. As a solution to this, Hadoop provides a backup node for Namenode.
Secondary NameNode is the replica of NameNode, and keeps track of NameNode by taking checkpoints, so that in case NameNode goes down, it can be recovered from Secondary NameNode.

1.4.4. Job Tracker

The tasks of NameNode and DataNode are performed by Job Tracker and Task Tracker respectively. JobTracker keeps track and manages the lifecycle of TaskTrackers whereas TaskTrackers have to report to JobTracker about their tasks.
Job Tracker is the daemon service for submitting and tracking Map Reduce jobs in Hadoop. There is only One Job Tracker process run on any Hadoop cluster. (Vishwas & Shweta, 2013)

1.4.5. Task Tracker

A TaskTracker is a node in the cluster that accepts tasks - Map, Reduce and Shuffle operations - from a JobTracker. Every TaskTracker is configured with a set of slots, these indicate the number of tasks that it can accept. (Apache)

Each DataNode has its own TaskTracker, which mainly does the map and reduce functions on the nodes. Task tracker is configured preliminary for the number of tasks it can accept, simultaneously. When job tracker tries to assign a task to tasktracker, first it checks its availability.

1.4.6. Balancer

When HDFS stores its blocks into the datanodes, it might happen that some nodes are getting overloaded and some are empty. Having unbalanced blocks stored in your cluster, might cause bottlenecks. HDFS Balancer will balance the data blocks across the cluster, with a threshold parameter as an allowance of difference between DataNodes.

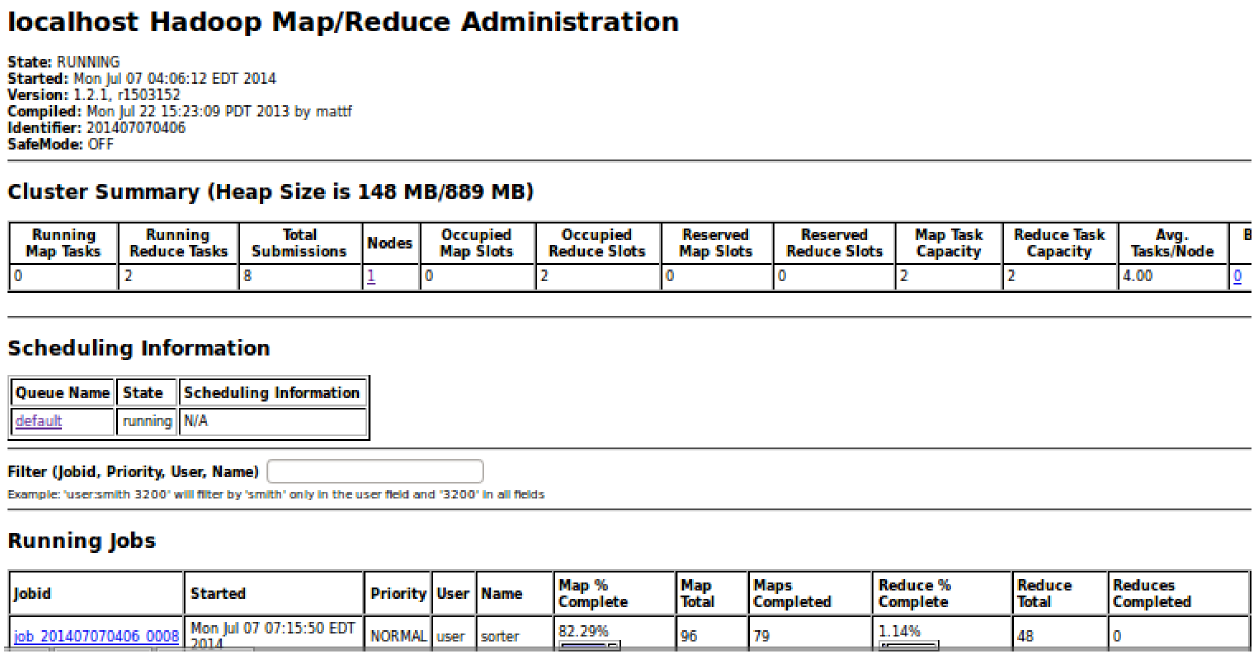
The command to run balancer

```
$ hdfs balancer -threshold 25

Threshold's value is from 1-100
```

1.4.7. Hadoop Web User Interface

Hadoop has its own Web User Interface, providing severe information for the tasks, states and data running in Hadoop. An interface is dedicated to Job Tracker showing the running and completed jobs with detailed information accessible by default via port 50030.



localhost Hadoop Map/Reduce Administration

State: RUNNING
Started: Mon Jul 07 04:06:12 EDT 2014
Version: 1.2.1, r1503152
Compiled: Mon Jul 22 15:23:09 PDT 2013 by mattf
Identifier: 201407070406
SafeMode: OFF

Cluster Summary (Heap Size is 148 MB/889 MB)

Running Map Tasks	Running Reduce Tasks	Total Submissions	Nodes	Occupied Map Slots	Occupied Reduce Slots	Reserved Map Slots	Reserved Reduce Slots	Map Task Capacity	Reduce Task Capacity	Avg. Tasks/Node	B
0	2	8	1	0	2	0	0	2	2	4.00	0

Scheduling Information

Queue Name	State	Scheduling Information
default	running	N/A

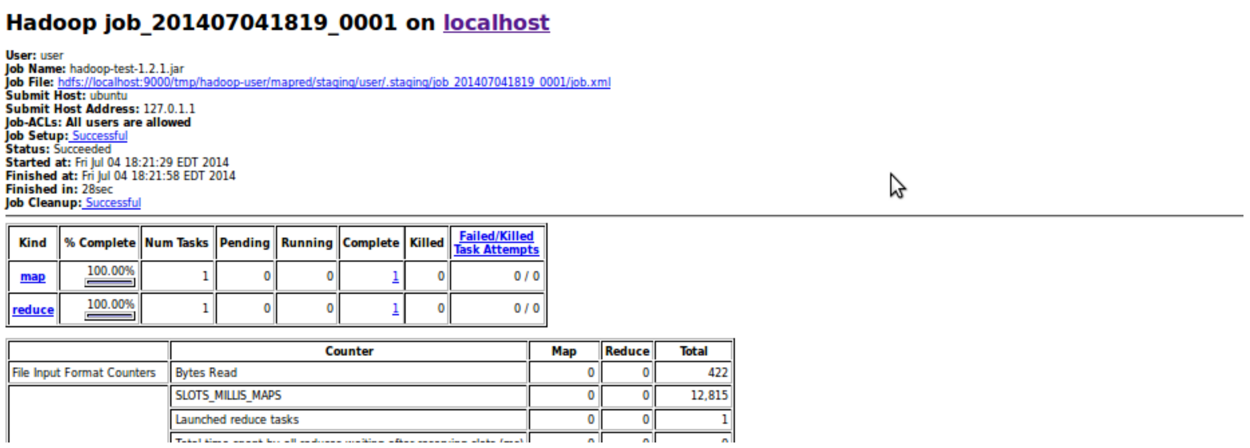
Filter (Jobid, Priority, User, Name)
Example: 'user:smith 3200' will filter by 'smith' only in the user field and '3200' in all fields

Running Jobs

Jobid	Started	Priority	User	Name	Map % Complete	Map Total	Maps Completed	Reduce % Complete	Reduce Total	Reduces Completed
job_201407070406_0008	Mon Jul 07 07:15:50 EDT 2014	NORMAL	user	sorter	82.29%	96	79	1.14%	48	0

On the screenshot above, a Cluster Summary shows the Nodes, Tasks, Map and Reduce Slots and Capacity information regarding the running Hadoop jobs.

Furthermore, jobs can be accessed and detailed information are shown on their page.



Hadoop job_201407041819_0001 on localhost

User: user
Job Name: hadoop-test-1.2.1.jar
Job File: hdfs://localhost:9000/tmp/hadoop-user/mapred/staging/user/.staging/job_201407041819_0001/job.xml
Submit Host: ubuntu
Submit Host Address: 127.0.1.1
Job-ACLs: All users are allowed
Job Setup: Successful
Status: Succeeded
Started at: Fri Jul 04 18:21:29 EDT 2014
Finished at: Fri Jul 04 18:21:58 EDT 2014
Finished in: 28sec
Job Cleanup: Successful

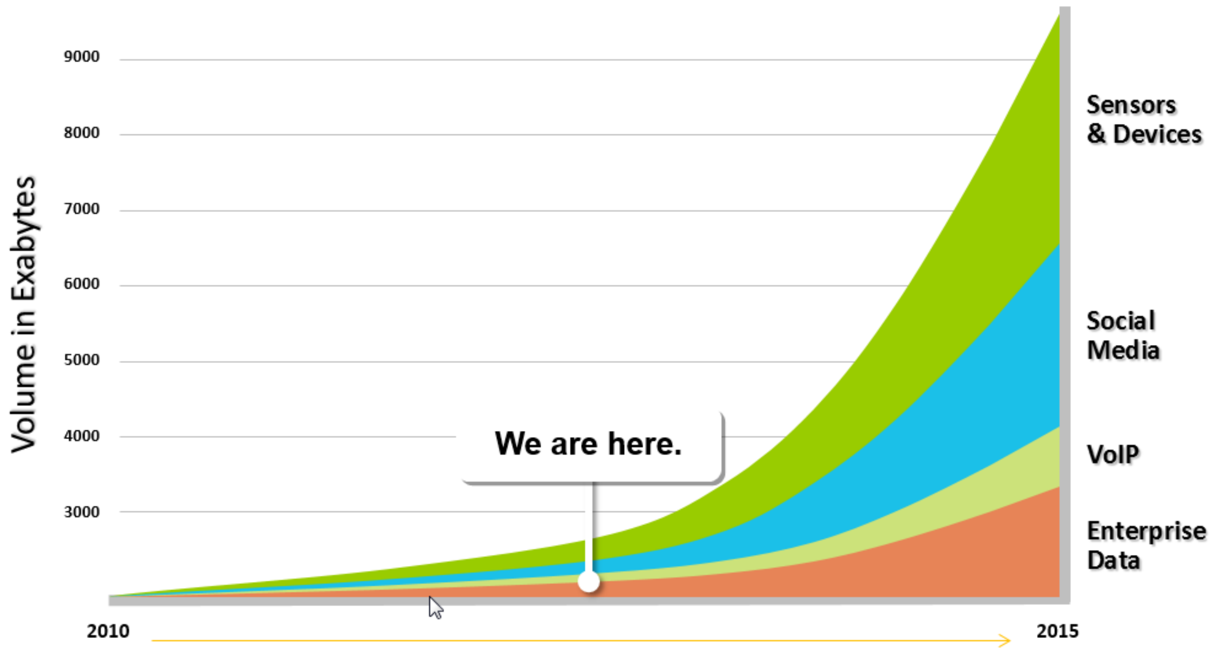
Kind	% Complete	Num Tasks	Pending	Running	Complete	Killed	Failed/Killed Task Attempts
map	100.00%	1	0	0	1	0	0 / 0
reduce	100.00%	1	0	0	1	0	0 / 0

	Counter	Map	Reduce	Total
File Input Format Counters	Bytes Read	0	0	422
	SLOTS_MILLIS_MAPS	0	0	12,815
	Launched reduce tasks	0	0	1
	Total time spent by all reduces waiting after reserving slots (ms)	0	0	0

1.5. Big Data

Big data is a large collection of data sets which differs from normal data stored in the sense that it requires special dedication to process and store. Recently, the big data solutions are cutting edge technologies in the market, which help companies and organizations maintain their infrastructure and cut financial cost.

According to CISCO, the amount of data center traffic will triple from 2012 to 2017.



(Tech Dynamics)

As you can see, in the graph above by Tech Dynamics, the amount of data is expecting to grow exponentially in the upcoming years, and the needs to process, distribute and store this data will have to meet the requirements.

1.5.1. Structured Data VS Unstructured Data

Structured data is data that is stored in a structured form in a relational database, stored on spreadsheet or filesystems. The structure of Data depends on its ability to be accessed, processed and stored, as in relational databases where data is easily searchable by using algorithm for querying it (such as Structured Query Language) . The opposite of Structured data is Unstructured data where, the data does not have a predefined data model and it usually comes as a text, or multimedia content. However the unstructured data content, internally it may contain structured content.

One of the main benefits of implementing Hadoop, is its ability to store unstructured data which wasn't able to be handled by traditional relational databases.

1.5.2. Hadoop Data

Hadoop was invented as a solution to vast amount of data, and struggle of handling such big data streams. However, the data generated and stored in Hadoop cluster, is playing an important role on organizations analysis and statistics to improve their business performance.

According to (Horton Works), there are five types of new data in Hadoop for this purpose.

Sentiment Data: The unstructured data generated by social networks, which can lead an organization to extracting information as how customers might feel about their products.

Clickstream Data : Data collected from the weblogs of the clicks, the user makes in a website.

Server Log Data: Log files saved by computer in networks which might be used in many manners.

Sensor / Machine Data: Automatic data generated by sensors and machines, by monitoring or measuring many factors. Extracting information from these logs can help organizations analyze and forecast the operations.

GeoLocation Data: GeoLocation data, mostly coming from mobile devices. Extracting this information can lead organizations to analyze the geographical density.

2. Performance Tuning

HDFS Parameters are the configurable parameters to tune Hadoop and customize it for the adequate performance. Most parameters are directly adjusting the performance, security, timing and addressing of Name node and Data nodes, where many features of Hadoop can be set, activated or deactivated.

Tuning Hadoop can be a very challenging task, due to many configurable parameters and distributed environment where many aspect have to be taken into consideration.

It requires techniques, such as data collection by running controlled experiments under different parameter settings, data analysis using optimization techniques, and analytical and reasoning skills. (Shumin, 2013)

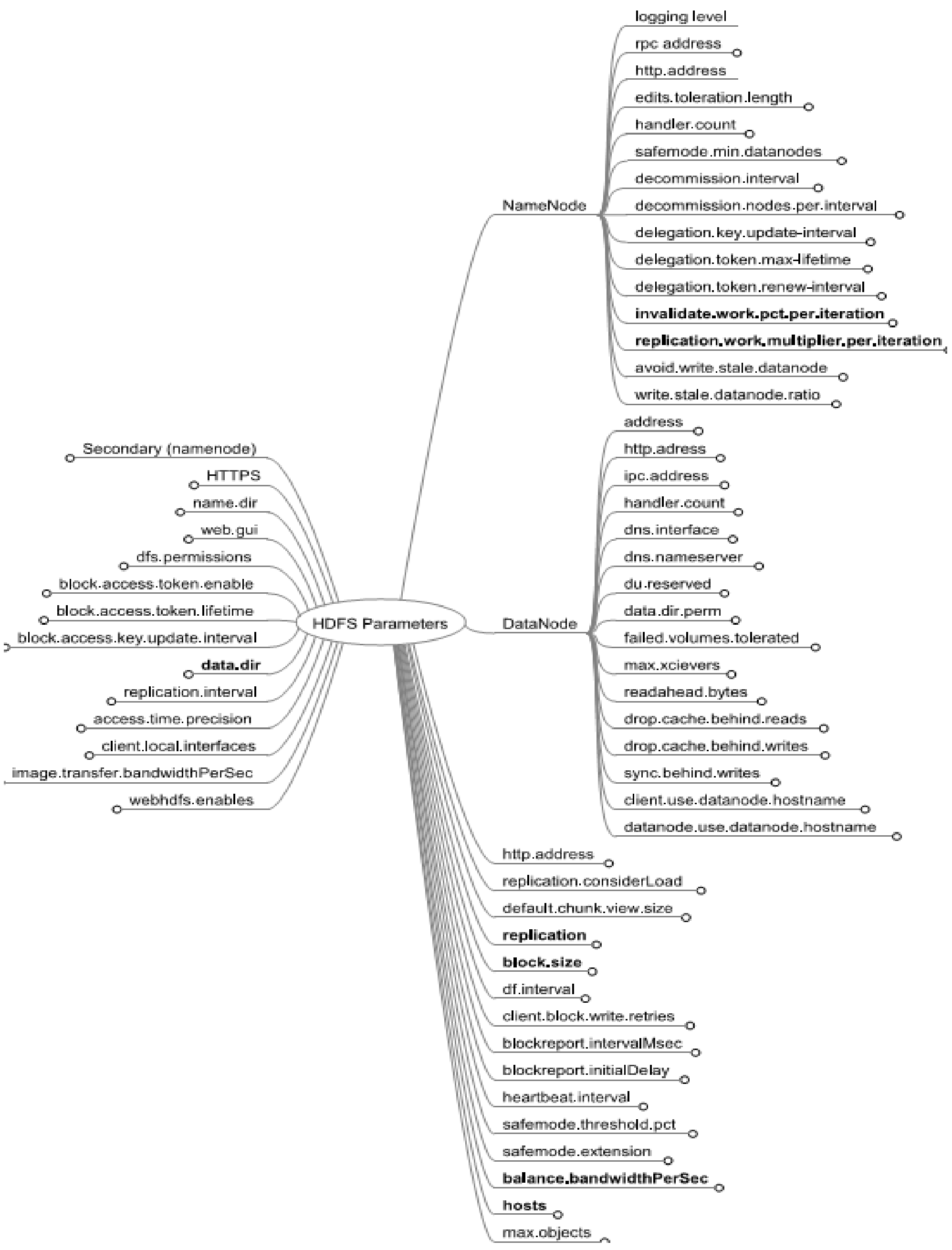
2.1. Parameters

Parameters can be added using the XML structure as the example below on hdfs-site.xml file, whereas the original ones can be found on hdfs-default.xml

Below are listed some of parameters which have higher impact on Hadoop’s performance.

```
<property>
            <name>fs.default.name</name>
            <value>hdfs://localhost:50030</value>
</property>
<property>
            <name>mapred.job.tracker</name>
            <value>hdfs://localhost:50031</value>
</property>
<property>
            <name>dfs.replication</name>
            <value>3</value>
</property>
<property>
<property>
  <name>dfs.block.size</name>
  <value>134217728</value>
  <description>The default block size for new files.</description>
</property>
<property>
  <name>dfs.default.chunk.view.size</name>
  <value>32768</value>
  <description>The number of bytes to view for a file on the browser.
  </description>
</property>
```

- **Replication(dfs.replication):** This is the value of how many times we want to replicate the blocks across the nodes in Cluster. It's the most critical parameter to affect the performance of our cluster. Normally, the default value of this parameter is 3, so the block files get replicated 3 times in the cluster. Setting it higher might lower the performance of our Cluster, whereas setting it lower might increase the risk of losing data.



- **Block size (dfs.block.size):** Here is where we set the size of blocks, partitioned from files that were chunked. The default size that comes with Hadoop installation is 64MB, whereas most commonly used is 128MB. Hadoop works best with large files, so setting it higher might (not in all cases) improve performance of Hadoop by lowering seek time.

- **Heartbeat.interval:** Datanodes send heartbeat signals to Namenode to confirm that they're alive. This parameter sets the interval in seconds to notify NameNode for each DataNode that is alive. (i.e. If NameNode doesn't receive heartbeat for a certain datanode in a certain amount of time, considers it dead)

- **failed.volumes.tolerated** : The limit number of volumes allowed to fail, before datanode stops offering service. By default, every volume fail will cause DataNode to stop.

- **client.block.write.retries** :How many times a user is allowed to try and write data, before he receives failure of application

- **dfs.default.chunk.view.size:** Number of bytes the user views when visiting a page in a browser. Default value is set to 32KB.

Performance of Hadoop Map-Reduce job can be increased without increasing the hardware cost, by just tuning some parameters according to the cluster specifications, input data size and processing complexities. (Hadoop Performance Testing, Technologies, Impetus)

3. Benchmarking

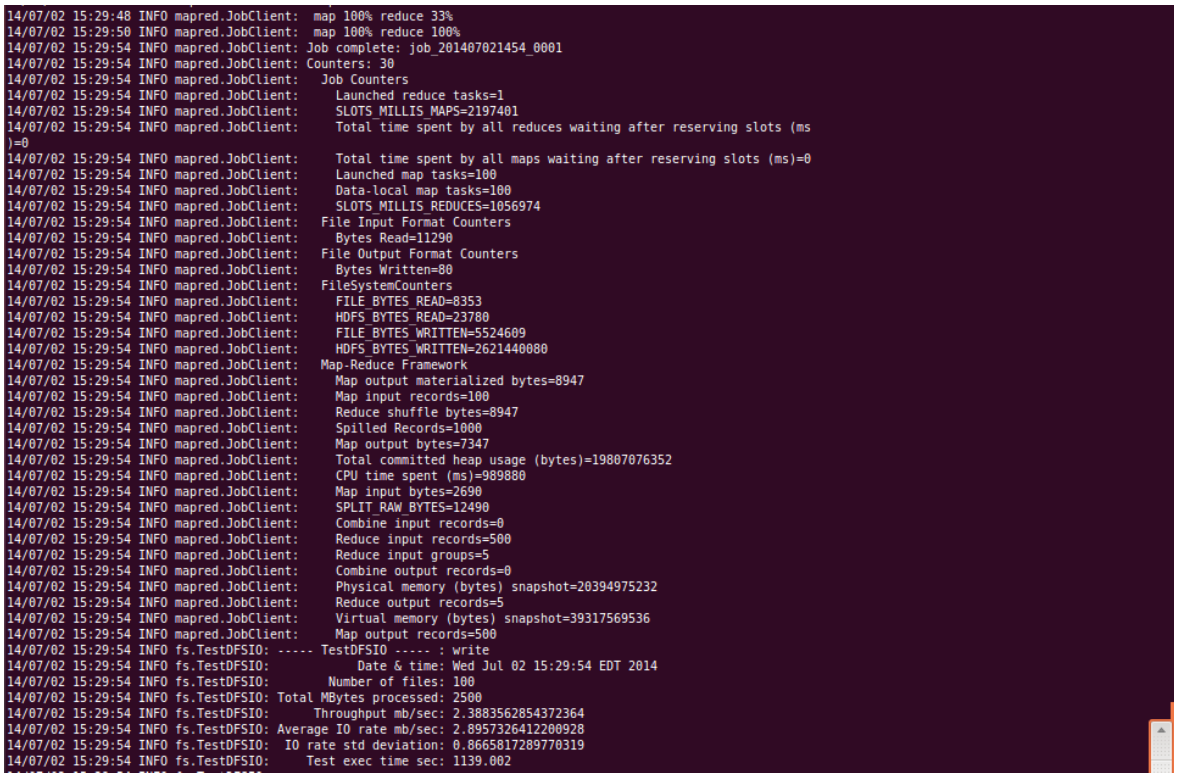
3.1. Introduction to Benchmarking in Hadoop

Benchmarking in distributed architecture systems where the performance of your clusters is affected by many hardware and software components is crucial. Benchmark is the evaluation of the capacity and performance, measured in many parameters which are yielded as outcome of benchmarking tests. Based on the results of these parameters we can decide how to tune Hadoop Cluster for best performance. The aim of Hadoop benchmarks is to push the workload to the limit and find the bottlenecks of cluster, by estimating the cost of tasks. Most benchmarks come with data generators(scripts that generate data), providing them the input data to be able to apply different algorithm on it, and test the performance of the cluster under heavy workload, trying to cause the bottlenecks of the system.

3.2. Counters

While running benchmarks in Hadoop, at each iteration a list of Counters is yielding statistical information about the running tests. Counter are grouped into different categories, listing detailed data on running benchmark, such as CPU time spent, Bytes read, Total time spent by reduce and map tasks etc.

Counters are useful to get statistical data from your running tests and be aware of the default values that are set on the benchmark. Counter are grouped into set of counters, based on the task they're conducting.



```
14/07/02 15:29:48 INFO mapred.JobClient:  map 100% reduce 33%
14/07/02 15:29:50 INFO mapred.JobClient:  map 100% reduce 100%
14/07/02 15:29:54 INFO mapred.JobClient: Job complete: job_201407021454_0001
14/07/02 15:29:54 INFO mapred.JobClient: Counters: 30
14/07/02 15:29:54 INFO mapred.JobClient:   Job Counters
14/07/02 15:29:54 INFO mapred.JobClient:     Launched reduce tasks=1
14/07/02 15:29:54 INFO mapred.JobClient:     SLOTS_MILLIS_MAPS=2197401
14/07/02 15:29:54 INFO mapred.JobClient:     Total time spent by all reduces waiting after reserving slots (ms
)=0
14/07/02 15:29:54 INFO mapred.JobClient:     Total time spent by all maps waiting after reserving slots (ms)=0
14/07/02 15:29:54 INFO mapred.JobClient:     Launched map tasks=100
14/07/02 15:29:54 INFO mapred.JobClient:     Data-local map tasks=100
14/07/02 15:29:54 INFO mapred.JobClient:     SLOTS_MILLIS_REDUCES=1056974
14/07/02 15:29:54 INFO mapred.JobClient:   File Input Format Counters
14/07/02 15:29:54 INFO mapred.JobClient:     Bytes Read=11290
14/07/02 15:29:54 INFO mapred.JobClient:   File Output Format Counters
14/07/02 15:29:54 INFO mapred.JobClient:     Bytes Written=80
14/07/02 15:29:54 INFO mapred.JobClient:   FileSystemCounters
14/07/02 15:29:54 INFO mapred.JobClient:     FILE_BYTES_READ=8353
14/07/02 15:29:54 INFO mapred.JobClient:     HDFS_BYTES_READ=23780
14/07/02 15:29:54 INFO mapred.JobClient:     FILE_BYTES_WRITTEN=5524609
14/07/02 15:29:54 INFO mapred.JobClient:     HDFS_BYTES_WRITTEN=2621440080
14/07/02 15:29:54 INFO mapred.JobClient:   Map-Reduce Framework
14/07/02 15:29:54 INFO mapred.JobClient:     Map output materialized bytes=8947
14/07/02 15:29:54 INFO mapred.JobClient:     Map input records=100
14/07/02 15:29:54 INFO mapred.JobClient:     Reduce shuffle bytes=8947
14/07/02 15:29:54 INFO mapred.JobClient:     Spilled Records=1000
14/07/02 15:29:54 INFO mapred.JobClient:     Map output bytes=7347
14/07/02 15:29:54 INFO mapred.JobClient:     Total committed heap usage (bytes)=19807076352
14/07/02 15:29:54 INFO mapred.JobClient:     CPU time spent (ms)=989880
14/07/02 15:29:54 INFO mapred.JobClient:     Map input bytes=2690
14/07/02 15:29:54 INFO mapred.JobClient:     SPLIT_RAW_BYTES=12490
14/07/02 15:29:54 INFO mapred.JobClient:     Combine input records=0
14/07/02 15:29:54 INFO mapred.JobClient:     Reduce input records=500
14/07/02 15:29:54 INFO mapred.JobClient:     Reduce input groups=5
14/07/02 15:29:54 INFO mapred.JobClient:     Combine output records=0
14/07/02 15:29:54 INFO mapred.JobClient:     Physical memory (bytes) snapshot=20394975232
14/07/02 15:29:54 INFO mapred.JobClient:     Reduce output records=5
14/07/02 15:29:54 INFO mapred.JobClient:     Virtual memory (bytes) snapshot=39317569536
14/07/02 15:29:54 INFO mapred.JobClient:     Map output records=500
14/07/02 15:29:54 INFO fs.TestDFSIO: ----- TestDFSIO ----- : write
14/07/02 15:29:54 INFO fs.TestDFSIO:            Date & time: Wed Jul 02 15:29:54 EDT 2014
14/07/02 15:29:54 INFO fs.TestDFSIO:        Number of files: 100
14/07/02 15:29:54 INFO fs.TestDFSIO: Total MBytes processed: 2500
14/07/02 15:29:54 INFO fs.TestDFSIO:      Throughput mb/sec: 2.3883562854372364
14/07/02 15:29:54 INFO fs.TestDFSIO: Average IO rate mb/sec: 2.8957326412200928
14/07/02 15:29:54 INFO fs.TestDFSIO:  IO rate std deviation: 0.8665817289770319
14/07/02 15:29:54 INFO fs.TestDFSIO:     Test exec time sec: 1139.002
```

In the screenshot above, you can see a list of counters, showing details of computations from a TestDFSIO benchmark.

3.3. Hadoop Benchmarks

In this section, a list of Benchmark Suites will be briefly described, showing necessary information to run them and adjust parameters, as well as the yielded outcome after running them.

3.3.1. HDFS Test Benchmark Suite

The following benchmarks come with the Hadoop installation package.

TestFileSystem: Tests the HDFS filesystem by generating a number of files with certain size. This benchmark does not yield any outcome, if not received any error the testing is considered successful.

- **Parameters:**
  - Number of files
  - Number of MB per file
- **Expected Output:**

There is no output for this benchmark, if there is no error outcome it means the testing was successful and filesystem is ready.

- **Command**:

```
hadoop jar $HADOOP_HOME/hadoop-test.jar testfilesystem -files 20 - megaBytes
20
```

DistributedFSCheck (Write) : This benchmark will check the read and write consistency on distributed filesystem, giving a workload with number of files with certain size.

- **Parameters:**
  - Number of files
  - Number of MB per file
- **Expected Output:**
  - Throughput mb/sec
  - Average IO rate mb/sec
  - IO rate std deviation
  - Test exec time sec

- **Command**

hadoop jar $HADOOP_HOME/hadoop-test.jar DistributedFSCheck -write - nrFiles 10 - fileSize 50

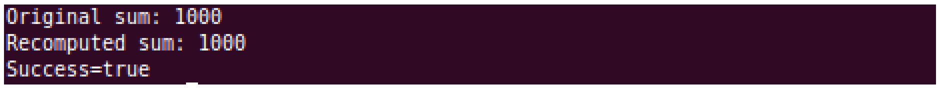
hadoop jar $HADOOP_HOME/hadoop-test.jar DistributedFSCheck -read - nrFiles 10 - fileSize 50

3.3.2. Hadoop MapReduce Benchmark Suite

The following benchmarks come with the Hadoop installation package.

MapRedTest: This benchmark tries to benchmark Map Reduce by loading a number of random generated integers, which test read and write capabilities.

- **Parameters:**
  - Range
  - Count
- **Expected Output:**
  - Original sum
  - Recomputed sum
  - Success / Fail

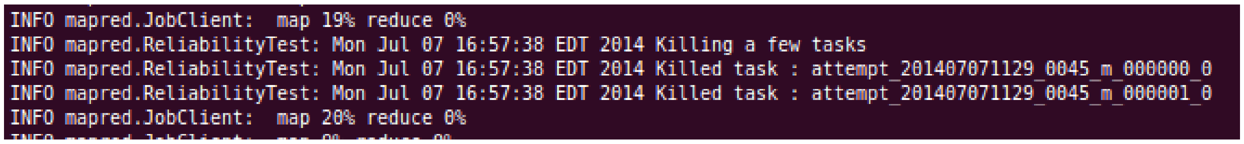


```
Original sum: 1000
Recomputed sum: 1000
Success=true
```

- **Command**

hadoop jar $HADOOP_HOME/hadoop-*test*.jar mapredtest 5 500

MRReliabilityTest: This benchmark will test the failure of Task Tracker, by making it fail intentionally and killing its task several times. By killing tasks, jobs will not be failing if the cluster is flexible to task failures. This test checks the task failure of MapReduce, so if job will fail, it means we have to tune the task trackers performance.



```
INFO mapred.JobClient:  map 19% reduce 0%
INFO mapred.ReliabilityTest: Mon Jul 07 16:57:38 EDT 2014 Killing a few tasks
INFO mapred.ReliabilityTest: Mon Jul 07 16:57:38 EDT 2014 Killed task : attempt_201407071129_0045_m_000000_0
INFO mapred.ReliabilityTest: Mon Jul 07 16:57:38 EDT 2014 Killed task : attempt_201407071129_0045_m_000001_0
INFO mapred.JobClient:  map 20% reduce 0%
```

This benchmarks requires to run solely, with no other tasks running in parallel.

- **Command**

```
hadoop jar $HADOOP_HOME/hadoop-test-*.jar MRReliabilityTest –libjars HADOOP_HOME/hadoop-examples-*.jar
```

MRBench: MRBench or MapReduce Benchmark is one of main benchmarks of MapReduce, it executes a large number of small jobs to be able to check the capability responsiveness of the map reduce in the cluster.

- **Parameters:**
  - numRuns [How many times the job will run]
  - maps
  - reduces
  - inputLines
  - inputType
- **Expected Output:**
  - AvgTime (milliseconds)
- **Command**

```
hadoop jar $HADOOP_HOME/hadoop-test-*.jar mrbench -numRuns 15
```

LoadGen: Load Generator Benchmark

- **Parameters:**
  - Maps
  - Reduces
  - keepmap
  - keepred
- **Expected Output:**
  - Original sum
  - Recomputed sum
  - Success
- **Command**

```
hadoop jar $HADOOP_HOME/hadoop-test .jar loadgen -m 100 -r 10 -keepmap 50 -keepred 50 -indir input -outdir output
```

NNBench : Namenode is the master part of an HDFS filesystem, and it keeps all the information regarding other files stored in the cluster. NNBench or Namenode benchmark is capable of making a powerful load test of NameNode's hardware and configuration, by generating many requests with small payloads in order of putting HDFS in a stress-test of NameNode. Request operations that this benchmark generates can be of reading, creating, deleting and renaming files. Moreover, this benchmark can be adjusted with several parameters to perform different type of tests.

- **Parameters:**
  - Operation :
    - create_write
    - open_read
    - rename_delete
  - maps
  - reduces
  - startTime
  - blockSize
  - bytesToWrite
  - bytesPerChecksum
  - numberOfFiles
  - replicationFactorPerFile
  - baseDir
  - readFileAfterOpen

- **Expected Output:**
  - TPS: Create/Write/Close
  - Avg Exec Time(ms): Create/Write/Close
  - Avg Latency(ms) / Create/Write
  - Avg Latency / Close
  - RAW DATA: AL Total
  - RAW DATA: TPC Total(ms)
  - RAW DATA: Longest Map Time (ms)
  - RAW DATA: Late maps
  - RAW DATA: # of executions

- **Command**

```
hadoop jar $HADOOP_HOME/hadoop-test-*.jar nnbench –operation create_write –m 50 –r 50 –blockSize 64
```

TestBigMapOutput: This Map Reduce benchmark works with very big files that cannot be split.

```
hadoop jar $HADOOP_HOME/hadoop-*test*.jar testbigmapoutput -input /user/home/Downloads -output /user/home/Downloads/temp -create 1024
```

ThreadedMapBench: A Map Reduce benchmark that compares the maps performance with many spills or collapses and maps performance with a single spill.

```
hadoop jar $HADOOP_HOME/hadoop-test-*.jar threadedmapbench
```

3.3.3. Hadoop Sort Benchmark

MapReduce concept derives from filtering and sorting operations and in this Sort benchmark we will Sort random data and boost the performance of our Hadoop Cluster to the peak level.

RandomWriter : This is a data generation script which generates random data to be used later by Sorting benchmarks as an input.

- **Expected Output:**
  - The job took seconds.
- **Command**

```
hadoop jar $HADOOP_HOME/hadoop-examples-*.jar randomwriter
random.writer.out
```

TestMapRedSort: To validate the Sort benchmark

- **Expected Output:**
  - Map output records
  - Job ended (if succeeded)

- **Command**:

```
hadoop jar $HADOOP_HOME/hadoop-test-*.jar testmapredsort -m 50 -r 5 -sortInput random.writer.out -sortOutput random.writer.out.sorted
```

3.3.4. TEST DFS IO

TestDFSIO Benchmark is used for testing I/O performance of Hadoop. DFSIO or Distributed Filesystem Input/Output writes or reads into a specified number of files and sizes. TestDFSIO is used to evaluate the performance of the throughput, by putting it on a stress test.

Stress testing (sometimes called **torture testing**) is a form of deliberately intense or thorough testing used to determine the stability of a given system or entity. It involves testing beyond normal operational capacity, often to a breaking point, in order to observe the results. (Wikipedia)

This benchmark uses a MapReduce Job to read and write files into separate map tasks, whose output is used for collecting statistics that are accumulated in the reduce tasks to produce a summary result.

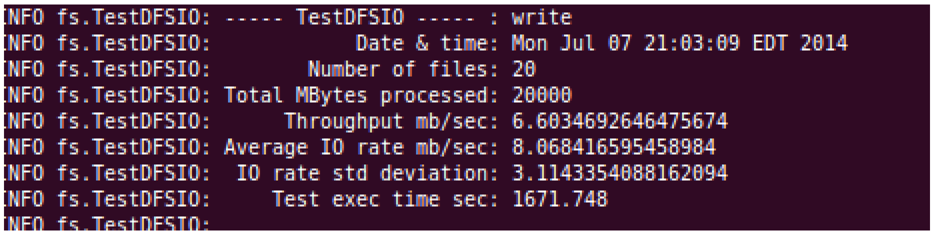
The Benchmark data then is appended to a local file named TestDFSIO_results.log and written to standard output.

- **Parameters:**
    - Operation
    - nrFiles
    - fileSize
    - result File Name
    - Buffer Size

- **Expected Output:**
    - Throughput mb/sec
    - Average IO rate mb/sec
    - IO rate stf deviation
    - Test Exec Time sec

- **Command**

```
hadoop jar $HADOOP_HOME/hadoop-*test*.jar TestDFSIO -read | -write | -clean [-nrFiles N]      [-fileSize MB] [-resFile resultFileName] [-bufferSize Bytes]
```



```
INFO fs.TestDFSIO: ----- TestDFSIO ----- : write
INFO fs.TestDFSIO:            Date & time: Mon Jul 07 21:03:09 EDT 2014
INFO fs.TestDFSIO:        Number of files: 20
INFO fs.TestDFSIO: Total MBytes processed: 20000
INFO fs.TestDFSIO:      Throughput mb/sec: 6.6034692646475674
INFO fs.TestDFSIO: Average IO rate mb/sec: 8.068416595458984
INFO fs.TestDFSIO:  IO rate std deviation: 3.1143354088162094
INFO fs.TestDFSIO:     Test exec time sec: 1671.748
INFO fs.TestDFSIO:
```

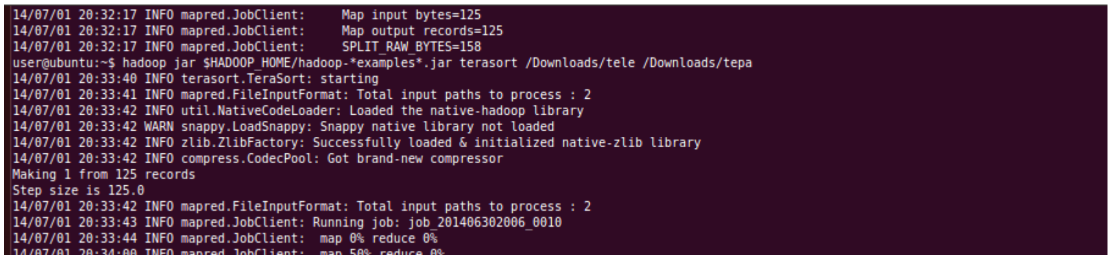
20 Files of 20 000MB processed with TestDFSIO benchmark

3.3.5. TeraSort Benchmark Suite

TeraSort is Hadoop's mainly used benchmark for sorting tests.

TeraSort Benchmark is used to test both, MapReduce and HDFS by sorting some amount of data as quickly as possible in order to measure the capabilities of distributing and mapreducing files in cluster. This benchmark comes into hand when comparing different clusters for their performance.

This benchmark consists of an Input Generator called **TeraGen** which generates random data, and then another benchmark **TeraSort** does the sorting. To be later validatet by Teravalidate.



```
14/07/01 20:32:17 INFO mapred.JobClient:     Map input bytes=125
14/07/01 20:32:17 INFO mapred.JobClient:     Map output records=125
14/07/01 20:32:17 INFO mapred.JobClient:     SPLIT_RAW_BYTES=158
user@ubuntu:~$ hadoop jar $HADOOP_HOME/hadoop-*examples*.jar terasort /Downloads/tele /Downloads/tepa
14/07/01 20:33:40 INFO terasort.TeraSort: starting
14/07/01 20:33:41 INFO mapred.FileInputFormat: Total input paths to process : 2
14/07/01 20:33:42 INFO util.NativeCodeLoader: Loaded the native-hadoop library
14/07/01 20:33:42 WARN snappy.LoadSnappy: Snappy native library not loaded
14/07/01 20:33:42 INFO zlib.ZlibFactory: Successfully loaded & initialized native-zlib library
14/07/01 20:33:42 INFO compress.CodecPool: Got brand-new compressor
Making 1 from 125 records
Step size is 125.0
14/07/01 20:33:42 INFO mapred.FileInputFormat: Total input paths to process : 2
14/07/01 20:33:43 INFO mapred.JobClient: Running job: job_201406302006_0010
14/07/01 20:33:44 INFO mapred.JobClient:  map 0% reduce 0%
14/07/01 20:34:00 INFO mapred.JobClient:  map 50% reduce 0%
```

- **Parameters:**
  - Number of 100 byte rows
- **Expected Output:**
  - Task finished / Done
- **Commands**

```
hadoop jar $HADOOP_HOME/hadoop-*examples*.jar teragen <number
of 100-byte rows> <output dir>


hadoop jar $HADOOP_HOME/hadoop-*examples*.jar terasort <input
dir> <output dir>
```

When sort was done, **TeraValidate** is run to ensure the data was sorted correctly.

```
hadoop   jar   $HADOOP_HOME/hadoop-*examples*.jar   teravalidate
<output dir> <terasort-validate dir>
```

When running terasort, there will be no output details when job is finished. Instead you can use the command of

hadoop job –history all /directory.

The benchmark history contains very useful and detailed information for the test

```
Hadoop job: 0016_1404952474849_user
=====================================
Job tracker host name: job
job tracker start time: Wed May 19 22:24:51 EDT 1976
User: user
JobName: TeraSort
JobConf:                      hdfs://localhost:9000/tmp/hadoop-
user/mapred/staging/user/.staging/job_201407091526_0016/job.xml
Submitted At: 9-Jul-2014 20:34:34
Launched At: 9-Jul-2014 20:34:35 (0sec)
Finished At: 9-Jul-2014 20:36:21 (1mins, 46sec)
Status: SUCCESS
```

It even shows analysis of the tests based on the tasks.

```
Analysis
=========

Time taken by best performing map task task_201407091526_0016_m_000001: 30sec
Average time taken by map tasks: 36sec
Worse performing map tasks:
TaskId                 Timetaken
task_201407091526_0016_m_000003 43sec
task_201407091526_0016_m_000002 42sec
task_201407091526_0016_m_000000 31sec
task_201407091526_0016_m_000001 30sec
The last map task task_201407091526_0016_m_000003 finished at (relative to the Job
launch time): 9-Jul-2014 20:35:56 (1mins, 20sec)
```

Apache Hadoop Terasort benchmark new world record set by MapR Technologies on a 1003-node cluster running on Google Compute Engine on the Google Cloud Platform. (Mapr)

3.3.6. GridMix

GridMix is a Hadoop benchmark suite that approaches most to a real cluster. It imitates the real life workload problems and tests the cluster for its performance. The input data in GridMix normally is random format, so it tests the performance of the cluster on different data format.

GridMix is configurable using its parameters.

```
hadoop jar <gridmix-jar>
org.apache.hadoop.mapred.gridmix.Gridmix \   [-generate
<size>] [-users <users-list>] <iopath> <trace>
```

3.3.7. MalStone

MalStone is Hadoop benchmark used for data intensive computing that uses records generated by MalGen. The records are a simulation of log files, where MalStone adapts a schema to common log file format containing EventID,Time Stamp, Site ID, Compromise flag, Entity ID.

Shortly, MalStone is a benchmark used on clusters whose big data comes as result of log files from wide range of applications.

- **Parameters**
    - Number of records

- **Command**

    To generate the input : python malgen.py

bin/hadoop jar ${HOME}/malstone_0.8.0.jar com.opendatagroup.malstone.hadoop. MalStoneB \ testdata malstoneB_output -r 199 2> ${HOME}/malstoneB_run.txt

i.e.

$ python malgen.py -O /home/user/testdata/ -t 0 1000000 10000000 25

We call MalStone stylized since we do not argue that this is a useful or effective algorithm for finding compromised sites. Rather, we point out that if the log data is so large that it requires large numbers of disks to manage it, then computing something as simple as this ratio can be computationally challenging. (Collin, Grossman, & Seidman, 2009)

3.3.8. HiBench Benchmark Suite

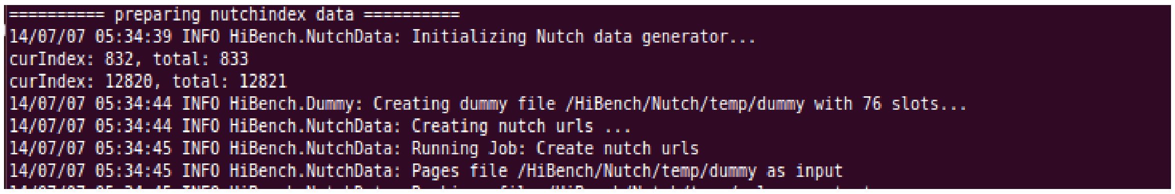
HiBench Benchmark is a suite of Hadoop testing programs that help evaluate the performance of the cluster in many aspects. Using HiBench, we can determine the speed, bandwidth, throughput, resource utilization etc. HiBench benchmark performs 9 different types of tests.

The recent versions of HiBench, generate the input data themselves, so no need to run any additional scripts nor provide input data.

HiBench suite is a more realistic and comprehensive benchmark suite for Hadoop, including not only synthetic micro-benchmarks, but also real-world Hadoop applications representative of a wider range of large-scale data analysis (e.g., search indexing and machine learning)

(Shengsheng, Jie, Jinquan, & Tao)

The benchmark starts by preparing or generating data to be used later by the benchmark.



```
========== preparing nutchindex data ==========
14/07/07 05:34:39 INFO HiBench.NutchData: Initializing Nutch data generator...
curIndex: 832, total: 833
curIndex: 12820, total: 12821
14/07/07 05:34:44 INFO HiBench.Dummy: Creating dummy file /HiBench/Nutch/temp/dummy with 76 slots...
14/07/07 05:34:44 INFO HiBench.NutchData: Creating nutch urls ...
14/07/07 05:34:45 INFO HiBench.NutchData: Running Job: Create nutch urls
14/07/07 05:34:45 INFO HiBench.NutchData: Pages file /HiBench/Nutch/temp/dummy as input
```

Not necessarily the list of 9 benchmarks, has to run when running HiBench. It's easily configured under conf/benchmarks.lst, by allowing you to choose the list and order of benchmarks.

Command to run all HiBench scripts: HiBench/bin bash run*.sh

Running scripts separately: I.E. HiBench/bin/wordcount bash run.sh

Micro Benchmarking

Sort: The data is generated by *RandomTextWriter*, and then accepted by this benchmark which then sorts input. It's a typical MapReduce task testing cluster with a heavy load.

- **Outcome**: The time the job took to execute in seconds and Job Counter details.

- **Parameters**:
    - DataSize
    - Maps
    - Reduces

- **Command**:

HiBench/bin/sort bash run.sh

WordCount : Receives input from the RandomTextWriter generator, and counts all the words and their occurrence in the input data.

Sort and WordCount programs are representative of a large subset of real-world MapReduce jobs – one transforming data from one representation to another, and another extracting a small amount of interesting data from a large data set.
(Shengsheng, Jie, Yan, & Lan)

TeraSort: TeraSort Benchmark applied as part of HiBench. Data generated by TeraGen. (information for this benchmark described above)

- **Command**:

HiBench/bin/terasort bash run.sh

Web Search

Nutch Indexing: Used for large scale search indexing. It benchmarks the automatically generated web data, based on number of pages.

- Parameters:
    - Maps
    - Reduces
    - Pages
- Command:

HiBench/bin/nutchindexing bash run.sh

PageRank: Benchmarks web data generated by its own script, by doing iterations on the IO operations.

- Parameters:
    - Maps
    - Reduces
    - Pages
    - Number of iteration
    - Block
    - Block width

- Command:

HiBench/bin/pagerank bash run.sh

Machine Learning

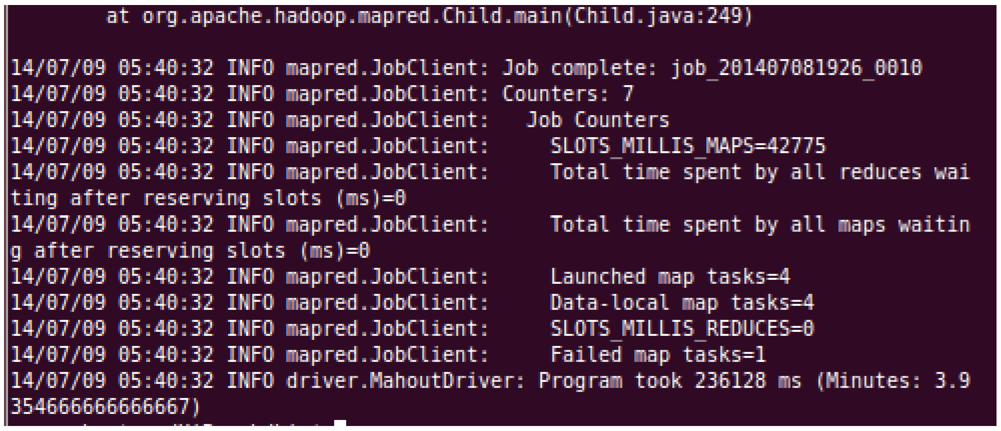
Bayesian Classification:

- Outcome:
    - Running time of benchmark

- Parameters:
    - Pages
    - Classes
    - Maps
    - Reduces
    - NGrams

- Command:

HiBench/bin/bayes bash run.sh



```
        at org.apache.hadoop.mapred.Child.main(Child.java:249)

14/07/09 05:40:32 INFO mapred.JobClient: Job complete: job_201407081926_0010
14/07/09 05:40:32 INFO mapred.JobClient: Counters: 7
14/07/09 05:40:32 INFO mapred.JobClient:   Job Counters
14/07/09 05:40:32 INFO mapred.JobClient:     SLOTS_MILLIS_MAPS=42775
14/07/09 05:40:32 INFO mapred.JobClient:     Total time spent by all reduces wai
ting after reserving slots (ms)=0
14/07/09 05:40:32 INFO mapred.JobClient:     Total time spent by all maps waitin
g after reserving slots (ms)=0
14/07/09 05:40:32 INFO mapred.JobClient:     Launched map tasks=4
14/07/09 05:40:32 INFO mapred.JobClient:     Data-local map tasks=4
14/07/09 05:40:32 INFO mapred.JobClient:     SLOTS_MILLIS_REDUCES=0
14/07/09 05:40:32 INFO mapred.JobClient:     Failed map tasks=1
14/07/09 05:40:32 INFO driver.MahoutDriver: Program took 236128 ms (Minutes: 3.9
354666666666667)
```

K-Means Clustering: Benchmark for testing K-Means algorithm for data mining and knowledge discovery. First it generates Mahout data sets as input and then runs benchmark iteration for each cluster.

- Outcome:
  - Running time of benchmark
- Parameters:
  - Number of Clusters
  - Number of Samples
  - Samples per Input file
  - Dimensions
  - Max Iteration
- Command:

```
HiBench/bin/kmeans  bash run.sh
```

Hive Bench

Hive Query Benchmark: Receives input from its own data generator. Test the cluster for its capabilities to handle large queries of Aggregation and Join. While performing map and reduce tests, it constantly measures the cumulative CPU usage in seconds.

- Parameters:
  - User Visits
  - Pages
  - Maps
  - Reduces
- Expected Output:
  - MapReduce Total cumulative CPU time
  - Total MapReduce CPU Time Spent
  - Time taken
- Command:

```
HiBench/bin/Hive  bash run.sh
```

HDFS

Enhanced DFSIO: It does the read and write operation of TestDFSIO, giving consequent outcome.

- Parameters:
  - Read number of files
  - Read file size
  - Write number of files
  - Write file size
- Outcome READ/WRITE:
  - Throughput mb/sec
  - Average IO rate mb/sec
  - IO rate std deviation
  - Test exec time sec
  - Average of Aggregated Throughput
  - Standard Deviation
  - Time spots Counted in Average

- Command

```
HiBench/bin/dfsioe bash run.sh
```

3.3.9. Puma

This benchmark suite consists of 13 benchmarks, including the existing benchmarks of Hadoop's distribution package TeraSort, Word-Count and Grep. The set of Puma benchmark suite, are intended to test MapReduce performance.

Word-Count : Can receive any type of document as input.

- Parameters:
  - reduces
- Expected Output:
  - Word
  - count
- Command

```
$ bin/hadoop jar hadoop-*-examples.jar wordcount –r <num-reduces> <input-dir> <output-dir>
```

Inverted-Index :

- Parameters:
  - Maps
  - reduces
- Expected Output:
  - Word
  - Doc id
- Command

$ bin/hadoop jar hadoop-*-examples.jar invertedindex –m<num-maps> -r <num-reduces> <input-dir> <output-dir>

Term-Vector:

- Parameters:
  - Maps
  - reduces
- Expected Output:
  - Host
  - Term vector
- Command

$ bin/hadoop jar hadoop-*-examples.jar termvectorperhost –m<num-maps> -r <num- reduces> <input-dir> <output-dir>

Self-Join

- Parameters:
  - Maps
  - reduces
- Command

$ bin/hadoop jar hadoop-*-examples.jar selfjoin –m<num-maps> -r <num-reduces> <input- dir> <output-dir>

Adjacency-List

- Parameters:
  - Maps
  - reduces
- Command

$ bin/hadoop jar hadoop-*-examples.jar adjlist –m<num-maps> -r <num-reduces> <input- dir> <output-dir>

K-Means

- Parameters:
    - Maps
    - reduces
- Command

```
$ bin/hadoop jar hadoop-*-examples.jar kmeans_itertxt_hr –m <num-maps> -r <num- reduces> <input-dir> <output-dir>
```

Classification:

- Parameters:
- Expected Output:
- Command

```
bin/hadoop jar hadoop-*-examples.jar classification –m <num-maps> -r <num-reduces> <input-dir> <output-dir>
```

Histogram-Movies: Used for data analysis, using the logic of average rating of movies, where in the mapping phase the average ratings are computed and in the reduce phase the output is collected and yielded.

- Parameters:
    - Maps
    - reduces
- Expected Output:
    - Value/rating
    - Nr of movies
- Command

Histogram-Ratings:

- Parameters:
    - Maps
    - reduces
- Expected Output:
    - Rating
    - Nr of user views
- Command

```
bin/hadoop jar hadoop-*-examples.jar histogram_ratings –m <num-maps> -r <num-reduces> <input-dir> <output-dir>
```

Sequence Count: Counts consecutively the set of 3 words for each tuple.

- Parameters:
  - Maps
  - reduces
- Expected Output:
  - Word 1 word 2 word 3 / count of them
- Command

```
bin/hadoop jar  hadoop-*-examples.jar sequencecounts –m  <num-maps> –r <num-reduces> <input-dir> <output-dir>
```

Ranked-Inverted-Index: Ranked Inverted Index benchmark takes a list of words in a document and their occurrence and yields the outcome of the words sorted by frequency in inverted order.

- Parameters:
  - Word sequence
  - File name
- Expected Output:
  - Word sequence
  - Count of files
- Command

```
bin/hadoop jar   hadoop-*-examples.jar rankedinvertedindex –m   <num-maps>   –r <num-reduces> <input-dir> <output-dir>
```

Grep: Used by data analyses to search for patterns

- Command

```
bin/hadoop jar   hadoop-*-examples.jar   grep   <input-dir>   <output-dir>   <num-reduces> <regex> [<group>]
```

(Ahmad, Seyong, Mithuna, & T.N)

4. Performance evaluation of Hadoop benchmarks

4.1. Benchmark testing

In this chapter, we applied the benchmarks listed above on a Hadoop installed machine, to be able to evaluate the performance of these benchmarks. Normally, evaluating benchmarks must be done in a real distributed environment, to be able to extract the key factors when adjusting a Hadoop performance, but in this document we focused on parameters which can be applied in a single machine and yield some reasonable outcome, based on which we can conclude their performance.

The benchmarking tests were performed in a virtual machine environment, with a Single Node Hadoop Cluster installed.

Processors: 2 (Intel Core I7- 3632QM CPU 2.20 GHZ)
Memory : 4GB
JAVA VERSION: 1.7.0_51
HADOOP VERSION: 1.2.1
VM TYPE: 32 bit

The following benchmarks are intended to observe the difference of outcomes when running on different Hadoop parameters, and the way we can optimize cluster tuning to improve the performance.

4.2. Benchmark 1: Block size

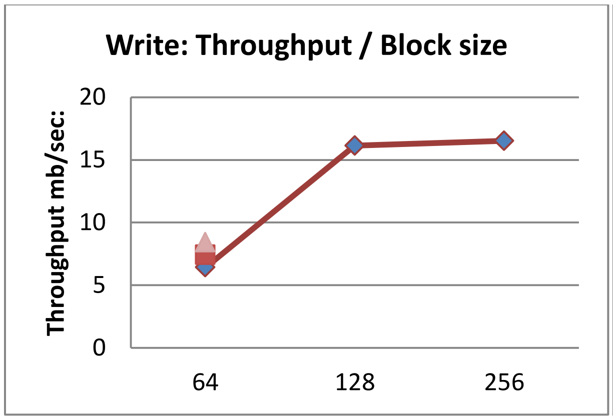
The parameter of block size that can be altered in Hadoop configuration is one of the key elements when optimizing the HDFS and Mapreduce to chunk out files into blocks, which then will be stored into datanodes. Not coincidentally, TestDFSIO was chosen as benchmark to test the block size, due to its testing capabilities of Input/Output rate and Throughput.

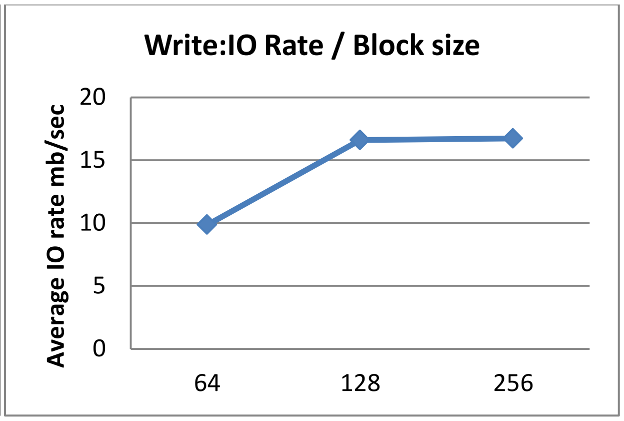
In this test, the benchmarks ran with different block and number of files, with the same total processed size in a Hadoop Cluster. The aim of this benchmark is to optimize the cluster for best performance depending on the block size, while testing it with different number of files and different file size.

The test was made using three different block size, starting with the Hadoop default block size, 64MB and continuing with most preferred block size in Hadoop clusters 128MB, and testing as well 256MB.

We used only the write operations of TestDFSIO benchmark, to make this test.

In the graph below you can see the line of throughput. The throughput with block size 128MB is higher than 64MB, whereas with 256MB takes a slight increase. The same occurs with IO rate, where 128MB and 256MB block size show a better performance in time.





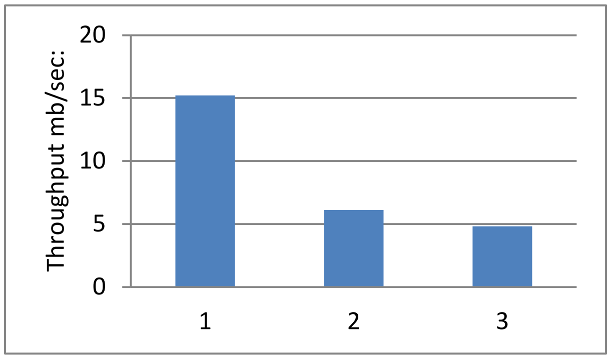
Nr Files	File Size*mb	Total MBytes processed	Throughput mb/sec:	Average IO rate mb/sec	IO rate std deviation	Test exec time sec:	Block size / MB
20	1000	20000	**6.60346926**	**8.068416595**	3.11433541	1671.748	**64**
20	1000	20000	**15.601342**	**16.03536606**	2.35043977	742.937	**128**
20	1000	20000	**17.1439984**	**17.25997162**	1.34668632	758.815	**256**

Three tests were conducted, and as a result we can conclude that when using block size of 64MB, the throughput and I/O rate will be lower than with 128MB.

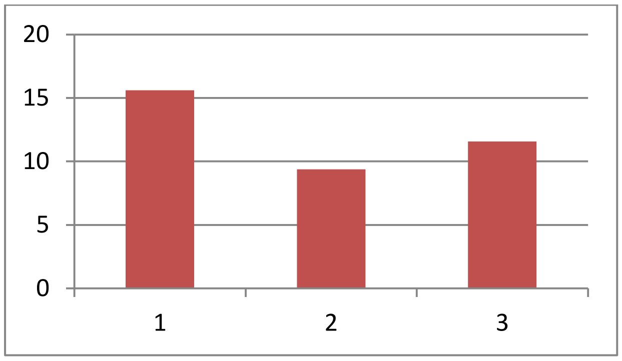
4.3. Benchmark 2: Replication Factor

When HDFS stores its blocks in the datanodes, it replicates them into different nodes and racks in order to provide reliability and fault tolerance to the clients.

By default, the blocks are replicated into 3 different nodes. In this test, we will examine the outcome with different replications and see what happens.



Write: Throughput



Write: IO rate

The tests were conducted using TestDFSIO benchmark, and we can the result in the graphs.

OPERATION: WRITE

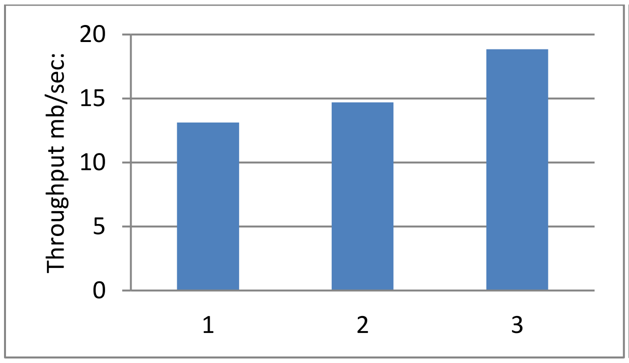
Nr of Files	File Size*mb	Total MB process.	Throughput mb/sec:	Average IO rate mb/sec	IO rate std deviation	Test exec time sec:	replication
30	500	15000	15.2172876	15.5987920	2.29424213	632.973	1
30	500	15000	6.10441231	9.38677406	4.87247412	1673.214	2
30	500	15000	4.81853710	11.5737876	4.31595184	1737.307	3

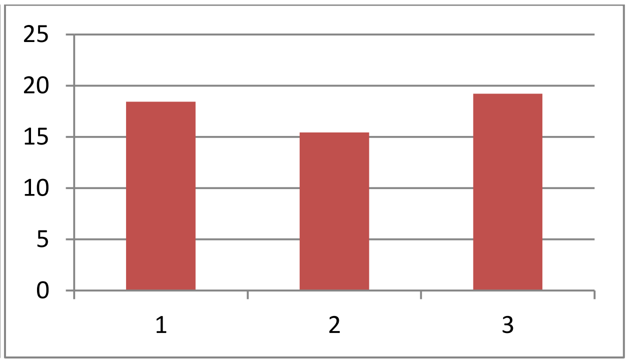
Throughput has a significant difference when replicating blocks into more than 1 node, due to the transfer and write time of pipeline between nodes.

IO rate as well, has a higher speed with 1 replication, but the difference isn't significant as in Throughput.

We performed the same benchmark, on Read operation. The throughput in read operation is increasing with higher number of replications.

OPERATION: READ





Nr of Files	File Size	Total MB processed	Throughput mb/sec:	Average IO rate mb/sec	IO rate std deviation	Test exec time sec:	replication
30	500	15000	13.115614	18.4255561	4.9526637	985.927	1
30	500	15000	14.698173	15.4223184	3.18655343	672.85	2
30	500	15000	18.838351	19.2239780	2.59074374	528.509	3

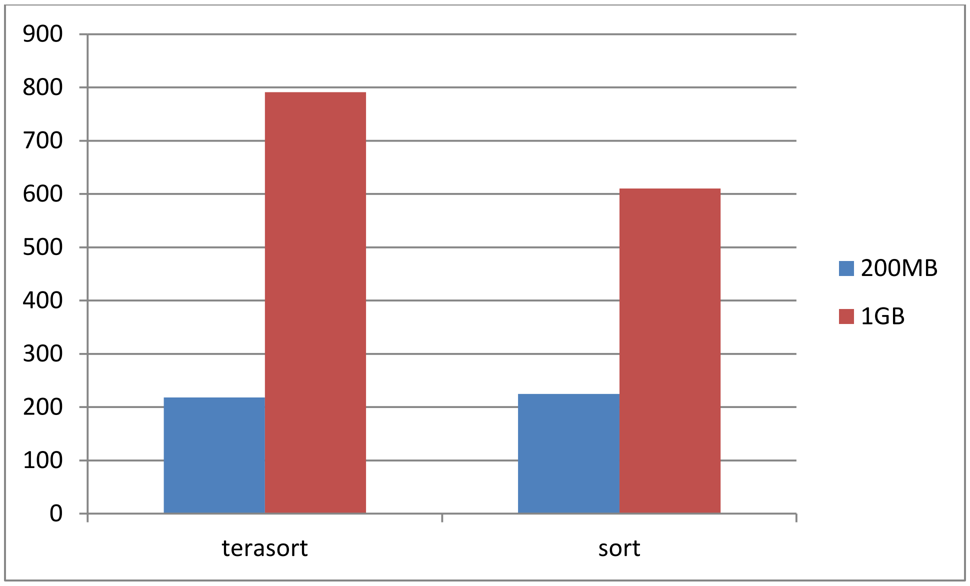
4.4. Benchmark 3: Sorting Benchmarks

4.4.1. Sorting benchmark : Terasort and HiBench Sort

Sorting benchmarks, take as input randomly generated data and the challenge of this benchmark is to sort it as quickly as possible. In this test, we used Terasort and Sort (from HiBench Suite), to test the sorting time.

We used the same number of maps and reduces in both tests, and tested them with two different filesizes.

In the graph below, you can see on the Y axis the number of seconds it took to generate sort and validate the data for the benchmarks.

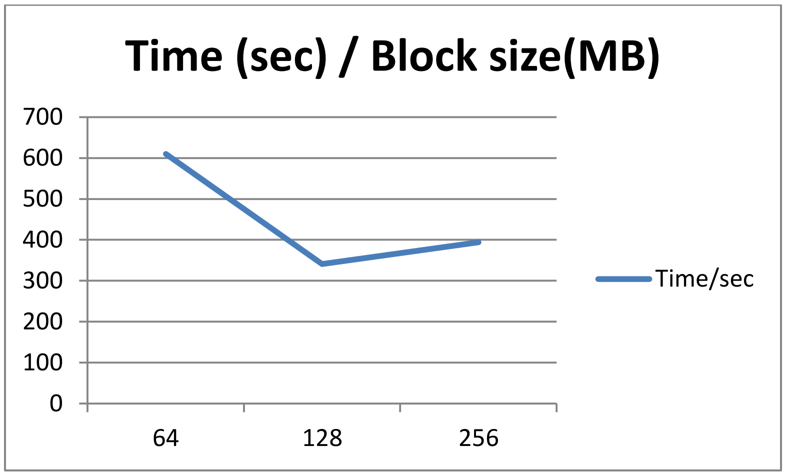


Terasort sorted the 200MB file few seconds earlier than Sort, whereas Sort benchmark sorted the 1GB file 181seconds in advance. In fact the sorting process was almost the same of terasort and sort benchmarks, the phase which delayed Terasort was the Teragen preparing the input and Sort benchmark validates its content as it sorts it, whereas Terasort has a separate task to validate content.

	200MB	**1GB**
TeraGen	55	286
TeraSort	106	450
TeraValidate	57	55
	218	**791**
Preparing Sort data	62	161
Sorting	163	449
	225	**610**

4.4.2. Sorting block size

In this test, we tried the Sorting benchmark HiBench Sort, with different block size. Sorting benchmark performs many read and write operations, so changing block size must have an impact on its outcomes.



We used a filesize of 1GB and sorted it with block size of 64MB, 128MB and 256MB, in order to find the optimum block size for sorting in a quicker way.

In the graph above, you can see that the minimum time it took to sort the file, was with 128MB, which not coincidentally is the most common block size used in Hadoop. Apparently the default block size of Hadoop, 64MB gave the slowest output and therefore, when adjusting your cluster parameters this parameter should be taken into consideration if your cluster is going to perform sorting computations.

Filesize	Time/sec	Blocksize/MB
1GB	610	64
1GB	341	128
1GB	394	256

5. Conclusion

Benchmarking Distributed Filesystem's is not a simple process, and in order to make a proper benchmark many factors should be taken into consideration.

In this document, with the benchmarks we tried to show, how by tuning few parameters we can increase the performance of the HDFS Filesystem and Mapreduce. We used Hadoop's main parameters, to be able to show a significant change on the outcomes of the tests.

As a conclusion, benchmarking depends on the output parameters you want to estimate. If you already have a cluster running and want to improve it, than you can decide on what parameters to benchmark, whereas in the case when you have a new cluster, the entire benchmarking suite such as HiBench or Puma can be suitable to find the bottlenecks and peak points of your cluster. If we want instead to see the mapreduce installation, Terasort benchmark would be suitable, or the HDFS benchmark suite for stress-testing the daemonds of Hadoop. NNBench, is the benchmark intended to test the most critical part of Hadoop, and should take place in all benchmarking tests when evaluating the performance of your cluster.

Based on our mechanism of testing, we can conclude that replication factors affects most the Input/Output write operations, as tested with TestDFSIO benchmark, whereas it has different effect on read operations of Input/Output. Block size should be taken into consideration when performing sorting computations..

Reference

Ahmad, F., Seyong, L., Mithuna, T., & T.N, V. (n.d.). *PUMA: Purdue MapReduce Benchmarks Suite.* Retrieved from http://docs.lib.purdue.edu/cgi/viewcontent.cgi?article=1438&context=ecetr

Apache. (n.d.). Retrieved from http://wiki.apache.org/hadoop/TaskTracker

Collin, B., Grossman, R., & Seidman, J. (2009). *MalStone: A Benchmark for Data Intensive Computing.*

Hadoop Performance Testing, Technologies, Impetus. (n.d.).

Horton Works. (n.d.). Retrieved from Horton Works Business Value of Hadoop: http://hortonworks.com/wp-content/uploads/2014/05/Hortonworks.BusinessValueofHadoop.v1.0.pdf

Horton Works. (n.d.). Retrieved from Horton Works: http://hortonworks.com/hadoop/yarn/

Karloff, H., Siddharth, S., & Vassilvitskii‡, S. (n.d.). A Model of Computation for MapReduce.

Mapr. (n.d.). Retrieved from http://www.mapr.com/resources/videos/mapr-terasort-record

Moody , C., & Jobs, W. (n.d.). Overcoming Networking Bottlenecks of Hadoop MapReduce.

Obaidat, M. S., & Misra, S. (2011). *Cooperative Networking.* Wiley.

Russom, P. (2013). *Integrating Hadoop into Business Intelligence and Data Warehousing.*

Shengsheng, H., Jie, H., Jinquan, D., & Tao, X. (n.d.). *The HiBench Benchmark Suite: Characterization of the MapReduce-Based Data Analysis.*

Shengsheng, H., Jie, H., Yan, L., & Lan, Y. (n.d.). *HiBench: A Representative and Comprehensive Hadoop Benchmark Suite.*

Shumin, G. (2013). *Hadoop Operations and Cluster Management Cookbook.* Packt Publishing.

Tech Dynamics. (n.d.). Retrieved from http://www.tech-dynamics.com/solution-overview/big-data

Vishwas, C., & Shweta, S. (2013). Knowledge Management in Cloud Using Hadoop.

Wikipedia. (n.d.). Retrieved from http://en.wikipedia.org/wiki/Stress_testing

Bibliography

Ahmad, F., Seyong, L., Mithuna, T., & T.N, V. (n.d.). *PUMA: Purdue MapReduce Benchmarks Suite.* Retrieved from http://docs.lib.purdue.edu/cgi/viewcontent.cgi?article=1438&context=ecetr

Apache. (n.d.). Retrieved from http://wiki.apache.org/hadoop/TaskTracker

Collin, B., Grossman, R., & Seidman, J. (2009). *MalStone: A Benchmark for Data Intensive Computing.*

Hadoop Performance Testing, Technologies, Impetus. (n.d.).

Horton Works. (n.d.). Retrieved from Horton Works Business Value of Hadoop: http://hortonworks.com/wp-content/uploads/2014/05/Hortonworks.BusinessValueofHadoop.v1.0.pdf

Horton Works. (n.d.). Retrieved from Horton Works: http://hortonworks.com/hadoop/yarn/

Karloff, H., Siddharth, S., & Vassilvitskii‡, S. (n.d.). A Model of Computation for MapReduce.

Mapr. (n.d.). Retrieved from http://www.mapr.com/resources/videos/mapr-terasort-record

Moody , C., & Jobs, W. (n.d.). Overcoming Networking Bottlenecks of Hadoop MapReduce.

Obaidat, M. S., & Misra, S. (2011). *Cooperative Networking.* Wiley.

Russom, P. (2013). *Integrating Hadoop into Business Intelligence and Data Warehousing.*

Shengsheng, H., Jie, H., Jinquan, D., & Tao, X. (n.d.). *The HiBench Benchmark Suite: Characterization of the MapReduce-Based Data Analysis.*

Shengsheng, H., Jie, H., Yan, L., & Lan, Y. (n.d.). *HiBench: A Representative and Comprehensive Hadoop Benchmark Suite.*

Shumin, G. (2013). *Hadoop Operations and Cluster Management Cookbook.* Packt Publishing.

Tech Dynamics. (n.d.). Retrieved from http://www.tech-dynamics.com/solution-overview/big-data

Vishwas, C., & Shweta, S. (2013). Knowledge Management in Cloud Using Hadoop.

Wikipedia. (n.d.). Retrieved from http://en.wikipedia.org/wiki/Stress_testing