# OS-specific tuning and benchmarking of Hadoop clusters

Research Report
by
Viktor Balogh

MSc in Cloud Computing
Cork Institute of Technology
Supervisor: Aisling O'Driscoll

14/12/2014

## Abstract

The growing amount of information sets challenges for companies so for Big Data processing emerging technologies like Hadoop will become more emphasized. Companies are pushing towards a more efficient resource usage. As a result of this there is an increasing need for performance tuning of such distributed processing systems. As in case of Hadoop chances are that we are speaking of hundreds of nodes, so even just a slight improvement in efficiency on a single node could result in a significant overall gain within the cluster. Hadoop consists of nodes based on the Linux kernel, yet there is very little information about kernel enhancements related to the Hadoop framework. In my research I am searching for the Linux kernel related aspects of Hadoop cluster tuning, and their gains with regards to the overall MapReduce performance. All the tests have been conducted with the same unchanged hardware configuration, and the aim was to find the optimal OS and kernel settings and to get exact figures about the effects of each of these tweaks.

## Table of Contents

# Chapter 1) Introduction

The use of the Hadoop framework gets more and more widespread for processing large amount of data. The aim of this research is to provide empirical data about the effects of tuning the OS related variables under the Hadoop stack; especially kernel related parameters. As the very first step I had to acquire the hardware for the project. I intend to do further Hadoop projects on the same hardware so this research project is the first of a series of tests.

Accordingly I had to set up the whole environment from scratch so a good amount of time has been spent on designing the architecture and scripting the automation framework. I have come up with a stateless cluster idea where the local disks only serve as the HDFS backend but all the OS related data is kept centrally as a read-only master image. This ensures that all OS settings are identical across the nodes. This master image can be installed by the enclosed kickstart file so it guarantees easy reproduction of my proposed architecture.

Furthermore I have written a testing framework where the job scheduling and performance data collection is fully automated. After running the tests the corresponding performance graphs are being generated from the binary performance metrics. Every test is uniqely identified by a timestamp and a runtime comment, this ensures traceability.

Once the environment is deployed and the scripting framework is developed I start conducting basic functionality tests to make sure everything is working as expected. After that I will measure the throughput of the initial system; this will serve as a baseline and all further tests will be compared to this baseline throughput. I plan to conduct tests with the TeraTest benchmark; this consists of the TeraGen, TeraSort and TeraValidate jobs. The first job will generate a dataset of the specified size, the second job will sort the dataset and the third job is a read intensive process which confirms that the dataset is indeed in the right order.

Note that my research focuses on the performance tuning of a single job. In a real-world Hadoop environment several jobs are running parallel so that could be a different scenario. As this is my first performance research on Hadoop I aimed to tune the performance of single running jobs. In most cases a typical Hadoop workload is a so called WORM workload (write-once-read-many) so it is read-intensive. Due to this I will mostly focus on the TeraSort results though I will also record the metrics for all the other tests. Depending on the number of running jobs it can be near-sequential but the more jobs are running the more this sequentiality turns to random access. As we are speaking of longer-running jobs started from some web frontend; latency is not a concern at all, we have to strive for more throughput instead.

As this is just a research project in a test environment and I don't intend to use it for production purposes; I felt free to experiment with the latest available operating system on top of the latest available kernel. For the same reason I did not set up high availability for the master services though I didn't even change the replication ratio for the HDFS blocks as that would have invalited my results.

After reading various materials on the topic it became clear that Hadoop performance tuning is a daunting task and involves many test cycles to find the right parameters. Furthermore there are various workload types so the results of this research might not be applicable if the characteristic of the planned workload significantly differs from the one of the TeraTest benchmark. After reading the available materials I hope to get at least 2-3% performance improvement with this tuning attempt.

## i. Thesis Contribution

I intend to enclose the following deliverables as part of this research and hope that these prove to be useful for other researchers:

- Research report (this thesis)
- Sar binary data about performance stats from each tests
- Performance graphs based on sar data in HTML and PDF format
- Hadoop logs from each tests
- Kickstart file for the Golden image
- Hadoop configuration files
- Offline graph generation scripts
- RUNBOOK.txt (i.e. research diary)

As a sidenote I have used extensive keyed SSH over the course of the research. For security reasons even if this is a test environment I decided to truncate every private key from all the materials submitted.

## ii. Thesis Outline

In the following I will summarize the remaining chapters of this document. In chapter two I take an overview of the available written references on this topic. As the TeraTest in general is a disk intensive workload, I will focus on the storage layer in my research. For this I have found some good references from other researchers which I plan to underpin with my empirical tests.

In chapter three I outline the design of the proposed architecture. Chapter four details the implementation steps and all the difficulties I encountered by setting up the system. Chapter five summarizes the key performance metrics I've collected during the tests and in the end of that chapter I present the results of the research. Finally in chapter six I draw the consequences and also suggest directions for future improvements.

# Chapter 2) Literature review

The Big Data related area of computing is quite popular nowadays, so it is very easy to find research papers in this topic. Some have tested the hardware related tuning possibilities while others have conducted researches regarding the optimization of the Mapreduce framework. Though I've found that not much has been mentioned about the layer in between these: about the Linux kernel.

My reference list can be divided into multiple groups based on the environment used. In [2], [3] and [8] clear hardware tests have been conducted while [1], [7] and [17] have used virtualization in their tests. Furthermore there are papers like [13] where a comparison of physical vs. virtual deployment has been given; or [10] where three hypervisors have been compared.

Papers [6], [9] and [11] are heavily theory based; they give mathematical formulas to help predict the execution time of a job. The latter solely focuses on MapReduce and gives a performance model to better understand the bottlenecks. Although there are compute-intensive researches, most of the researchers concluded that the majority of Hadoop jobs are rather I/O intensive. This means that in my research I should mostly focus on the I/O aspects while tuning the Linux kernel. Xie et al [18] give an interesting proposal for heterogeneous Hadoop deployments, where the data distribution should be aligned according to the compute capacity of the workers. This could be leveraged for heterogeneous deployments, but I am testing on identical nodes, so that paper can be ignored for my research.

The research titled "Evaluation on the Performance Fluctuation of Hadoop Jobs in the Cloud" [1] is written about the performance fluctuation of the virtualized framework. Yet it is useful for my research, as it gives a detailed description on what tools have been used for benchmarking. The main focus was the performance fluctuation itself, rather than tuning of the Hadoop framework. They were running performance tests every 4 hours in a public cloud for over three months and compared the results to the metrics from a private cloud. They found that the fluctuation is small and is mostly noticeable with irregular long running jobs.

The researches [2], [12], [16], and [18] mostly keep the focus on HDFS and provide I/O related tips for tuning, so they are most helpful for my research. Some papers are specifically written about Hadoop used in conjunction with another software, like [3] is a performance evaluation of Hadoop with MongoDB, and [12] is an older paper benchmarking a WebGIS solution running on top of Hadoop.

In [4] Ding et al write about the differences between Hadoop and Hadoop Streaming. Hadoop Streaming is a utility included in Hadoop and enables one to have any executable as the mapper and reducer. However this flexibility does not come without any drawback, they find that the pipe operation means a major bottleneck for I/O intensive applications. On the other hand they even observed some performance improvement by CPU-bound tasks; this was due to the mapper and reducer being written in a lower level code compared to the default Java.

Another aspect of the tests is the size of the testing environment. Dede et al [3] had the opportunity to test the Hadoop framework on a Cray XE6 supercomputer; they've built a rather complex Hadoop-MongoDB environment on top of that. They have deployed HDFS with the default settings and experimented with changing the workload. They did failover tests and even conducted scalability tests with different number of cores in the cluster.

Similar scalability tests have been conducted by Shafer, Rixner & Cox in [16] where among others they found that oversubscription of map and reduce tasks may help eliminate the idle CPU time while the CPU waits for I/O. They claim their paper to be the first to clearly describe the interaction between Hadoop and the underlying storage stack. It gives a deep dive about this interaction, which will surely help to identify the right kernel parameters for my project.

Furthermore there is a research report by Joshi [8] which gives a high level overview about the possible tunables in Hadoop. It is quite a short paper with just a single reference, and does not provide graphs, but there are some percentage values mentioned in the body of the report. However this is the only paper I've found which mentions best practices regarding BIOS and kernel settings, so I feel a bit of a gap in this area, which gives me motivation to proceed with my research.

# Chapter 3) System Design

At the planning stage of the project the hardware still wasn't available. I have considered several possibilities e.g. having it all running in a public cloud. As this projects involves kernel tuning and changing OS settings, that would have meant dealing with a public IaaS offering; though most of such offerings do not allow one to tinker with the kernel. Another possibility was to get some sponsors for the research that could have provided me with the required hardware. In the end I decided to buy my own hardware for the tests, with the intention that I can repurpose it later for other projects.

## i. Hardware

Here is a list of hardware components I've used for the testing:

- 1x Cisco Catalyst 2970 Gigabit Ethernet switch
- 4x nodes of Hewlett Packard ML310e Gen8 each with:
  - 1x Intel Xeon E3-1220 v2 (8M cache, 3.1GHz)
  - 2x 8GB DDR3 PC3-12800 ECC RAM
  - 1x 3TByte Seagate ST3000DM001 Barracuda (SATA III, 6GB/s, 7200rpm)
  - 2x Broadcom NetXtreme Gen I (TG3, integrated)
- 1x Raspberry Pi Model A (256MB RAM) for "orchestrator server"

As a rule of thumb a real-world Hadoop node would have at least the same number of disks as CPU cores. The aim of this research was to optimize the cluster for the TeraTest benchmark suite, which is clearly an I/O bound workload. Alas I was on a tight budget which restricted me to get more disks for the test environment, and as it turned out this meant a significant bottleneck for the tests. Even though I have expected the disk to be the bottleneck, I was precautious with the hardware procurement and calculated with additional gigabit interfaces for the nodes. The idea behind that was to be able to do trunking if required.

## ii. Operating system

Over the years I have gained quite a lot of experience with the Red Hat distribution so CentOS seemed to be a straightforward choice for the operating system. The CentOS distribution is basically the same as its upstream distribution (Red Hat Enterprise Linux or RHEL) though it does not include the branding elements so it's completely free. At the time I started to deploy the systems (summer of 2014) RHEL 7 (and also CentOS 7) has just been released.

Even though it wasn't officially supported by the current Cloudera CDH, I thought I will give it a try and figure it out if I would run into any incompatibility issues. Here is a list of software components and their versions I deployed:

- CentOS 7 (7.0.1406)
- Linux Kernel tree 3.17-rc4
- Cloudera Hadoop Distribution 5.1.0
- Oracle JDK 7 Update 60

For CentOS 7.0.1406 the default inbox kernel is of the version 3.10.0-123.el7.x86_64 which was already more than a year old. By the time I've finished the baseline tests, the current mainline kernel was the 3.17-rc4 so I've used it for compiling a custom kernel. I have extracted the configuration from the distro kernel and omitted the unneeded modules so the compilation did not take unnecessarily long. I write more details on this process in chapter four.

## iii. Hadoop distribution

The Apache Hadoop is a software framework which consists of various components. To ease the deployment and avoid configuration issues there are some vendors which offer support and packaged versions of the known-to-be-stable component versions, Cloudera is one of them. The reason I have chosen Cloudera Hadoop is that it is a well-maintained and fairly stable distribution with packages available for the Red Hat line. It also has one of the biggest install-base among all the “boxed” Hadoop distributions. To keep me up-to-date regarding the current installation and performance issues I have also subscribed to the cdh-user Google groups.

For the Hadoop distribution I went with the most current CDH version which was 5.1.0 at the time of the initial deployment. According to the CDH 5.1.0 documentation the minimum supported JVM version is 1.7.0_55 [19], so I've chosen the most current version which was available back then to avoid any future java-related frustrations.

A Hadoop cluster consists of various services. As this project was intended to be a performance test, I only wanted to deploy the most essential services and did not bother with the high availability of these components. This means that my test environment does not have any secondary NameNode, HA HDFS or HA JobTracker configured. However I've left the HDFS replication factor on its default setting (3) as I wanted to simulate a real world workload.

Another question was whether the global services like NameNode and ResourceManager (i.e. JobTracker in MRv1) should be kept external to the nodes to be tuned. In a production deployment, these global services are often kept separated from the "worker nodes" as they have slightly different resource requirements. As I did not have enough hardware lying around I've decided to build a self-containing test environment with all these global services scattered – and possibly load-balanced – across the worker nodes. Here is a list of services and their placement on the nodes:

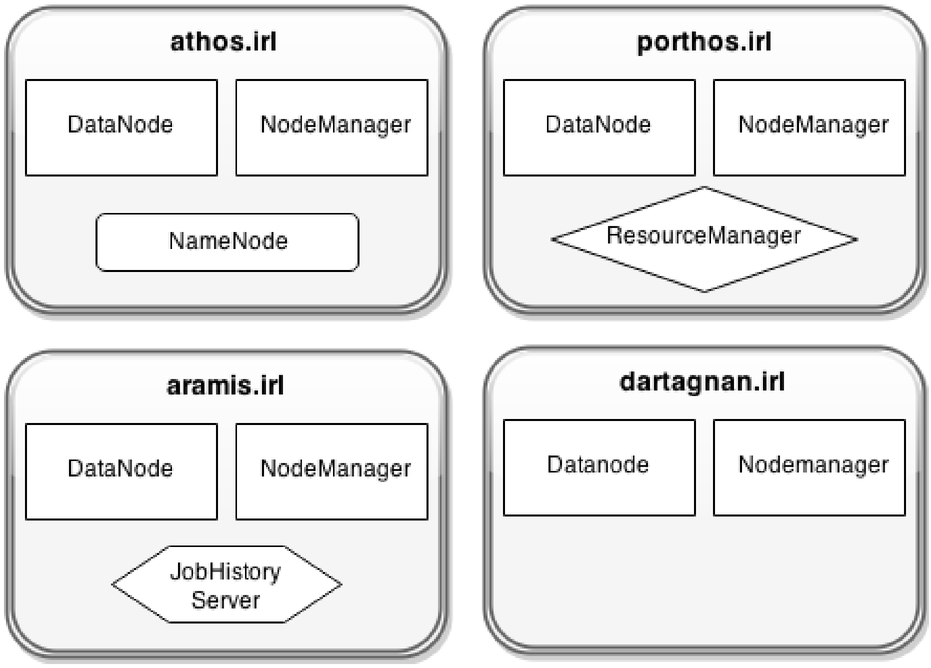


Figure 1: Hadoop services and their placement across the nodes

As it can be seen in the above figure, the standard worker processes are the DataNode and the NodeManager services; these can be found on each worker node. On top of that there are global services like NameNode (HDFS), ResourceManager (Yarn) and the JobHistory server. I also wanted to deploy a Hue server on the fourth node, which provides a web based GUI for managing Hadoop. However I've faced difficulties deploying it due to version incompatibilities, see the next chapter for more on that. So this has left the fourth node with less overhead.

## iv. Stateless nodes

In order to provide a consistent configuration across the cluster and to avoid human errors as much as possible, I came up with the idea of a stateless cluster. The Red Hat distribution has a so called "readonly-root" feature, which is mostly used for Live CDs. With this feature one could mark part of the root filesystem as read-only, and yet creating some writeable space by loopback mounting directories from tmpfs. By marking the whole root filesystem as read-only I could create a single read-only NFS share which serves as the common root filesystem for all the nodes. Here is the structure of the root filesystem with regards to the write-enabled parts:

```
[root@athos ~]# df
Filesystem                          1K-blocks     Used  Available Use% Mounted on
192.168.1.99:/btrfs/CentOS7_Hadoop   16277504  9659200    4970944  67% /
devtmpfs                              8131824        0    8131824   0% /dev
tmpfs                                 8170720        0    8170720   0% /dev/shm
tmpfs                                 8170720     8560    8162160   1% /run
tmpfs                                 8170720        0    8170720   0% /sys/fs/cgroup
none                                  8170720      360    8170360   1%
/var/lib/stateless/writable
none                                  8170720      360    8170360   1% /var/cache/man
none                                  8170720      360    8170360   1% /var/lib/dbus
none                                  8170720      360    8170360   1% /var/lib/nfs
none                                  8170720      360    8170360   1% /tmp
none                                  8170720      360    8170360   1% /var/lib/dhclient
none                                  8170720      360    8170360   1% /var/tmp
none                                  8170720      360    8170360   1% /etc/adjtime
none                                  8170720      360    8170360   1% /etc/resolv.conf
none                                  8170720      360    8170360   1%
/var/lib/NetworkManager
none                                  8170720      360    8170360   1%
/var/lib/logrotate.status
none                                  8170720      360    8170360   1% /var/spool
none                                  8170720      360    8170360   1% /var/cache/yum
none                                  8170720      360    8170360   1% /var/lib/systemd
none                                  8170720      360    8170360   1% /etc/hadoop/conf.irl
none                                  8170720      360    8170360   1% /kernel
none                                  8170720      360    8170360   1% /etc/hostname
none                                  8170720      360    8170360   1% /etc/chrony.keys
none                                  8170720      360    8170360   1% /etc/sysconfig/network
none                                  8170720      360    8170360   1% /etc/rwtab
none                             8170720       360    8170360   1%
/etc/sysconfig/readonly-root
none                             8170720       360    8170360   1% /var/lib/dhclient
none                             8170720       360    8170360   1%
/var/lib/stateless/writable/var/lib/dhclient
/dev/sda1                     2884137136 806651608 2077469144  28% /hdfs
[root@athos ~]#
```

Note the tmpfs loopback mounts and the special kernel mounts with “none” in the first column. Those are files and folders which are made read-write through loopback mounts. There are some critical files which have to be writable. One example is the resolv.conf file which gets populated from the DHCP data; another example is the structures under /var like the yum cache. Note that the root filesystem is mounted over NFS, and as the last line tells the /hdfs filesystem containing the persistent HDFS data is mounted from the local disk.

By conducting performance tests one must make sure that the results are consistent and reproducible. Speaking of distributed computing, I wanted to plan a test environment where the configuration of the nodes is identical. I wanted to avoid human error as much as possible so I’ve chosen a centralized configuration approach. I’ve built a so called “stateless” Hadoop cluster, where the root filesystem gets mounted from the central read-only NFS server. This makes sure that every node has the same baseline and the same kernel, while all the HDFS data gets stored on the local disks persistently. The wiki of the Coda project was a significant help for the implementation of the network boot. [25]

## v. Orchestration server

Due to budget constraints I’ve used my Raspberry Pi for storing the central configuration and serving the golden image over NFS. I have considered the risk what it could impose to the project as it only has limited resources. I found that since I don’t intend to benchmark the boot process and all the tests are conducted after “warming” the NFS cache on the clients, this should not have a significant effect to the benchmark results. After the system is loaded and the cache is warm, the bottleneck will still be the local storage stack, as my results have just shown. Besides the NFS server there are various other services running on the Raspberry Pi, e.g. NTP for timekeeping or bind (DNS) as name service. Here is a list of services running on that central “orchestration” server:

```
root@raspberry:~# pstree
init─┬─cron
     ├─dbus-daemon
     ├─dhcpd
     ├─getty
     ├─2*[ifplugd]
     ├─named───3*[{named}]
     ├─ntpd
     ├─rpc.mountd
     ├─rpcbind
     ├─ssh-agent
     ├─sshd───sshd───bash───pstree
     ├─udevd───2*[udevd]
     └─xinetd
root@raspberry:~#
```

The TFTP used for booting is running from the xinetd service. I have documented all the MAC addresses in the environment and configured the DHCP server so a specific MAC would always get the same IP address. This ensures a very scalable approach: the hostname and IP address are only stored in the central repository which would make a further expansion hassle-free. For more info on adding a new node see the scalability subchapter. Finally here is a list of services which are running on the Raspberry Pi, including TCP and UDP port numbers:

```
root@raspberry:~# netstat -lntup | sort -k4r
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 192.168.1.99:53         0.0.0.0:*               LISTEN      1946/named
udp        0      0 192.168.1.99:53         0.0.0.0:*                           1946/named
udp        0      0 192.168.1.99:123        0.0.0.0:*                           2101/ntpd
udp        0      0 192.168.1.255:123       0.0.0.0:*                           2101/ntpd
tcp        0      0 127.0.0.1:953           0.0.0.0:*               LISTEN      1946/named
tcp        0      0 127.0.0.1:53            0.0.0.0:*               LISTEN      1946/named
udp        0      0 127.0.0.1:53            0.0.0.0:*                           1946/named
udp        0      0 127.0.0.1:123           0.0.0.0:*                           2101/ntpd
udp        0      0 0.0.0.0:997             0.0.0.0:*                           1670/rpcbind
udp        0      0 0.0.0.0:69              0.0.0.0:*                           2283/xinetd
udp        0      0 0.0.0.0:67              0.0.0.0:*                           2071/dhcpd
udp        0      0 0.0.0.0:57228           0.0.0.0:*                           2250/rpc.mountd
tcp        0      0 0.0.0.0:54902           0.0.0.0:*               LISTEN      2250/rpc.mountd
tcp        0      0 0.0.0.0:54446           0.0.0.0:*               LISTEN      -
tcp        0      0 0.0.0.0:50813           0.0.0.0:*               LISTEN      2250/rpc.mountd
udp        0      0 0.0.0.0:49490           0.0.0.0:*                           -
tcp        0      0 0.0.0.0:46734           0.0.0.0:*               LISTEN      2250/rpc.mountd
udp        0      0 0.0.0.0:35823           0.0.0.0:*                           2250/rpc.mountd
udp        0      0 0.0.0.0:33571           0.0.0.0:*                           2250/rpc.mountd
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN      2169/sshd
tcp        0      0 0.0.0.0:2049            0.0.0.0:*               LISTEN      -
udp        0      0 0.0.0.0:2049            0.0.0.0:*                           -
udp        0      0 0.0.0.0:123             0.0.0.0:*                           2101/ntpd
tcp        0      0 0.0.0.0:111             0.0.0.0:*               LISTEN      1670/rpcbind
udp        0      0 0.0.0.0:111             0.0.0.0:*                           1670/rpcbind
udp        0      0 0.0.0.0:11072           0.0.0.0:*                           2071/dhcpd
root@raspberry:~#
```

The DHCP daemon listens on UDP 67. During a server boot it broadcasts a DHCP request, which gets answered with the corresponding information including BOOTP configuration. After the kernel has been loaded from TFTP (UDP 69, see above) the NFS root gets mounted from the Raspberry Pi and a chroot takes place so the system will continue to run with the NFS root.

### vi. Guaranteeing reproducibility

A particularly important part of the system is the central image used for network booting the nodes. I wanted to take a 100% reproducible approach so I decided to create the “golden image” via a scripted network installation. For this I have deployed a network software repository with the CentOS and Cloudera binaries on my laptop. Then I have created a so called kickstart or “answer” file which answers all the installation questions. In this kickstart file I could also define all the required customizations including partitioning, SSH keys for key-based authentication, user profiles, custom startup scripts and the like.

This kickstart setup guarantees that I only have to put a blank USB stick to one of the hosts, boot it off the network with the install option and it will generate the very same golden image onto the plugged stick. In chapter four I will give a detailed explanation of the most important parts of the kickstart file.

### vii. Cabling

Figure 2 illustrates the physical cabling of the system I’ve built. The yellow lines are the cables leading to the iLO interface, the Raspberry got attached over an additional router so I could completely power off the Cisco switch and all the servers; yet I was still able to access the test results from the Raspberry. I know that with the SoHo router I have introduced serious latency though it was not a concern as I did not benchmark the boot: the aim of the research was to get a higher Hadoop throughput with an already warmed cache.

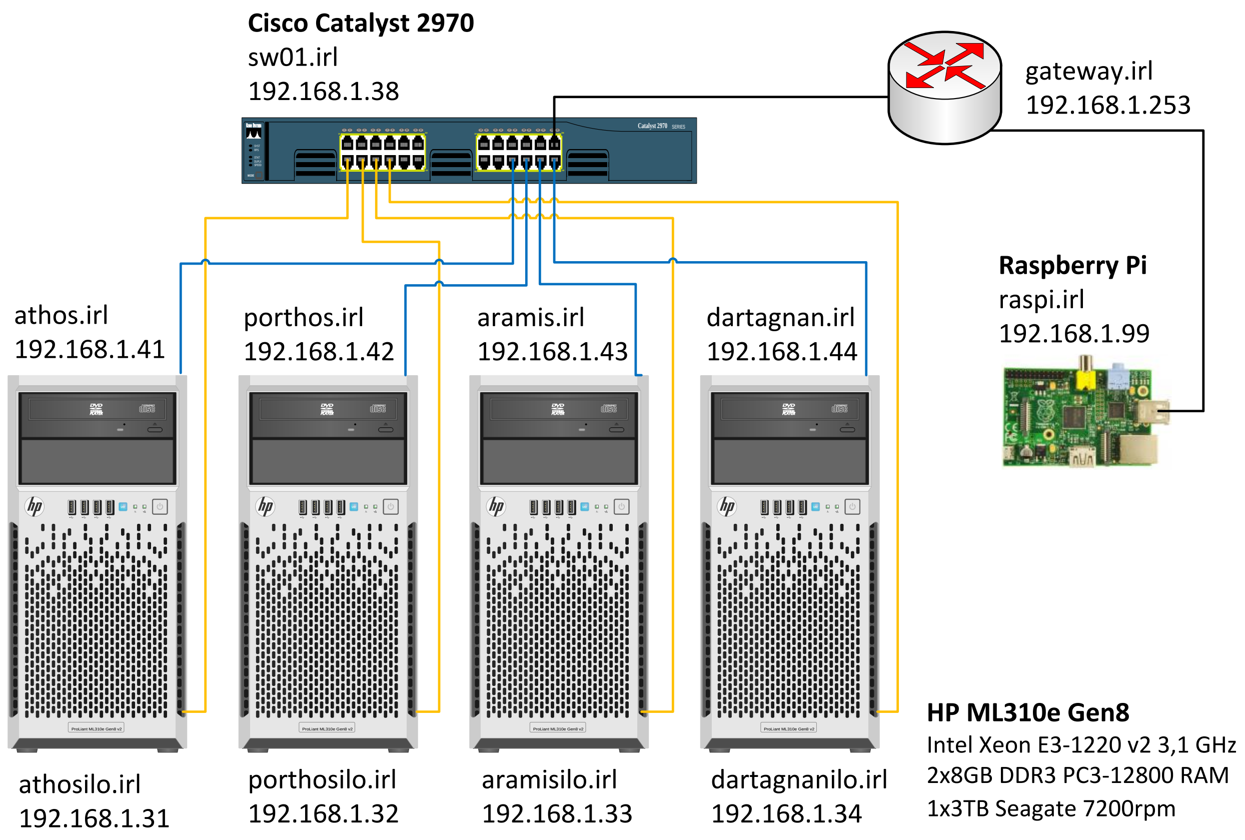


**Figure 2: Hadoop environment layout**

# Chapter 4) System Implementation

After I have concluded with the design phase and had some rough ideas on how to build up the testing environment, I had to order the hardware. It took almost two weeks until I got all the components from the various suppliers. In the following subchapter I detail the deployment of the various parts of the architecture.

## i. Install server

While I was waiting for the hardware components to get delivered, I started to set up the install server. This is basically a plain Apache HTTP server containing the packages and their metadata in XML format. Here is the structure of the install server, with only the directories listed:

```
# find repo -type d -exec ls -ld {} \+
drwxr-xr-x 1 root root     42 Aug  5 21:35 repo
drwxr-xr-x 1 root root     42 Aug  5 22:16 repo/cdh5
drwxr-xr-x 1 root root    992 Aug  2 22:50 repo/cdh5/repodata
drwxr-xr-x 1 root root     44 Aug  2 22:49 repo/cdh5/RPMS
drwxr-xr-x 1 root root   6294 Aug  2 15:46 repo/cdh5/RPMS/noarch
drwxr-xr-x 1 root root     44 Aug  2 22:49 repo/cdh5/RPMS/Oracle_JDK
drwxr-xr-x 1 root root   8040 Aug  2 15:53 repo/cdh5/RPMS/x86_64
drwxr-xr-x 1 root root      2 Jul 26 21:59 repo/centos
drwxr-xr-x 1 root root     18 Jul 26 23:34 repo/centos/7
drwxr-xr-x 1 root root     12 Jul  5 12:06 repo/centos/7/os
drwxr-xr-x 1 root root    254 Jul  7 20:06 repo/centos/7/os/x86_64
drwxr-xr-x 1 root root      8 Jul  4 16:59 repo/centos/7/os/x86_64/EFI
drwxr-xr-x 1 root root     98 Jul  4 16:59 repo/centos/7/os/x86_64/EFI/BOOT
drwxr-xr-x 1 root root     22 Jul  4 16:59 repo/centos/7/os/x86_64/EFI/BOOT/fonts
drwxr-xr-x 1 root root     52 Jul  4 16:59 repo/centos/7/os/x86_64/images
drwxr-xr-x 1 root root     56 Jul  4 16:59 repo/centos/7/os/x86_64/images/pxeboot
drwxr-xr-x 1 root root    196 Jul  6 01:02 repo/centos/7/os/x86_64/isolinux
drwxr-xr-x 1 root root     24 Jul  4 16:59 repo/centos/7/os/x86_64/LiveOS
drwxr-xr-x 1 root root 650166 Jul  4 16:59 repo/centos/7/os/x86_64/Packages
drwxr-xr-x 1 root root   1334 Jul  4 17:01 repo/centos/7/os/x86_64/repodata
drwxr-xr-x 1 root root     12 Jul  5 16:58 repo/centos/7/updates
drwxrwxr-x 1 root root     42 Jul  9 14:14 repo/centos/7/updates/x86_64
drwxrwxr-x 1 root root 101904 Aug 14 16:39 repo/centos/7/updates/x86_64/drpms
drwxrwxr-x 1 root root  52764 Aug 14 16:39 repo/centos/7/updates/x86_64/Packages
drwxrwxr-x 1 root root   4572 Aug 14 16:39 repo/centos/7/updates/x86_64/repodata
drwxr-xr-x 1 root root     34 Sep 14 19:16 repo/ks
drwxr-xr-x 1 root root    450 Sep 14 11:24 repo/ks/b64.ks.bkup
#
```

The cdh5 folder contains the Cloudera RPMs and the Oracle JDK. For the exact versions of the CDH5 components installed see Appendix A. The centos folder is the mirror of the CentOS distro packages while the ks folder contains the kickstart file. I have spent more than a month tinkering with that kickstart file so that the reproduction of the "golden image" would not need any manual intervention. For each manual configuration change on the system it had to be done in the kickstart file as well, after which the golden image had to be regenerated in order to test if it has the desired effects on the image.

## ii. Kickstart file

As I have mentioned it in the previous chapter, I have created a kickstart file with all the configuration required for creating the golden image which serves as the read-only NFS root for the worker nodes. The whole kickstart file can be found in Appendix B – Kickstart file. For troubleshooting normally the ALT+F1 -> ALT+F6 hotkeys can be used [20] but since I have worked through an SSH session I had to use the CTRL + B [NUM] combination. The kickstart file generation was a lengthy trial-and-error process which took me a bit more than a month. However the installation is fully automated now and therefore I have built a 100% reproducible setup. For this trial-and-error I have used btrfs on the Raspberry so I could efficiently utilize its snapshotting capabilities. That means, I could keep several golden images on the repository server as only the changes have been stored via filesystem tree snapshots.

The kickstart file consists of heavy scripting parts so in the following I'll explain the non-trivial parts of it. The HP ML310e has two integrated Broadcom-based network interfaces, plus I've built in two dual-port Intel NICs for future purposes as well. So I had to specify it in the kickstart file that I want the system to be booted off of the first NIC:

```
network --noipv6 --onboot=yes --bootproto dhcp --device=eth0 --hostname=goldenimage
network --noipv6 --onboot=no --device=eth1
network --noipv6 --onboot=no --device=eth2
network --noipv6 --onboot=no --device=eth3
network --noipv6 --onboot=no --device=eth4
network --noipv6 --onboot=no --device=eth5
```

Note that I've given the hostname "goldenimage", this gets changed during boot time for each node. The following section lists the services. I have turned off almost everything including the Hadoop services, I have only left the sshd and the chrony (NTP) daemon enabled. The reason I disabled the Hadoop services is that before starting them I had to double-check that the time is set correctly and I also had to set the correct hostname. So here is the kickstart service definition:

```
services --disabled="kdump,iprdump,iprinit,iprupdate,auditd,irqbalance,postfix,tuned,systemd-
readahead-collect,systemd-readahead-drop,systemd-readahead-replay,hadoop-0.20-mapreduce-
jobtracker,hadoop-0.20-mapreduce-tasktracker,hadoop-hdfs-datanode,hadoop-hdfs-namenode, rhel-
autorelabel-mark, systemd-random-seed" --enabled="sshd,chronyd"
```

The following shows the list of packages being installed. I wanted to keep the install as lightweight as possible. Gzip and Bzip were needed for experimenting with HDFS compression, sysstat provides the system activity reporter aka. sar which was used as the performance data collector facility. Note that no Hadoop packages are listed here since they are being installed from a separate network repository, in a later step.

```
%packages --nobase --ignoremissing
@core
yum
openssh-server
chrony
kernel-firmware
wget
sudo
perl
psmisc
bind-utils
nfs-utils
cachefilesd
nfstest
nfsometer
sysstat
screen
netstat
zip
unzip
bzip2
bzip2-1.0.6-12.el7.x86_64
gzip
lsof
lm_sensors
net-tools
gcc
gcc-c++
autoconf
automake
binutils
bison
flex
gettext
libtool
make
patch
pkgconfig
redhat-rpm-config
rpm-build
rpm-sign
patchutils
git
subversion
ncurses-devel
hmaccalc
zlib-devel
binutils-devel
```

```
elfutils-libelf-devel
-*firmware
-b43-openfwwf
-efibootmgr
-fcoe*
-iscsi*
%end
```

The next part describes the pre-install steps written in bash. A very important thing is that this part is not running in the chroot environment, so the log file will be lost after the installation. Yet it is a best practice to keep the whole procedure logged to a file, this is useful for live troubleshooting. In the first part the include.me file gets created with a “here document”. This makes sure that the installation will be put to a device smaller than 16 Gigabytes i.e. the thumb drive gets used. This was a security measure added by me, ensuring that the internal hard drives don’t get overwritten by the installation: this makes the persistent HDFS store really persistent. As long as a suitable thumb drive is plugged in the install script will find it and choose it as the target for the installation. The installer will create a basic filesystem layout with a single 6 Gbyte (6144 Mbytes) ext4 root filesystem.

```
%pre --log=/root/install-pre.log
#!/bin/bash

incfile=/tmp/include.me
> $incfile

# Check physical disks
MAXSIZE=16384  # in Megabytes
for disk in /sys/block/sd*
do
        dsk=$(basename $disk)

        if [[ `cat $disk/ro` -eq 1 ]];
        then
                echo "Skipping disk $dsk: READONLY"
                continue;
        fi

        if [[ $((`cat $disk/size`*512/1024/1024)) -gt ${MAXSIZE} ]];
        then
                echo "Skipping disk $dsk: larger than ${MAXSIZE} megabytes"
                continue;
        else
                echo "Using disk $dsk"
                chosen=$dsk;
                break;
        fi
done

if [[ -n "$chosen" ]];
then
        echo "zerombr" >> $incfile
        echo "bootloader --location=mbr --timeout=3 --boot-drive=$chosen --driveorder=$chosen
--append=\"nomodeset\"" >> $incfile
        echo "ignoredisk --only-use=$chosen" >> $incfile
        echo "clearpart --all --initlabel --drives=$chosen" >> $incfile
        echo "part / --fstype=ext4 --asprimary --ondisk=$chosen --size=6144 --grow" >>
$incfile
else
        echo "" > $incfile
fi
%end
```

The next part is the post-install script. This is different from the pre-install script as this gets called from within a chroot to the installed system. So its log file will be available even after the installation. I have set up keyed based SSH to the systems by creating and configuring all the required users. I have also defined here the host keys so it did not cause me any hassle with the ever changing keys between reinstalls.

```
%post --log=/root/install-post.log
#!/bin/bash

cat <<EOF > /etc/ssh/ssh_host_rsa_key.pub
ssh-rsa somekeyhere_censored
EOF
chmod 644 /etc/ssh/ssh_host_rsa_key.pub

echo "SSH config setup"
cat <<EOF > /root/.ssh/config
GSSAPIAuthentication no
ForwardAgent yes
StrictHostKeyChecking no
EOF
chmod 644 /root/.ssh/config
```

Here I amended the default profile of the root user with my preference. Note the jps alias which is a handy tool to list all the JVM processes and their status. Since Hadoop is Java based this is very useful to get an overview on what service is running on the node.

```
echo "Creating bash profile"
/usr/bin/perl -npe '/alias/ && s/^/#/' -i /root/.bashrc
cat <<EOF >> /root/.bashrc
set -o vi
alias grep='grep --colour=auto'
alias less='less -i'
alias ll='ls -la --color'
alias ls='ls --color=auto'
alias shutdown='echo SHUTDOWN!?!?'
alias syslog='tail -n 55 -f /var/log/messages'
alias dfs='/usr/bin/df -h | /usr/bin/grep -v ^none'
alias jps='/usr/java/default/bin/jps'
export HISTFILE=/hdfs/scratch/.bash_history_root
EOF
```

In the next part I have set up the NTP timekeeping with the chrony daemon. The time server was set to the orchestration server, time.irl is an A record (DNS alias) to the Raspberry Pi. Even though I have set up timekeeping I had difficulties due to the power supply disconnect which I’ve done from time to time as I did not want to leave the whole system always on. Read more on the workaround later.

```
echo "Setting up timekeeping with chrony"
/usr/bin/perl -npe '/^server/ && s/^/#/' -i /etc/chrony.conf
echo "server time.irl iburst" >> /etc/chrony.conf
```

The following is the readonly-root configuration. Here I have specified all the required files and folders which I wanted to keep read-write. Setting the TEMPORARY_STATE variable to yes ensures that all the read-write parts get mounted from tmpfs as part of the RAM.

```
echo "Marking root read-only"
/bin/sed -i 's/^READONLY.*/READONLY=yes/' /etc/sysconfig/readonly-root
/bin/sed -i 's/^TEMPORARY_STATE.*/TEMPORARY_STATE=yes/' /etc/sysconfig/readonly-root
echo "" >> /etc/rwtab
echo "dirs  /var/cache/yum" >> /etc/rwtab
echo "dirs  /var/lib/systemd" >> /etc/rwtab
echo "dirs  /etc/hadoop/conf.irl" >> /etc/rwtab
echo "dirs  /kernel" >> /etc/rwtab
echo "files /etc/hostname" >> /etc/rwtab
echo "files /etc/chrony.keys" >> /etc/rwtab
echo "files /etc/sysconfig/network" >> /etc/rwtab
echo "files /etc/rwtab" >> /etc/rwtab
echo "files /etc/sysconfig/readonly-root" >> /etc/rwtab
echo "files /var/lib/chrony/drift.tmp" >> /etc/rwtab
```

The following symlink was required as the kernel version has changed slightly due to a patch, it got a “4.4” in the versioning. Then I have disabled IPv6 via the sysctl facility; deleted the IPv6 entries from the hosts and the netconfig file and made sure sshd will listen on all the addresses:

```
echo "Linking kernel modules"
ln -s /lib/modules/3.10.0-123.el7.x86_64 /lib/modules/3.10.0-123.4.4.el7.x86_64
mkdir /kernel

echo "Disabling IPv6"
echo "net.ipv6.conf.all.disable_ipv6 = 1" >> /etc/sysctl.conf
echo "net.ipv6.conf.default.disable_ipv6 = 1" >> /etc/sysctl.conf
/bin/sed -i 's/^#\(Listen.*0.0.0.0\)$/\1/' /etc/ssh/sshd_config
/bin/sed -i '/^::.*/d' /etc/hosts
/bin/sed -i -e 's/^tcp6/#tcp6/' -e 's/^udp6/#udp6/' /etc/netconfig
```

The following part of the kickstart file is responsible for the unique hostnames. The hostname file contains the entry “goldenimage” which serves as an initial value. Then I define a startup script in /etc/rc.local which gets the hostname from DNS based via reverse lookup, and puts that to the hostname file during boot time. I also had to make sure that the rsyslog gets restarted so the correct hostname will show up in the logs. Also note the reference to the reload_config.sh which is a startup script loading the Hadoop configuration.

```
echo "Setting up rc.local"
echo "goldenimage" > /etc/hostname
cat <<EOF >> /etc/rc.d/rc.local
MYIPADDR=\$(ip addr show eno1 | sed -n 's/\sinet \([0-9]*\.[0-9]*\.[0-9]*\.[0-9]*\)\/[0-9]*
brd.*/\1/p')
MYHOSTNAME=\$(/usr/bin/dig -x \${MYIPADDR} +short | sed 's/\.$//')
/usr/bin/hostnamectl set-hostname \$MYHOSTNAME
echo \$MYHOSTNAME > /etc/hostname
echo HOSTNAME=\$MYHOSTNAME >> /etc/sysconfig/network

# to get correct hostname in syslog
systemctl restart rsyslog.service
# Loading Hadoop configuration
/root/reload_config.sh
EOF
chmod +x /etc/rc.d/rc.local
```

The following is a key part in the Hadoop startup; it gets called at boot time from rc.local. I wanted to keep the Hadoop configuration separate from the golden image, so it gets populated from a separate NFS share. The following snippet will create a startup script which mounts that NFS share and copies the configuration to the worker node. This was one of the reasons why I have disabled the automatic startup of the Hadoop services at boot time.

```
echo "Creating reload_config.sh"
cat <<EOF > /root/reload_config.sh
#!/bin/bash
# Populating Hadoop config from central repo

/usr/bin/rm /etc/hadoop/conf/* 2>/dev/null
/usr/bin/mount -o ro raspi:/btrfs/hadoop_conf /mnt
/usr/bin/cp -p /mnt/* /etc/hadoop/conf/
/usr/bin/umount /mnt

printf "%-20s%s\n" \$(hostname) "Hadoop configuration [RELOADED]"
EOF
chmod +x /root/reload_config.sh
```

Similarly to the reload_config.sh, the get_sources.sh can be used for fetching the kernel source onto the node. Note that /kernel is a read-write directory mounted from tmpfs so its contents are being deleted with a reboot. I could have set this to the persistent /hdfs filesystem so the already compiled parts won’t get recompiled, though I wanted to have the least possible overhead for that filesystem.

```
echo "Creating get_sources.sh"
cat <<EOF > /root/get_sources.sh
#!/bin/bash

/usr/bin/mount -o ro raspi:/btrfs/kernel /mnt
/usr/bin/xz --decompress --keep --stdout /mnt/linux-3.17-rc4.tar.xz | /usr/bin/tar -xf - -C
/kernel/
/usr/bin/umount /mnt

EOF
```

```
chmod +x /root/get_sources.sh
```

Then I have set the JAVA_HOME for all the Hadoop-related services; it is mentioned in the Cloudera documentation as well. After defining the Cloudera repository I am installing all the required CDH packages and the Oracle JDK in the chroot environment:

```
echo "Setting default JAVA_HOME"
echo 'export JAVA_HOME=/usr/java/default' >> /etc/default/bigtop-utils

echo "Adding Cloudera repository"
/usr/bin/curl http://repo.irl/cdh5/cdh5.repo > /etc/yum.repos.d/cdh5.repo
echo "Installing Hadoop"
/usr/bin/yum -y install hadoop-0.20-mapreduce-jobtracker hadoop-hdfs-namenode hadoop-hdfs-fuse
hadoop-0.20-mapreduce-tasktracker hadoop-hdfs-datanode hadoop-mapreduce hadoop-yarn hadoop-
yarn-nodemanager hadoop-yarn-resourcemanager hadoop-mapreduce-historyserver hadoop-hdfs-nfs3
jdk hue oozie hive
```

In the following section I change the repositories so they will point to my locally setup repo.irl running on my laptop, then I issue a yum update and then a yum clean:

```
echo "Updating YUM Repositories"
# disabling mirrorlist for os and updates
/usr/bin/perl -npe '/mirrorlist=.*repo=os/ && s/^/#/' -i /etc/yum.repos.d/CentOS-Base.repo
/usr/bin/perl -npe '/mirrorlist=.*repo=updates/ && s/^/#/' -i /etc/yum.repos.d/CentOS-
Base.repo
# enabling baseurl pointing to a single http server
/usr/bin/perl -npe '/^#baseurl=.*\/os\// && s/^#//' -i /etc/yum.repos.d/CentOS-Base.repo
/usr/bin/perl -npe '/^#baseurl=.*\/updates\// && s/^#//' -i /etc/yum.repos.d/CentOS-Base.repo
# setting my own repo
/usr/bin/perl -npe '/^baseurl=.*\/os\// && s/mirror.centos.org/repo.irl/' -i
/etc/yum.repos.d/CentOS-Base.repo
/usr/bin/perl -npe '/^baseurl=.*\/updates\// && s/mirror.centos.org/repo.irl/' -i
/etc/yum.repos.d/CentOS-Base.repo
# turning off keycheck
/usr/bin/perl -npe 's/^gpgcheck=1/gpgcheck=0/' -i /etc/yum.repos.d/CentOS-Base.repo
# and now update
/usr/bin/yum -y update
/usr/bin/yum clean all
```

The next section is needed because the Hadoop services got enabled by default after the installation. Since we have to pull the configuration from the central repository first, we cannot start the Hadoop services right away after a boot. The random seed service had to be disabled as well; there is no point to pick up the same initial seed on all the hosts.

```
echo "Disabling some legacy services and unneeded cron jobs"
# Hadoop roles handled by custom initscripts
/sbin/chkconfig hadoop-0.20-mapreduce-jobtracker off
/sbin/chkconfig hadoop-0.20-mapreduce-tasktracker off
/sbin/chkconfig hadoop-hdfs-datanode off
/sbin/chkconfig hadoop-hdfs-namenode off
/sbin/chkconfig hadoop-hdfs-nfs3 off
/sbin/chkconfig hadoop-mapreduce-historyserver off
/sbin/chkconfig hadoop-yarn-nodemanager off
/sbin/chkconfig hadoop-yarn-resourcemanager off
```

```
/sbin/chkconfig hue off
/sbin/chkconfig oozie off
/sbin/chkconfig iprupdate off
# we don't want to store rnd seed on a stateless server
/usr/bin/systemctl disable systemd-random-seed.service
rm /etc/cron.daily/*
rm /etc/cron.hourly/*
```

It is a best practice to keep site-specific Hadoop configuration under /etc/hadoop/conf-SITE. My environment was set up with the .irl domain so I've chosen the same for the site initials. The "alternatives" framework will make sure that the corresponding configuration gets used at every call. Check the man page for the update-alternatives command on how to set these symlinks. I have chosen to set the symlinks manually and set priority 15 to it:

```
echo "Preparing Hadoop configuration"
mkdir /etc/hadoop/conf.irl
rm /etc/alternatives/hadoop-conf
ln -s /etc/hadoop/conf.irl /etc/alternatives/hadoop-conf
echo "/etc/hadoop/conf.irl" >> /var/lib/alternatives/hadoop-conf
echo "15" >> /var/lib/alternatives/hadoop-conf
```

Initially I was planning to run an oozie and a hue server on the fourth node, and by that putting roughly the same load to all four nodes. These snippets are setting up the prerequisites for these services:

```
echo "Preparing Oozie configuration"
rm -r /var/lib/oozie
ln -s /hdfs/oozie /var/lib/oozie
/usr/bin/curl 'http://archive.cloudera.com/gplextras/misc/ext-2.2.zip' > /tmp/ext-2.2.zip
unzip /tmp/ext-2.2.zip -d /usr/lib/
rm /tmp/ext-2.2.zip

echo "Installing python 2.6 from EPEL - hue prerequisite"
/usr/bin/rpm --quiet -i http://download.fedoraproject.org/pub/epel/5/i386/epel-release-5-
4.noarch.rpm
/usr/bin/yum install python26

echo "Preparing Hue configuration"
cp -r /etc/hue/conf.empty /etc/hue/conf.irl
rm /etc/alternatives/hue-conf
ln -s /etc/hue/conf.irl /etc/alternatives/hue-conf
echo "/etc/hue/conf.irl" >> /var/lib/alternatives/hue-conf
echo "40" >> /var/lib/alternatives/hue-conf
sed -i 's/\(\s\)## \(webhdfs_url=http...\).*\(.webhdfs.*\)/\1\2namenode.irl:50070\3/'
/etc/hue/conf.irl/hue.ini
```

The following code snippet is required so the persistent filesystem gets mounted under /hdfs. The system will check all the available partitions and it will only mount the filesystem if it has the label HADOOP. This makes sure that with no matter which device file the local disk ends up it will always get identified and mounted as expected.

```
echo "Setting up HDFS backing store"
echo 'LABEL=HADOOP   /hdfs ext4  defaults 0 2' > /etc/fstab
mkdir /hdfs
```

The next part is for setting up the home directory and the profile for each Hadoop user. These users are yarn, hdfs and mapred. There is no need for a key-based SSH for them: I've always used the su command to change the effective user after logging in with root.

```
echo "Setting profile for user yarn"
echo "set -o vi" >> /var/lib/hadoop-yarn/.bash_profile
echo "alias jps=/usr/java/default/bin/jps" >> /var/lib/hadoop-yarn/.bash_profile
echo "export PS1=\"yarn % \"" >> /var/lib/hadoop-yarn/.bash_profile
echo "export HISTFILE=/hdfs/scratch/.bash_history_yarn" >> /var/lib/hadoop-yarn/.bash_profile
/usr/bin/chown 994:995 /var/lib/hadoop-yarn/.bash_profile
/usr/bin/mkdir -p /var/lib/hadoop-yarn/cache/yarn/nm-local-dir
/usr/bin/chown -R 995:995 /var/lib/hadoop-yarn/cache/yarn
```

Finally this last part concludes the kickstart file, which is for copying the sysstat binaries. Remember, this is the tool containing the system activity reporter (sar command) which was used for collecting all the metrics from the system during the tests.

```
echo "Deploying latest sysstat"
mount -o ro raspi:/btrfs/sysstat /mnt
cd /mnt
find . | cpio -pdm /
cd /
umount /mnt
```

You've probably noted that this package has already been specified during the base system install. I have also deployed this package to my laptop and that was the environment which I've used for generating all the graphs. As on my laptop I have Ubuntu, it contains another version of this tool. I found out that due to the version mismatch the binary performance file generated on CentOS 7 is not 100% compatible with the sysstat package I have on my desktop. So I have downloaded the sysstat sources from the internet and I've compiled common binaries for both my desktop and CentOS 7. So the above script just copies these compiled binaries from the central NFS located on the Raspberry, the same sysstat version will be part of the golden image.

### iii. Hardware

While I've spent my time working on the local repository the hardware has arrived in the first week of July so I could start assembling the parts. The first step was to download the Installation guide for the servers and go through it making sure that I comply with all the safety considerations. To prevent electrostatic discharge I have used an antistatic wrist strap connected to ground. During the installation of the internal parts the system was disconnected from the mains. I had to install the extra RAM and the additional network cards then I mounted the drive caddies to the hard disks and shoved them in to the drive bays.

After this all has been done the next step was to cable all the servers and connect them to the Cisco switch. For an overview on the cabling see Figure 2 in Chapter 3). Then aligning with the best practices I have updated the firmware on the systems. This covered an iLO update; a new BIOS and a recent firmware has been installed for the onboard NICs and for the storage controller as well.

The configuration of the Cisco switch did not take much time: I have connected to the switch over a serial cable then did a configuration reset. After that I set a password, configured an IP address for the management and enabled SSH. Once everything is set the crucial part was to copy the running-config to the startup-config so everything gets preserved after a reboot:

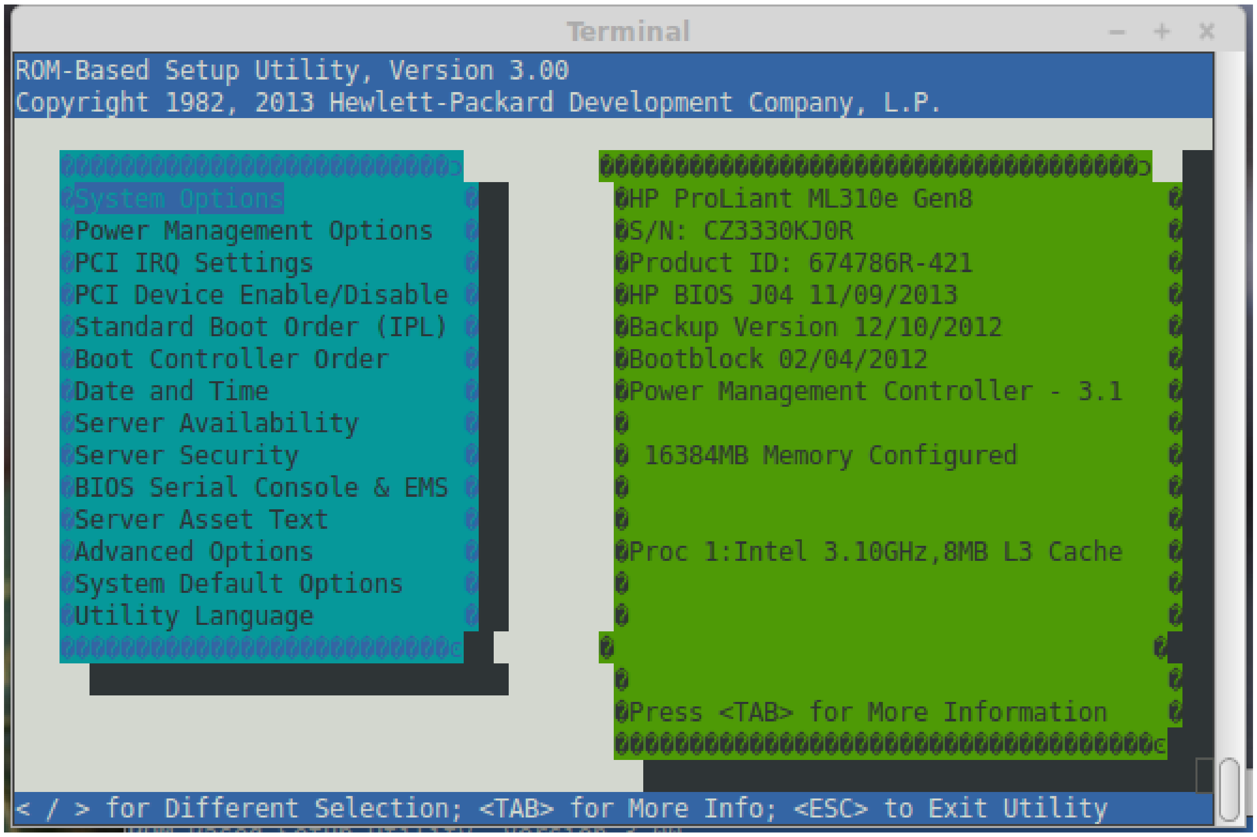
```
sw01#copy running-config startup-config
Destination filename [startup-config]?
Building configuration...
[OK]
sw01#
```

As the last step I have backed up the configuration of the switch for future reference. This can be found in Appendix C – Switch configuration. Note that even though this switch supports layer 3 functions, I haven’t configured routing on it since I already have a router in my home environment. I also did not configure any VLANs. From the security point of view it would be advisable to keep the lights out management separated from the production network traffic.

Since I have a Linux desktop I had to overcome the difficulties dealing with the iLO web interface. The whole management interface is designed to be compatible only with Internet Explorer. This caused issues especially with the remote console. So I had to borrow a Windows desktop for doing the initial configuration. As a workaround for the remote console I have found that key-based SSH can be used which allows me to connect to the remote console from an SSH session. The only gotcha was that I had to set the HostKeyAlgorithms SSH option to ssh-rsa, without that I would get a strange disconnect. Here is my SSH config file which I’ve used for the tests:

```
% cat ~/.ssh/config
GSSAPIAuthentication=no
ForwardAgent yes

Host athosilo
   User admin
   ServerAliveInterval 60
   HostKeyAlgorithms ssh-rsa
   IdentityFile /home/vbalogh/.ssh/id_rsa_ilo

Host porthosilo
   User admin
   ServerAliveInterval 60
   HostKeyAlgorithms ssh-rsa
   IdentityFile /home/vbalogh/.ssh/id_rsa_ilo

Host aramisilo
   User admin
   ServerAliveInterval 60
   HostKeyAlgorithms ssh-rsa
   IdentityFile /home/vbalogh/.ssh/id_rsa_ilo

Host dartagnanilo
   User admin
   ServerAliveInterval 60
   HostKeyAlgorithms ssh-rsa
   IdentityFile /home/vbalogh/.ssh/id_rsa_ilo

Host raspi
   User root
Host athos
   User root
Host porthos
   User root
Host aramis
   User root
Host dartagnan
   User root
%
```

The next screenshot in Figure 3 shows the BIOS screen over the remote console connected through SSH. The SSH console can display any ASCII character including the color escape sequences. For any graphical content I would get a blank screen over SSH therefore I had to activate text mode in BIOS in order to be able to track the boot process.



```
ROM-Based Setup Utility, Version 3.00
Copyright 1982, 2013 Hewlett-Packard Development Company, L.P.

System Options                    HP ProLiant ML310e Gen8
Power Management Options          S/N: CZ3330KJ0R
PCI IRQ Settings                  Product ID: 674786R-421
PCI Device Enable/Disable         HP BIOS J04 11/09/2013
Standard Boot Order (IPL)         Backup Version 12/10/2012
Boot Controller Order             Bootblock 02/04/2012
Date and Time                     Power Management Controller - 3.1
Server Availability
Server Security                    16384MB Memory Configured
BIOS Serial Console & EMS
Server Asset Text
Advanced Options                  Proc 1:Intel 3.10GHz,8MB L3 Cache
System Default Options
Utility Language

                                  Press <TAB> for More Information

< / > for Different Selection; <TAB> for More Info; <ESC> to Exit Utility
```

Figure 3: Remote console via SSH showing the BIOS main screen

## iv. PXE boot

After I have finished with the firmware update the next step was to prepare the golden image. For this I put an 8 GB USB thumb drive to the first host and booted it off the install server. The resulting USB installation has been booted one more time then I took an overview what kernel modules are being used. My goal was to create a smaller initrd ramdisk which also supports network boot. In CentOS the dracut module can be used for this. The following command line generated the required initrd package:

```
# dracut --host-only --nofscks --nolvmconf --nomdadmconf --add "nfs network base" \
--add-drivers "tg3 nfs lockd sunrpc nfsv4 nfs_acl dns_resolver fscache" \
 /root/diskless_initrd.img
```

The NFS root did not require any fsck, LVM or mdadm modules in the ramdisk. Though it is very important that one should include the driver for the network interface from which the host gets booted. In my case it is the TG3 driver for the integrated Broadcom-based NetXtreme adapter. I also added the fscache module for further testing. After I created the initial ramdisk I transferred it to the central repository, (running from the Raspberry):

```
# scp -p root@athos:/tmp/diskless_initrd.img /var/lib/tftpboot/centos/netboot/
```

So the generated ramdisk has been copied to the TFTP server residing on the Raspberry. The TFTP server provides the pxelinux.0 which is the bootloader; and the menu.c32 contains the menu structure. These are standard components of the syslinux package which is used for network booting. The file named “default” contains the boot menu entries and refers to each kernel image and its initial ramdisk. All the test kernels landed in the folder named “netboot”. The following shows the structure of the TFTP server, the files are reverse-sorted by mtime (modification time):

```
root@raspberry:~# find /tftp -type f -exec ls -ltr {} \+
-rw-r--r-- 1 root root    26595 Jul 26 21:49 /tftp/pxelinux.0
-rw-r--r-- 1 root root    61056 Jul 26 21:49 /tftp/menu.c32
-rwxr-xr-x 1 root root  4902656 Jul 27 00:15 /tftp/centos/install/vmlinuz
-rw-r--r-- 1 root root 33127644 Jul 27 00:15 /tftp/centos/install/upgrade.img
-rw-r--r-- 1 root root 34935964 Jul 27 00:15 /tftp/centos/install/initrd.img
-rw-r--r-- 1 root root  4903392 Aug  8 00:20 /tftp/centos/netboot/vmlinuz-3.10.0-123.4.4.el7.x86_64
-rw-r--r-- 1 root root 15109540 Aug 10 20:58 /tftp/centos/netboot/diskless_initrd.img
-rw-r--r-- 1 root root  4902656 Aug 24 17:21 /tftp/centos/netboot/vmlinuz-3.10.0-123.el7.x86_64.new
-rw-r--r-- 1 root root  5192560 Sep 14 21:48 /tftp/centos/netboot/vmlinuz-3.17.0-rc4-build101.x86_64
-rw-r--r-- 1 root root 21565880 Sep 14 22:41 /tftp/centos/netboot/initramfs-3.17.0-rc4-build101.x86_64.img
-rw-r--r-- 1 root root  2809040 Oct  5 23:19 /tftp/centos/netboot/vmlinuz-3.17.0-rc4-build102.x86_64
-rw-r--r-- 1 root root 11172820 Oct  5 23:35 /tftp/centos/netboot/initramfs-3.17.0-rc4-build102.x86_64.img
-rw-r--r-- 1 root root  2809152 Oct 25 15:22 /tftp/centos/netboot/vmlinuz-3.17.0-rc4-build103.x86_64
-rw-r--r-- 1 root root 11176332 Oct 25 15:30 /tftp/centos/netboot/initramfs-3.17.0-rc4-build103.x86_64.img
-rw-r--r-- 1 root root  2808864 Oct 27 22:06 /tftp/centos/netboot/vmlinuz-3.17.0-rc4-build104.x86_64
-rw-r--r-- 1 root root 11177088 Oct 27 22:10 /tftp/centos/netboot/initramfs-3.17.0-rc4-build104.x86_64.img
-rw-r--r-- 1 root root      960 Nov 11 18:05 /tftp/pxelinux.cfg/default
root@raspberry:~#
```

Here are the contents of the “default” file. There are three options specified there. The first one is the standard kernel which came with the default installation. The second entry is the recompiled kernel. I have changed this entry between the tests; this was also specified as the default option (see the “ONTIMEOUT” parameter). The third boot option is for completely reinstalling the node. Note that it does not endanger the contents of the local disk as the preinstall script would search for devices smaller than 16Gbytes. (see previous chapter on this)

```
root@raspberry:/# cat /tftp/pxelinux.cfg/default
DEFAULT menu.c32
PROMPT 0
TIMEOUT 30
ONTIMEOUT NFS_Boot_CentOS7_TEST
MENU TITLE PXE Network Boot
```

```
LABEL NFS_Boot_CentOS7
    MENU LABEL ^CentOS 7 NFS boot
    KERNEL centos/netboot/vmlinuz-3.10.0-123.4.4.el7.x86_64
    APPEND initrd=centos/netboot/diskless_initrd.img rdblacklist=igb rd.ip=auto root=dhcp
rd.luks=0 rd.lvm=0 rd.md=0 rd.dm=0 video=vesafb:off vga=normal nomodeset vga=normal

LABEL NFS_Boot_CentOS7_TEST
    MENU DEFAULT
    MENU LABEL ^CentOS 7 NFS boot TEST
    KERNEL centos/netboot/vmlinuz-3.17.0-rc4-build103.x86_64
    APPEND initrd=centos/netboot/initramfs-3.17.0-rc4-build103.x86_64.img rdblacklist=igb
rd.ip=auto root=dhcp rd.luks=0 rd.lvm=0 rd.md=0 rd.dm=0 video=vesafb:off vga=normal nomodeset
vga=normal

LABEL install_centos7
    MENU LABEL ^CentOS 7.0 (64-bit) Install
    KERNEL centos/install/vmlinuz
    APPEND ks=http://repo.irl/ks/b64.ks initrd=centos/install/initrd.img ramdisk_size=100000
ipv6.disable=1 nomodeset vga=normal
root@raspberry:/#
```

The timeout value is quite misleading in case of syslinux: one should specify the timeout in 1/10th seconds so a value of 30 just equals to 3 seconds in my case. Now let’s get back to the golden image as that still has to be transferred to the Raspberry. So I powered off the HP node I have just installed, pulled the USB stick and plugged it to the Raspberry. Then I transferred the contents of this base installation to the btrfs filesystem via find and cpio. That is the best way to transfer complex filesystem structures as it keeps all the permissions and does not change any special files. As I have already mentioned, I used btrfs due to its space-saving snapshot feature. The following shows the layout of the btrfs filesystem on the Raspberry; the backing store was provided by an additional USB stick:

```
root@raspberry:/# ll /btrfs/
total 20
drwxr-xr-x  1 root root  422 Sep 14 22:18 .
drwxr-xr-x 25 root root 4096 Aug 18 21:24 ..
drwxr-xr-x  1 root root  344 Jan 11 21:44 BACKUP
drwxr-xr-x  1 root root  172 Sep 14 18:12 CentOS7_Hadoop
drwxr-xr-x  1 root root  172 Sep 14 18:12 CentOS7_Hadoop_1409_1726
drwxr-xr-x  1 root root  160 Aug 31 18:30 CentOS7_Hadoop_3108_1828
drwxr-xr-x  1 root root  468 Oct  1 23:15 hadoop_conf
drwxr-xr-x  1 root root   96 Sep 14 22:45 kernel
drwxr-xr-x  1 root root  346 Sep 14 22:30 modules_3.17.0-rc4-build101
drwxr-xr-x  1 root root  346 Oct  5 23:34 modules_3.17.0-rc4-build102
drwxr-xr-x  1 root root  346 Oct 25 15:24 modules_3.17.0-rc4-build103
drwxr-xr-x  1 root root  346 Oct 27 22:09 modules_3.17.0-rc4-build104
drwxr-xr-x  1 root root    0 Dec 29 18:37 SAR_RESULTS
drwxr-xr-x  1 root root   12 Sep  6 22:33 sysstat
root@raspberry:/#
```

As this project was critical for my Cloud Computing MSc studies, I have scheduled a regular backup of the most important files. The BACKUP folder just contains those files before uploading to Dropbox. The folders named “Centos7*” are the golden images which have been copied. The two with the timestamps are just previous versions created by btrfs snapshots. The kernel folder is the place for the pulled kernel sources, the corresponding modules are within the modules* folders. The SAR_RESULTS directory contains the test results collected from the nodes just before powering them off. This ensured that I could automatically power off the systems after finishing with the tests, yet I was able to process the results. The sysstat is the already mentioned performance data collector; this was the place for manually compiled version.

## v. Persistent store

Now that the golden image is on the boot server I could boot the nodes and make the required changes to the local disk. First I have created a single partition on the local disks and formatted it with ext4:

```
# parted /dev/sda mklabel gpt
# parted /dev/sda mkpart primary ext4 1 100%
# mke2fs -t ext4 -L HADOOP -m 0 /dev/sda1
# mount /dev/sda1 /hdfs
```

Note the label “HADOOP” which has been added via mke2fs. This makes sure that no matter in which order the disks get recognized during boot, they will get mounted in the right order. In my case I only had a single disk but this is a handy feature for the case of a future expansion. After mounting the /hdfs filesystem I have created further structures on it:

```
# mkdir -p /hdfs/namenode/hdfs_cache
# mkdir -p /hdfs/yarn/nm-local-dir
# chown 994:995 /hdfs/yarn/nm-local-dir
# mkdir /hdfs/datadir
# chown -R hdfs:hdfs /hdfs
# chmod 700 /hdfs/namenode/hdfs_cache
# chmod 700 /hdfs/datadir

# mkdir /hdfs/scratch
# chown root:root /hdfs/scratch
# chmod 1777 /hdfs/scratch

# mkdir -p /hdfs/kernel
# chown -R root:root /hdfs/kernel

# mkdir /hdfs/sar

# mkdir -p /hdfs/oozie/work
# chown -R 991:989 /hdfs/oozie
# ln -s /usr/lib/ext-2.2 /hdfs/oozie/ext-2.2
```

The right permissions are very crucial in case of Hadoop. I was fighting with strange errors in the java stacktrace at first until I realized that it was a permission problem. So as a best practice one should read the Cloudera documentation before deploying. Here is the list of Hadoop users which get created with the default installation:

```
root@raspberry:/# tail -n7 /btrfs/CentOS7_Hadoop/etc/passwd
zookeeper:x:997:996:ZooKeeper:/var/run/zookeeper:/sbin/nologin
hue:x:996:994:Hue:/usr/lib/hue:/sbin/nologin
hdfs:x:995:993:Hadoop HDFS:/var/lib/hadoop-hdfs:/bin/bash
yarn:x:994:992:Hadoop Yarn:/var/lib/hadoop-yarn:/bin/bash
mapred:x:993:991:Hadoop MapReduce:/var/lib/hadoop-mapreduce:/bin/bash
hive:x:992:990:Hive:/var/lib/hive:/sbin/nologin
oozie:x:991:989:Oozie User:/var/lib/oozie:/bin/false
root@raspberry:/#
```

Once the persistent store for the HDFS has been created and all the permissions were correctly set, I could start the Hadoop services on their respective nodes - see Figure 1 for their placement.

### vi. Initializing HDFS

In order to be able to start Hadoop I had to come up with an initial configuration. Some basic setting like the list of the nodes (slave file) and the path to the HDFS backing store had to be set up. I will elaborate more on this later. In case of HDFS the first step is to format the namenode then start the service:

```
# su - hdfs -c "hdfs namenode -format"
# service hadoop-hdfs-namenode start
```

Once HDFS is up and running we have to create some directory structures on it which will be used by other Hadoop components. Note the changed prompt: I have switched to the hdfs user with the su command before executing these:

```
hdfs % hdfs dfs -mkdir -p /var/log/hadoop-yarn/apps
hdfs % hdfs dfs -chmod 1777 /var/log/hadoop-yarn/apps
hdfs % hadoop fs -mkdir /tmp
hdfs % hadoop fs -chmod -R 1777 /tmp
hdfs % hadoop fs -mkdir -p /user/history
hdfs % hadoop fs -chmod -R 1777 /user/history
hdfs % hadoop fs -chown mapred:hadoop /user/history
hdfs % hadoop fs -mkdir -p /var/log/hadoop-yarn
hdfs % hadoop fs -chown yarn:mapred /var/log/hadoop-yarn
hdfs % hadoop fs -mkdir /user/oozie
hdfs % hadoop fs -chown oozie:oozie /user/oozie
```

For oozie to work we have to copy the local libraries to the HDFS store then the provided script will create the database structure for the underlying Apache Derby database:

```
# su - oozie
# oozie-setup sharelib create -fs hdfs://namenode:8020 -locallib /usr/lib/oozie/oozie-
sharelib-yarn.tar.gz
# /usr/lib/oozie/bin/ooziedb.sh create –run
```

Finally here is how it should look like after creating all the required directories. Note the permissions as well; the right permissions here are also very important:

```
hdfs % hadoop fs -ls -R /
drwxrwxrwt   - hdfs hadoop          0 2014-08-30 18:14 /tmp
drwxr-xr-x   - hdfs hadoop          0 2014-08-30 18:15 /user
drwxrwxrwt   - mapred hadoop          0 2014-08-30 18:15 /user/history
drwxr-xr-x   - hdfs   hadoop          0 2014-08-30 17:45 /var
drwxr-xr-x   - hdfs   hadoop          0 2014-08-30 17:45 /var/log
drwxr-xr-x   - yarn   mapred          0 2014-08-30 17:46 /var/log/hadoop-yarn
drwxr-xr-x   - hdfs   hadoop          0 2014-08-30 17:46 /var/log/hadoop-yarn/apps
hdfs %
```

As I have mentioned earlier, I was planning to deploy oozie and the hue server on the fourth node. These would have enabled me to get a graphical interface for managing Hadoop and scheduling jobs. However it turned out that these services are heavily relying on python 2.6 and the CentOS (Red Hat) 7 already comes with python 2.7 pre-packaged. At first I was thinking about either getting the earlier oozie and hue binaries or compiling python 2.6 from source. However I realized that python is an essential component of this distribution. It even gets deployed with a base installation as facilities like the yum package manager are written in python. So I did not spend any more time on this issue. I accepted it as a trade-off for my decision on not to go with a supported CentOS version.

## vii. Functional testing

After I have completed the setup of everything the next step was to conduct functional tests of the operating system and the Hadoop framework. I did some startup tests with all four nodes turned on at once and monitored the resource usage on the Raspberry Pi. The CPU load went high during the loading of the kernel and initrd however this was capped all the time at around

2.0. Regarding the boot process the bottleneck was clearly the 100 Mbps interface on the Raspberry. That throughput divided between the four nodes concluded to a roughly 3 Mbytes/sec throughput for a single node. However after the system has been loaded everything was running fine thanks to the NFS caching. So it is crucial to warm the cache before starting the tests.

I also did functional tests on the Hadoop framework. For this I had to start all the required services on all the nodes manually. It is a best practice to start them via the RHEL "service" facility: *"…when starting, stopping and restarting CDH components, always use the service (8) command rather than running /etc/init.d scripts directly. This is important because service sets the current working directory to / and removes most environment variables (passing only LANG and TERM) so as to create a predictable environment in which to administer the service. If you run the /etc/init.d scripts directly, any environment variables you have set remain in force, and could produce unpredictable results." [21]* Once all the services were up and running I could start experimenting with some basic jobs. The Hadoop framework comes with the following built-in examples:

```
yarn % yarn jar /usr/lib/hadoop-mapreduce/hadoop-mapreduce-examples.jar
An example program must be given as the first argument.
Valid program names are:
  aggregatewordcount: An Aggregate based map/reduce program that counts the words in the input
files.
  aggregatewordhist: An Aggregate based map/reduce program that computes the histogram of the
words in the input files.
  bbp: A map/reduce program that uses Bailey-Borwein-Plouffe to compute exact digits of Pi.
  dbcount: An example job that count the pageview counts from a database.
  distbbp: A map/reduce program that uses a BBP-type formula to compute exact bits of Pi.
  grep: A map/reduce program that counts the matches of a regex in the input.
  join: A job that effects a join over sorted, equally partitioned datasets
  multifilewc: A job that counts words from several files.
  pentomino: A map/reduce tile laying program to find solutions to pentomino problems.
  pi: A map/reduce program that estimates Pi using a quasi-Monte Carlo method.
  randomtextwriter: A map/reduce program that writes 10GB of random textual data per node.
  randomwriter: A map/reduce program that writes 10GB of random data per node.
  secondarysort: An example defining a secondary sort to the reduce.
  sort: A map/reduce program that sorts the data written by the random writer.
  sudoku: A sudoku solver.
  teragen: Generate data for the terasort
  terasort: Run the terasort
  teravalidate: Checking results of terasort
  wordcount: A map/reduce program that counts the words in the input files.
  wordmean: A map/reduce program that counts the average length of the words in the input
files.
  wordmedian: A map/reduce program that counts the median length of the words in the input
files.
  wordstandarddeviation: A map/reduce program that counts the standard deviation of the length
of the words in the input files.
yarn %
```

I have started with some smaller jobs in order to verify basic functionality. A quick but useful example job is the Bailey-Borwein-Plouffe algorithm for computing the digits of pi. I have also used it for warming the cache before running the benchmark tests. Here is an example which specifies the first 40 digits of pi by using 4 mapper processes, the output gets stored under /benchmarks/bbp on HDFS:

```
yarn % yarn jar /usr/lib/hadoop-mapreduce/hadoop-mapreduce-examples.jar bbp 1 40 4 /benchmarks/bbp
```

It is also possible to specify parameters at the command line, thereby overriding the default settings. The following is a TeraGen example where I have overridden the default number of map tasks with the –D option:

```
yarn % yarn jar /usr/lib/hadoop-mapreduce/hadoop-mapreduce-examples.jar teragen \
-Dmapred.map.tasks=1000 /benchmarks/teratest
```

There can be more than one parameter specified at the same command line, for every one of them an additional leading –D is required. I have used this feature several times for my testing framework. For more useful examples and a nice explanation on the TeraTest tools I have found a very good reading material in the Hortonworks documentation. [22]

I have already mentioned the jps tool which lists all the running JVM processes and their PID (process ID). That gives a nice overview if all the required services are running on the node. Note that this tool is not included in the default PATH variable so I've set up an alias in my bash profile for easier access. Here is the output of jps command from each node:

```
[root@athos ~]# jps
2599 DataNode
2823 NodeManager
2998 Jps
2455 NameNode
[root@athos ~]#

[root@porthos ~]# jps
3133 Jps
2399 DataNode
2548 ResourceManager
2694 NodeManager
[root@porthos ~]#

[root@aramis ~]# jps
2404 DataNode
2553 NodeManager
2993 Jps
2706 JobHistoryServer
```

```
[root@aramis ~]#

[root@dartagnan ~]# jps
2552 NodeManager
2402 DataNode
2803 Jps
[root@dartagnan ~]#
```

Note that the process ID for the jps process itself will also get displayed as this is a java tool running in JVM. This output corresponds exactly to the service placement in Figure 1, the additional hue and oozie services are missing from the fourth node due to incompatibility issues with python.

## viii. Hadoop parameters

After I have confirmed the functionality with the successful smoke tests I have started to configure reasonable Hadoop parameters before kicking off with the benchmarks. I know I have mentioned that this project is not about the Hadoop parameter tuning yet I wanted to start at least with meaningful parameters and conduct the further tests in a sanely configured environment. For sizing the various parameters I have found various sources on the internet. It is really a very hard and time consuming task to figure out the optimal values. It seems that despite some are mentioning best practices; there is no such thing as an ultimate configuration. The values are heavily dependent on the hardware specifications and on the type of Hadoop jobs.

One of the main questions was the number of mappers and reducers I have to configure. I should mention that my servers are equipped with Intel Xeon E3-1220 v2 which does not support Hyper-Threading technology, which had to be taken into account for the sizing. The best sizing guide I've found is the one in the Hortonworks documentation [23]. I have also found a very good blog article titled "Anatomy of a MapReduce Job" [24] which has helped with the sizing.

In my case each node has 16 Gbytes or RAM, out of which I have reserved 2 Gbytes for the system, see the table in the Hortonworks guide. Since I did not plan to utilize HBASE this concludes to a total of 14 Gbytes usable for Hadoop on a single node. Now from [23] here is the formula on how to calculate the number of containers:

```
# of containers = min (2*CORES, 1.8*DISKS, (Total available RAM) / MIN_CONTAINER_SIZE)
```

The recommended minimum container size is 1 Gbyte for my setup (again, see the Hortonworks guide). So the number of containers would be 8 per node, and from this formula [23] I could conclude on a value of 2 Gbytes per container:

```
RAM-per-container = max(MIN_CONTAINER_SIZE, (Total Available RAM) / containers))
```

Table 1 lists the most important parameters I have changed. Note that these have been changed to simulate a reasonably configured Hadoop environment which provided the baseline for the benchmarks. However I did not alter any of these parameters during the further tests. For the list of all parameters see the enclosed configuration files. I have conducted some initial tests and came up with the following settings for my environment:

| **Configuration** | **Value Calculation** |
|---|---|
| yarn.nodemanager.resource.memory-mb | = 15360  MB |
| yarn.scheduler.minimum-allocation-mb | = 2048 MB |
| yarn.scheduler.maximum-allocation-mb | = 15360 MB |
| mapreduce.map.memory.mb | = 2048 MB |
| mapreduce.reduce.memory.mb | = 4096 MB |
| mapreduce.map.java.opts | = -Xmx1638m |
| mapreduce.reduce.java.opts | = -Xmx3276m |
| yarn.app.mapreduce.am.resource.mb | = 4096 MB |
| yarn.app.mapreduce.am.command-opts | = -Xmx3276m |

**Table 1: Hadoop parameter sizing**

## ix. Testing framework

Having a Systems Engineer background I’ve found it easier to write my own test framework for this project rather than learning and adapting some existing tool. This gave me the flexibility that I could customize the testing framework according to my own needs. I’ve coded scripts which – started from the Raspberry – can turn on the servers, start Hadoop, warm the cache and then run the several hour long jobs. I came up with a runbook – a group of Hadoop test jobs – which are running in a loop for multiple times.

The operating system specific metrics get monitored by the system activity reporter (sar). The job logs are also saved and there is a history server running within the cluster so all errors can be tracked. The system activity reporter gets started for each job separately. This ensures that the graphs will only contain the relevant metrics for the given job. The jobs are extensively logged and there is a possibility to inject comments to the sar raw metrics. At the end of each job the sar raw data gets pulled to the central Raspberry server, and finally a script generates the graphs from these. Here are the scripts I’ve written for the testing:

```
root@raspberry:~/scr# ll *sh
-rwxr-xr-x 1 root root 1034 Dec 29 20:28 get_hadoop_status.sh
-rwxr-xr-x 1 root root  526 Dec 29 20:28 poweroff_nodes.sh
-rwxr-xr-x 1 root root  248 Dec 29 20:28 poweron_nodes.sh
-rwxr-xr-x 1 root root  368 Dec 29 20:27 reboot_nodes.sh
-rwxr-xr-x 1 root root  262 Dec 29 20:27 reload_all_config.sh
-rwxr-xr-x 1 root root 3095 Dec 29 20:27 run_job.sh
-rwxr-xr-x 1 root root  927 Dec 27 14:06 schedule.sh
-rwxr-xr-x 1 root root 1348 Dec 29 20:26 start_hadoop.sh
-rwxr-xr-x 1 root root  839 Dec 29 20:28 stop_hadoop.sh
root@raspberry:~/scr#
```

In the following I will explain the basic functionality of these scripts. The main script is the schedule.sh, which calls all the helper scripts. The scripts are based on keyed SSH and the captured metrics and logs are being tracked through timestamp and job comment. Therefore specifying a job comment is mandatory:

```
root@raspberry:~/scr# cat schedule.sh
#!/bin/bash

if [ "$1" = "" ];then
   echo "Specify job comment!"
   exit 0
fi

if [ "$2" != "--nostart" ]; then
```

```
   if [ "$2" != "--nopoweron" ]; then
      /bin/date "+%y/%m/%d %H:%M:%S Powering on servers"
      /root/scr/poweron_nodes.sh 1>/dev/null 2>/dev/null
   fi

   /bin/date "+%y/%m/%d %H:%M:%S Waiting while servers are booting"
   sleep $((10*60))       # wait 10 minutes

   /bin/date "+%y/%m/%d %H:%M:%S Starting Hadoop services"
   /root/scr/start_hadoop.sh 1>/dev/null 2>/dev/null

   /bin/date "+%y/%m/%d %H:%M:%S Waiting while services are settling"
   sleep $((10*60))       # wait 10 minutes
fi

for MYJOB in $(/bin/grep -v -e "^#" SCHEDULED.LIST)
do
   /bin/date "+%y/%m/%d %H:%M:%S Starting job ${MYJOB}"
   /root/scr/run_job.sh ${MYJOB} "$1" >/dev/null 2>/dev/null
done

if [ "$2" != "--nopoweroff" ] && [ "$3" != "--nopoweroff" ]; then
   /bin/date "+%y/%m/%d %H:%M:%S Powering off nodes"
   /root/scr/poweroff_nodes.sh 1>/dev/null 2>/dev/null
fi
root@raspberry:~/scr#
```

I have implemented additional parameters such as “--nostart” and “--nopoweron” though I haven’t used these options for the benchmark tests. As you can see the schedule.sh powers on the nodes then starts the Hadoop services and finally it goes through the list of specified jobs from the benchmark runbook. Here are the contents of the poweron_nodes.sh script, which starts the nodes through keyed SSH iLO connection:

```
root@raspberry:~/scr# cat poweron_nodes.sh
#!/bin/bash

trap echo 2

SSH="/usr/bin/ssh -o BatchMode=yes -o ConnectTimeout=10 -o StrictHostKeyChecking=no"

echo "Powering on nodes"
for DATANODE in `</btrfs/hadoop_conf/slaves`
do
   ${SSH} -o BatchMode=yes ${DATANODE}ilo "power on" &
done
wait
root@raspberry:~/scr#
```

As it can be seen the node names are fetched from the slaves file from the Hadoop configuration; this ensures scalability and seamless integration with Hadoop. The SIGINT signal (interrupt) gets ignored as we don’t want this script to be interrupted via an accidental CTRL+C. That would just leave the cluster in an inconsistent state. All the nodes are being powered on parallel through issuing the SSH commands in the background; then the wait command will wait until all of them are finished. As best practice I've used BatchMode for SSH and turned off StrictHostKeyChecking as the host keys can differ due to frequent reinstalls of the golden image. I have found that the ConnectTimeout has to be raised in case of a higher number of nodes since my orchestration server (Raspberry Pi) has fewer resources. All the other scripts are similar to this one, they are based on for loops and similar keyed SSH sessions running parallel.

Another important thing was timekeeping. Since I've kept the servers and the Cisco switch unplugged between the tests, the system clock was always reverting to the year 2012. I've set up an NTP server on the Raspberry Pi yet I had to make sure that the date gets adjusted at server boot time as with NTP only slight changes are allowed. Therefore in the start_hadoop.sh script I have implemented a loop which temporarily stopped the NTP daemon and set the current time before starting the Hadoop services.

The names of the tests are defined in the SCHEDULED.LIST file; every line starting with a hashmark is considered to be a comment. So in case of running the main script the following tests will get executed in the specified order (first 11 lines are shown, these jobs have been running 5 times in a loop by default):

```
root@raspberry:~/scr# head -n 11 SCHEDULED.LIST
BBP_1_16384_16
BBP_1_16384_64
BBP_1_100k_8
BBP_1_100k_32
TeraGen_16G
TeraSort_16G
TeraValidate_16G
TeraGen_1T
TeraSort_1T
TeraValidate_1T
#
root@raspberry:~/scr#
```

The BBP tests are the Bailey-Borwein-Plouffe tests for generating the exact value of pi for the given number of digits. Those are purely CPU-bound tests and rather uncommon for a normal Hadoop job so I have only used those for warming the NFS cache. I have designed the framework to handle predefined job profiles stored in configuration files. All the job profiles are kept under the JOBS subdirectory. I have defined the following jobs on the Raspberry:

```
root@raspberry:~/scr# ls -l JOBS/
total 60
-rw-r--r-- 1 root root 853 Oct  2 00:07 BBP_1_100k_32.job
-rw-r--r-- 1 root root 850 Oct  2 00:13 BBP_1_100k_8.job
-rw-r--r-- 1 root root 854 Sep 29 23:43 BBP_1_16384_16.job
-rw-r--r-- 1 root root 854 Sep 29 23:43 BBP_1_16384_64.job
-rw-r--r-- 1 root root 848 Sep 29 22:43 BBP_1_256_16.job
-rw-r--r-- 1 root root 843 Sep 29 22:44 BBP_1_64_16.job
-rw-r--r-- 1 root root 849 Dec 27 14:03 TeraGen_16G.job
-rw-r--r-- 1 root root 855 Sep 29 23:01 TeraGen_1T.job
-rw-r--r-- 1 root root 856 Sep 29 23:37 TeraGen_4T.job
-rw-r--r-- 1 root root 848 Dec 27 14:03 TeraSort_16G.job
-rw-r--r-- 1 root root 848 Sep 29 23:38 TeraSort_1T.job
-rw-r--r-- 1 root root 849 Sep 29 23:38 TeraSort_4T.job
-rw-r--r-- 1 root root 878 Dec 27 14:03 TeraValidate_16G.job
-rw-r--r-- 1 root root 878 Oct  5 19:10 TeraValidate_1T.job
-rw-r--r-- 1 root root 875 Sep 29 23:39 TeraValidate_4T.job
root@raspberry:~/scr#
```

As it is shown above I have also defined a TeraTest with 4 Tbytes of dataset. Though as the nodes only have 3 Tbyte HDDs and I've left the HDFS replication factor on the default value of 3, it turned out that my system does not have enough capacity for that test. In the following I will demonstrate the job profile for the 16 Gbytes TeraGen job:

```
root@raspberry:~/scr# cat JOBS/TeraSort_16G.job
# Hadoop job description file
#
# this gets into sar binary data as comment 64 chars max
JOB_NAME="TeraSort 16G"
#
#
JOB_BINARY=/usr/lib/hadoop-mapreduce/hadoop-mapreduce-examples.jar
#
#
# prefix used for output dir, a timestamp will be appended as postfix
JOB_OUTDIR=/benchmarks/TeraSort16G_sorted
#
#
# terasort <input dir>
JOB_OPTS="terasort /benchmarks/TeraSort16G"
#
#
# how much we should wait BEFORE starting job
SADC_PRELUDE=30
#
#
# how much we should wait AFTER starting job
SADC_POSTLUDE=30
#
#
# sampling interval in seconds for sadc, 1 sec is the minimum
# keep it high for larger jobs in order to prevent massive amount of logs
SADC_INTERVAL=1
#
#
# if output should be rolled over at the end
# note that cleanup will be ignored if set
JOB_ROLLOVER="true"
#
```

```
#
# if output should be deleted after finishing the job
JOB_CLEANUP="true"
root@raspberry:~/scr#
```

As it can be seen, this is basically just a set of variables which get sourced by the main script. Every job definition has the same variables but with different values. The performance data gets collected by the sar (system activity reporter) subsystem. One can define “prelude” and “postlude” values; these are basically settling intervals before and after the job execution. This way we’ll end up with nice graphs where one can see the system settling down.

The results from the job can be kept or deleted immediately, for these the JOB_ROLLOVER and JOB_CLEANUP variables should be set respectively. For example in case of TeraTest the output of TeraGen will serve as the input of the next TeraSort test, and the output of this will be fed as input to the TeraValidate job. The following line is from the run_job.sh, this is how the data collection gets started before each Hadoop job:

```
${SSH} ${NODE} '/usr/bin/kill $(pidof sadc) 2>/dev/null ;
/usr/local/lib/sa/sadc -C "'${JOB_NAME} - ${RUNTIME_COMMENT}'"\
'/hdfs/sar/${JOB}__${TIMESTAMP}_${NODE}.sa' ;'
"/usr/bin/nohup /usr/local/lib/sa/sadc -S ALL ${SADC_INTERVAL}
/hdfs/sar/${JOB}__${TIMESTAMP}_${NODE}.sa 1>/dev/null 2>&1 &" &
```

First I wanted to make sure that sadc (sar data collector) is not running then I started it with the parameters defined in the job profile. All the tests are identified by a RUNTIME_COMMENT and also by a TIMESTAMP. This ensures that for each run I would get a unique identifier. The generated sar binary was also stamped by the node name it is coming from, the resulting file has the .sa ending. Once the sar data collector is running this is how a Hadoop job gets started:

```
/bin/date "+%y/%m/%d %H:%M:%S Starting job ${JOB}" | tee --append ${LOGFILE}
STARTTIME=$(/bin/date "+%s.%N")
/bin/date "+%y/%m/%d %H:%M:%S Kernel version is \"${KERNEL_VERSION}\"" | tee --append
${LOGFILE}
/bin/date "+%y/%m/%d %H:%M:%S Runtime comment is \"${RUNTIME_COMMENT}\"" | tee --append
${LOGFILE}
${SSH} resourcemgr.irl "su yarn -c \"/bin/yarn jar ${JOB_BINARY} ${JOB_OPTS}
${JOB_OUTDIR}_${TIMESTAMP}\"" 2>&1 | tee --append ${LOGFILE}
STOPTIME=$(/bin/date "+%s.%N")
JOB_DURATION=$(printf "%.3f" $(echo ${STOPTIME}-${STARTTIME} | /usr/bin/bc -l))
/bin/date "+%y/%m/%d %H:%M:%S Job ${JOB} has been finished" | tee --append ${LOGFILE}
/bin/date "+%y/%m/%d %H:%M:%S It was running for ${JOB_DURATION} seconds." | tee --append
${LOGFILE}
```

As it can be seen I have logged the respective kernel version and the “runtime comment” which identifies each test round. Once the job has finished I also calculated the runtime in milliseconds and appended to the logfile then stopped sadc and fetched the resulting binary sar data:

```
/bin/date "+%y/%m/%d %H:%M:%S Stopping performance collector"
for NODE in `</btrfs/hadoop_conf/slaves`
do
        ${SSH} ${NODE} '/usr/bin/kill $(pidof sadc) 2>/dev/null' &
done
wait

/bin/date "+%y/%m/%d %H:%M:%S Fetching performance data"
for NODE in `</btrfs/hadoop_conf/slaves`
do
        ${RSYNC} ${NODE}:/hdfs/sar/* /btrfs/SAR_RESULTS
done
```

After that the Hadoop system can be powered off completely and disconnected from the mains yet I was able to access the results from the Raspberry. The Cisco switch does not have a power button so the easiest was to disconnect the whole setup from the mains including the switch.

## x. Graph generation

With the data collector of the system activity reporter I have generated binary files during each job. The further processing was done on my Linux desktop by the process_sar_data.sh script. First I have generated the ASCII metrics from the sar binary (excerpt from the process_sar_data.sh):

```
LC_TIME=POSIX /usr/local/bin/sar -C -u ALL -f ${BINFILE} >> ${SARPATH}/${OUTFILE}.txt
# CPU all
##LC_TIME=POSIX /usr/local/bin/sar -I SUM -f ${BINFILE} >> ${SARPATH}/${OUTFILE}.txt
# Interrupts
LC_TIME=POSIX /usr/local/bin/sar -q -f ${BINFILE} >> ${SARPATH}/${OUTFILE}.txt
# Load
LC_TIME=POSIX /usr/local/bin/sar -w -f ${BINFILE} >> ${SARPATH}/${OUTFILE}.txt
# Processes / Contexts
#
LC_TIME=POSIX /usr/local/bin/sar -R -f ${BINFILE} >> ${SARPATH}/${OUTFILE}.txt
# Page
LC_TIME=POSIX /usr/local/bin/sar -B -f ${BINFILE} >> ${SARPATH}/${OUTFILE}.txt
# Paging activity
LC_TIME=POSIX /usr/local/bin/sar -r -f ${BINFILE} >> ${SARPATH}/${OUTFILE}.txt
# Memory usage / misc
#
LC_TIME=POSIX /usr/local/bin/sar -b -f ${BINFILE} >> ${SARPATH}/${OUTFILE}.txt
# I/O
```

```
##LC_TIME=POSIX /usr/local/bin/sar -d -f ${BINFILE} >> ${SARPATH}/${OUTFILE}.txt
# Block transfer / wait
#
# filtering out unused adapter
LC_TIME=POSIX /usr/local/bin/sar -n ALL -f ${BINFILE} | grep -v -e eno2 -e ens >>
${SARPATH}/${OUTFILE}_interface.txt
# Interface traffic / errors / NFS / Sockets
```

The LC_TIME had to be set to POSIX as ksar cannot interpret the am/pm time format. I have filtered out the network statistics for the eno2 and the ensX interfaces as these were unplugged in my environment.

After some days of testing I have noticed a bug in the framework: for longer running jobs which span over midnight ksar got confused about the sar data start and end timestamp. This is because sar is generally used by a cronjob rotating the binary at midnight. In my case I had to align the timestamp i.e. had to put a new header indicating the start of a new day in the ASCII file. This part is done here (excerpt from process_sar_data.sh):

```
# putting new sar header for each starting day
# this is required for jobs running over multiple days
LASTTIMESTAMP=$(/usr/bin/awk '/^23:5/ {print $1}' ${SARPATH}/${OUTFILE}.txt | tail -n 1)
if [ -n "${LASTTIMESTAMP}" ] # run this hook only if the job stretched multiple days
then
        TODAY=$(/usr/bin/awk 'NR == 1 {print $4}' ${SARPATH}/${OUTFILE}.txt)
        TOMORROW=$(/bin/date --date="${TODAY} 1 day" +%m/%d/%y)
        OLDHEADER=$(/usr/bin/head -n 1 ${SARPATH}/${OUTFILE}.txt)
        NEWHEADER=$(/usr/bin/head -n 1 ${SARPATH}/${OUTFILE}.txt |
         /bin/sed 's,'${TODAY}','${TOMORROW}',')
        # and this puts the new header
        /bin/sed -i '/^'${LASTTIMESTAMP}'/a '"${NEWHEADER}" ${SARPATH}/${OUTFILE}.txt
        #
        /bin/sed -i '/^'${LASTTIMESTAMP}'.*lo/a '"${NEWHEADER}" \
         ${SARPATH}/${OUTFILE}_interface.txt
        /bin/sed -i '/eno1/! { s,\(^Average.*\),\1'"\n""${OLDHEADER}"', }' \
         ${SARPATH}/${OUTFILE}_interface.txt
        /bin/sed -i '$d' ${SARPATH}/${OUTFILE}_interface.txt
        /bin/sed -i '/eno1/! { s,\(^'${LASTTIMESTAMP}'.*\),\1'"\n""${NEWHEADER}"', }' \
         ${SARPATH}/${OUTFILE}_interface.txt
fi
```

After all of this is done the ASCII file is formatted and ready for ksar, so the PNG files can be generated by ksar. It is basically done by the following oneliner, plus we also get a nice formatted PDF by just using a further parameter of ksar:

```
/usr/bin/java -jar /usr/local/share/ksar/kSar.jar -input ${SARPATH}/${OUTFILE}.txt -outputPNG
${PREFIX}/images/${OUTFILE} -outputPDF ${PREFIX}/pdf/${OUTFILE}.pdf
```

While checking the resulting graphs and looking for tuning opportunities or things that went wrong during the tests I found it very informative if I could look at the same metrics from all nodes at once. So for having an easier overview I also generated a basic HTML file which showcased the same metric for all nodes in one view. For easier evaluation I have put the results into folders named after the start time and the elapsed time in seconds – here is an example showing the last 20 job result folders:

```
# ls -1 01_BASELINE/ | tail -n 20
TeraSort_16G__2014-10-02T00-37-20__307.999_seconds
TeraSort_16G__2014-10-02T11-14-14__323.126_seconds
TeraSort_16G__2014-10-02T21-56-21__321.954_seconds
TeraSort_16G__2014-10-03T08-39-31__313.243_seconds
TeraSort_16G__2014-10-03T20-15-54__338.543_seconds
TeraSort_1T__2014-10-02T02-32-21__24968.600_seconds
TeraSort_1T__2014-10-02T13-44-04__23281.147_seconds
TeraSort_1T__2014-10-02T23-52-04__25786.129_seconds
TeraSort_1T__2014-10-03T11-08-54__25094.860_seconds
TeraSort_1T__2014-10-03T22-17-49__23363.417_seconds
TeraValidate_16G__2014-10-02T00-44-12__34.530_seconds
TeraValidate_16G__2014-10-02T11-21-20__33.650_seconds
TeraValidate_16G__2014-10-02T22-03-26__33.700_seconds
TeraValidate_16G__2014-10-03T08-46-28__32.611_seconds
TeraValidate_16G__2014-10-03T20-23-17__39.798_seconds
TeraValidate_1T__2014-10-02T09-38-09__4599.195_seconds
TeraValidate_1T__2014-10-02T20-21-48__4539.023_seconds
TeraValidate_1T__2014-10-03T07-11-31__4147.745_seconds
TeraValidate_1T__2014-10-03T18-16-48__5992.151_seconds
TeraValidate_1T__2014-10-04T04-56-52__4436.654_seconds
#
```

Each of these folders contains the corresponding logs, sar binary, ASCII performance metrics and the corresponding graphs in PNG and PDF format. There is also a zero byte file named after the runtime comment for easier identification of the job and the index.html referring to the graphs:

```
# ll 01_BASELINE/TeraValidate_1T__2014-10-04T04-56-52__4436.654_seconds/
total 24
drwxr-xr-x 1 root root   176 Oct 19 02:27 .
drwxr-xr-x 1 root root  5090 Oct 19 02:40 ..
drwxr-xr-x 1 root root  7576 Oct 19 02:27 images
-rw-r--r-- 1 root root  5519 Oct 19 02:27 index.html
drwxr-xr-x 1 root root   382 Oct 19 02:27 pdf
drwxr-xr-x 1 root root   756 Oct 19 02:27 sar
-r--r--r-- 1 root root 14794 Oct 19 01:57 TeraValidate_1T__2014-10-04T04-56-52.log
-rw-r--r-- 1 root root     0 Oct 19 02:27 TeraValidate_1T_-_baseline
#
```

## xi. Linux kernel compilation process

For compiling the kernel I have exclusively used the first node. This is how I transferred and extracted the xz kernel source archive to that node:

```
# mount -o rw raspi:/btrfs/kernel /kernel
# tar xf /kernel/linux-3.17-rc4.tar.xz -C /hdfs/kernel/
# cp /boot/config-3.10.0-123.el7.x86_64 /hdfs/kernel/linux-3.17-rc4/.config
# umount /kernel
# cd /hdfs/kernel/linux-3.17-rc4
```

Note that it had to be done only once initially, for the subsequent compilations I have used the existing source files. While being in the root of the kernel source I started a menuconfig session and changed the required parameters. For compiling the kernel itself and the modules I have used the –j4 parameter to utilize all 4 cores of the CPU:

```
# make menuconfig
# make -j4
# make -j4 modules
```

Then I created a subdir for the modules (note that the default /usr/lib/modules path would not work as it is read-only) then I specified this path as INSTALL_MOD_PATH:

```
# mkdir /hdfs/kernel/modules_3.17.0-rc4-build101
# make modules_install INSTALL_MOD_PATH=/hdfs/kernel/modules_3.17.0-rc4-build101/
```

Note that build101 was the very first build, containing the very same settings as the distro kernel. The only difference was that I have stripped it from the unused modules and drivers to speed up the booting. Once this was done I could transfer the resulting kernel and the modules to the Raspberry:

```
raspi:# scp -p athos:/hdfs/kernel/linux-3.17-rc4/arch/x86/boot/bzImage \
   /btrfs/kernel/compiled/vmlinuz-3.17.0-rc4-build102.x86_64

raspi:# btrfs subvolume create /btrfs/modules_3.17.0-rc4-build101
raspi:# cd /btrfs/modules_3.17.0-rc4-build101
raspi:# scp -pr \
   athos:/hdfs/kernel/modules_3.17.0-rc4-build101/lib/modules/3.17.0-rc4-build101/kernel .
raspi:# scp -pr \
   athos:/hdfs/kernel/modules_3.17.0-rc4-build101/lib/modules/3.17.0-rc4-build101/modules* .
raspi:# btrfs subvolume snapshot /btrfs/modules_3.17.0-rc4-build101 \
   /btrfs/CentOS7_Hadoop/lib/modules/3.17.0-rc4-build101
```

Then I have generated a ramdisk with all the required modules for network boot and transferred it to the Raspberry and changed the TFTP setting there to point to the new kernel:

```
# /usr/sbin/dracut --host-only --nofscks --nolvmconf --nomdadmconf --no-early-microcode \
   --xz --kver 3.17.0-rc4-build101 \
   --kmoddir /hdfs/kernel/modules_3.17.0-rc4-build101/lib/modules/3.17.0-rc4-build101 \
   --fwdir /hdfs/kernel/modules_3.17.0-rc4-build101/lib/firmware --ro-mnt \
   --add "nfs network base" \
   --add-drivers "tg3 nfs lockd sunrpc nfsv4 nfs_acl dns_resolver fscache" \
    /hdfs/kernel/initramfs-3.17.0-rc4-build101.x86_64.img

raspi:# scp -p athos:/hdfs/kernel/initramfs-3.17.0-rc4-build102.x86_64.img \
   /btrfs/kernel/compiled/
raspi:# chmod 644 /tftp/centos/netboot/initramfs-3.17.0-rc4-build101.x86_64.img

raspi:# vi /tftp/pxelinux.cfg/default
```

Note that the TFTP server does not need to be restarted after the configuration change since it is running via xinetd and being start automatically as soon as a request hits the UDP port 69 (standard TFTP port).

## xii. Scalability – adding a new node

Due to energy saving reasons in the very initial phase I have started the function tests on less number of nodes. So I had the opportunity to test the scalability of the scripted framework and of the stateless Hadoop system. Here are the steps I had to take in order to extend the cluster by a new hardware node:

1. Add hostname to DNS config – making sure reverse lookup works as well
2. Add MAC to DHCP config
3. Boot new node manually – this will boot up into emergency mode due to missing HDFS backing store
4. Prepare local disk (see subchapter Persistent store)
5. Reboot
6. Add hostname to /btrfs/hadoop-conf/slaves on orchestration server
7. Add hostname to processes_sar_data.sh script

# Chapter 5) Results and Analysis

After describing the process of installing the hardware, setting up the environment and finally writing the testing framework scripts, I am presenting the results of the research project in this chapter. However to better understand the results let's see the methodology behind these tests first!

The key resources in a system are CPU, RAM, I/O and network. In order to get the best result during the tests one has to make sure that these resources are utilized as intensive as they can be. That will ensure the highest throughput from the Hadoop cluster perspective. As soon as one of these resources is overloaded, it starts to become a bottleneck and ends up dragging down the overall throughput. In the most cases this can be seen in a graph as the specific resource hitting the maximum capability of the device and/or as the under-utilization of the other three resources. Of course I have to add that there will always be a bottleneck at the "weakest chain" though we have to aim at the roughly equal usage of these resources.

As with every performance tests at first we need to get a clear picture about the capabilities of the initial configuration so that we can compare the later results to it. In order to get that initial view I have conducted the baseline performance tests right after the successful functional tests and just before doing any alterations to the system.

Regarding the RAM capabilities there wasn't really much to test as I already knew the capacity of the built in DIMMs. For the CPU that was roughly the same; the boot-time BogoMips value can give us a clue about the CPU capabilities or one can look up official benchmark values e.g. in the PassMark catalogue [26].

On the other hand; with the networking it gets more interesting as in most cases it is not just throughput but also latency what we care about. A good network testing tool is netperf so I have used that. By connecting the nodes to each other with a datacenter-grade Gigabit switch I could expect low latencies. However by adding the orchestration server through a SoHo router – even if configured on the same network – I have introduced higher latency. And the 100MBps interface which is available on the Raspberry did not do any good to the network throughput either. However the Raspi server is utilized mostly at boot time of the Hadoop nodes. Once the required binaries are cached on the worker nodes it does not affect the test results. As I've already mentioned I did the cache warming with the short BBP jobs before doing the TeraTests.

Regarding the disk throughput one could do basic read and write tests with the dd tool or utilize a more advanced tool like iometer, iozone or fio. The advantage of these tools over dd is that one could specify more than one test profile and run these as parallel workloads while with dd one would have to manually run multiple instances as it can only generate a sequential load.

The disk throughput is measured in IOPS which stands for I/O per second. That is a plain number without any additional unit and it tells how many I/O commands could be sent down the storage stack within one second. Another important factor is the size of the I/O; a typical value would be 64kbyte. The following formula shows the connection between I/O size, IOPS and throughput:

**[Throughput] = [IOPS] x [I/O size]**

So if e.g. a system can achieve a throughput of 100 IOPS with 64kbyte I/O size that would conclude to a throughput of 6.4Mbyte/sec. Another important storage metric is the latency. Generally speaking the higher the load on the storage stack is, the higher latencies we will see on the storage side. As in case of Hadoop the interactivity of the system is not much of importance, we don't really have to care about the latency: we have to tune the system to reach the highest possible throughput.

Before we start evaluating the metrics it is very important to understand the anatomy of a Hadoop job. A usual Hadoop job consists of two phases, these are called Map and Reduce phases. (Figure 4) First the input is split to multiple chunks of data then the mapper processes these and generates key-value pairs. The Reducer gets these key-value pairs and aggregates them which will conclude to the result and these get written back to HDFS.

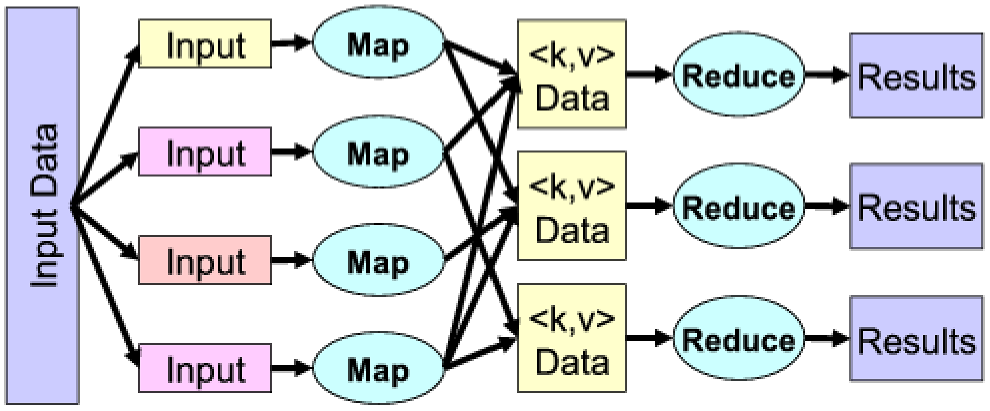


**Figure 4: Anatomy of a MapReduce job [28]**

There could be some overlap between the Map and Reduce phases (see [24]) but generally speaking the second part of the job – the Reduce part – has slightly different performance characteristics compared to the pure Mapper phase at the beginning. These can be clearly distinguished in most of the performance graphs. In the following I will explain each metrics and my observations about them. Unless otherwise stated the following graphs are from the baseline test “TeraSort_1T.job - 2014-10-02T23-52-04”.

## i. CPU metrics

One of those resource metrics is the CPU utilization, shown as *% used cpu* in the graph as a percentage where 100% means a fully saturated CPU (see Figure 5). From the graph it can be seen clearly that the CPU has 4 cores. In some cases some of these cores were fully utilized while the others were idle and this generated the “stepping” in the graph at 25%, 50% and 75% respectively. In the upper section one can see a distinction between user and system processes. The system portion is responsible for keeping the system running. If that is over 10% of the CPU time then something is going wrong, it means we spend too many cycles just to keep the OS running. There was a Linux kernel bug related to transparent huge pages where the compaction of memory pages took unreasonable amount of cycles and this was indicated as high system CPU time values, see [27].

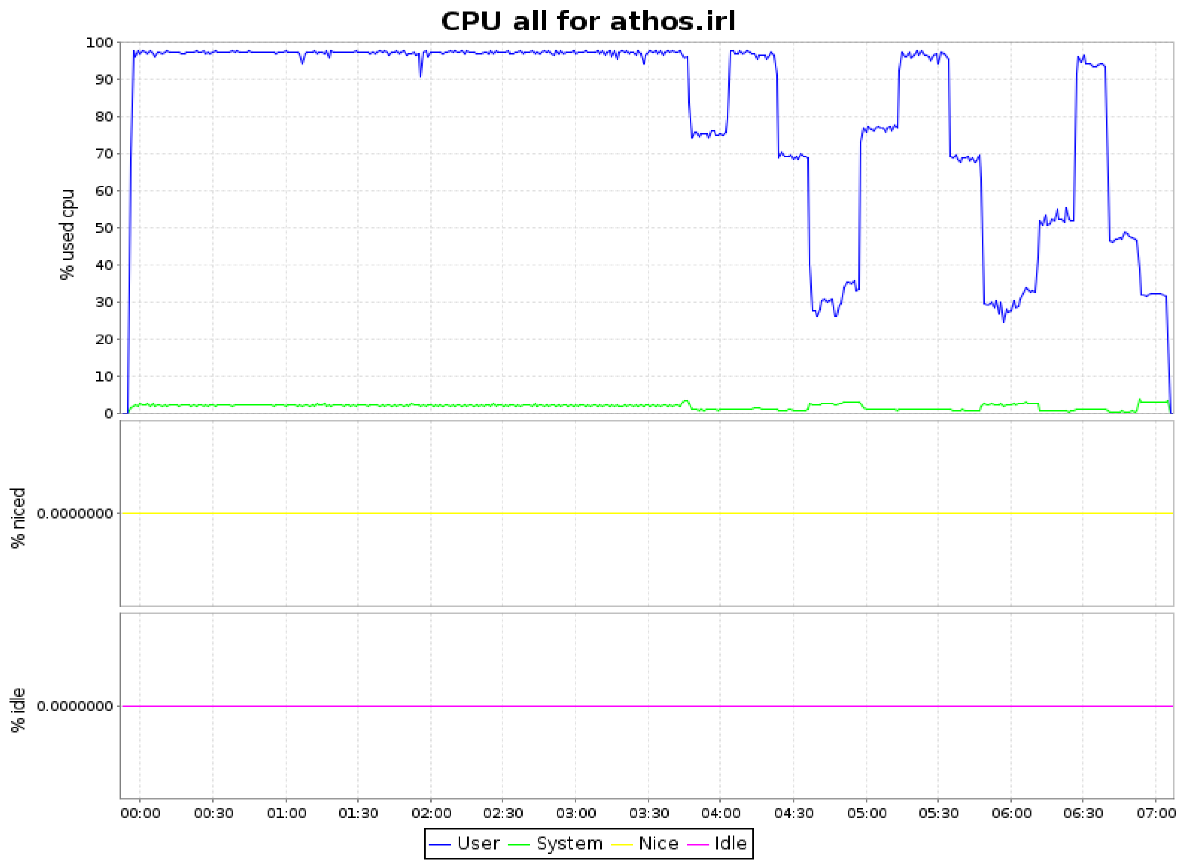


Figure 5: CPU related metrics by ksar

As it can be seen in the above graph, the other two metrics are “Nice” and “Idle”. Alas the ksar version used was not 100% compatible with the sar data collector so these are shown with zeroes in the graphs. The “Idle” would show the percentage of time where the CPUs were idle and the system did not wait for any I/O. The “Nice” value would show the percentage of CPU time spent on prioritized processes (these being with a lower than default nice value). Since all the processes have been started with normal priority, the nice percentage is zero for all the tests.

## ii. CPU context switches

The next graph (Figure 6) is also CPU related and shows the number of CPU context switches per second. A context switch happens when a specific CPU core stops processing a thread, stores its state and resumes another thread. A context switch is generally very resource intensive as it can involve changing the contents of the CPU cache and the TLB. The key tunable here is the number of containers (see the Hadoop parameters subchapter) as that will influence the number of processes per node. Since that parameter was constant I have seen similar context switch values throughout the tests.

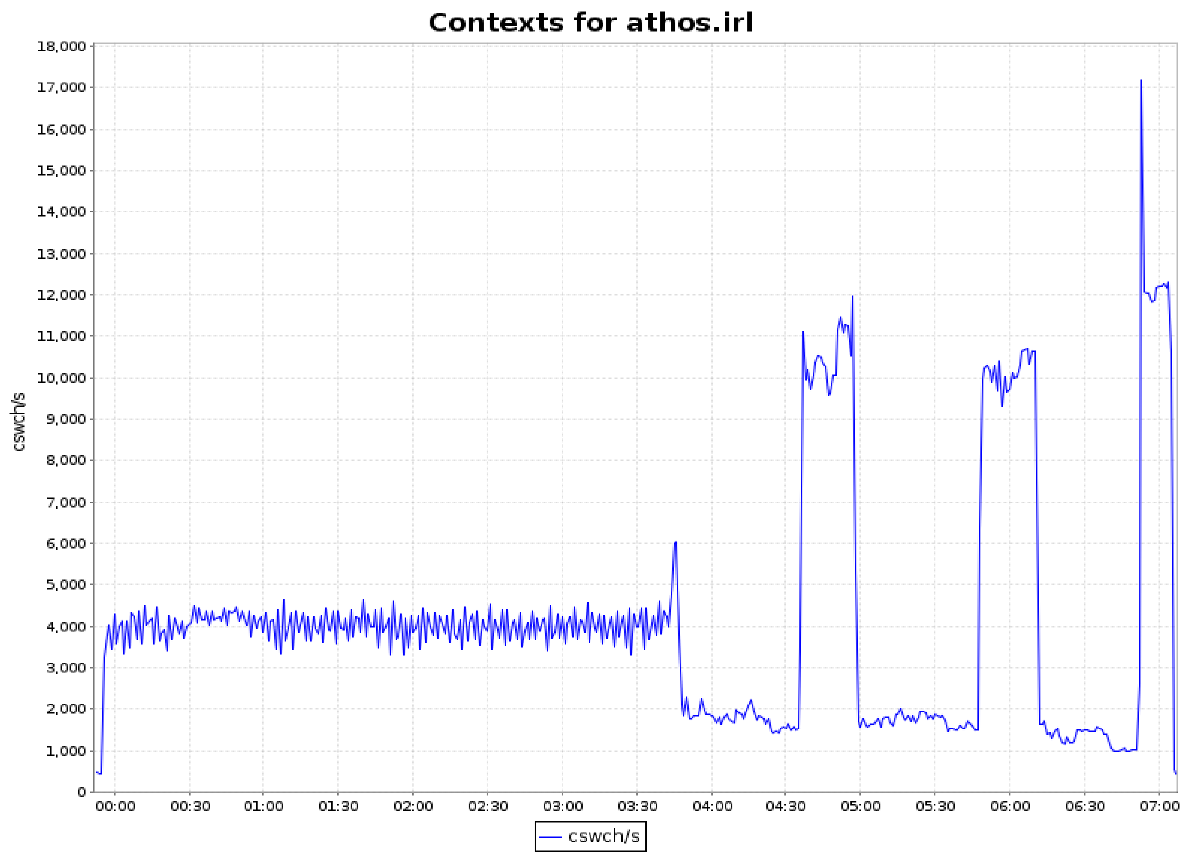


**Figure 6: Number of CPU context switches per seconds**

### iii. System load

The first graph in next figure shows the run queue size (*runq-sz*). The run queue consists of processes which are waiting for the CPU and the graph shows the length of this queue for every given time. It is the task of the process scheduler to determine which process gets executed next. Depending on the scheduler type this can be done based on a combination of multiple variables such as elapsed wait time and priority (nice value). For Hadoop if the run queue is higher than twice the number of CPU cores then the CPU is over-utilized, see the sizing in the previous chapter.

The second graph in green is showing the number of tasks in the task list (plist-sz). In Linux the terms process and task are interchangeable. However the difference compared to the run queue is that a process waiting for I/O would not show up in the run queue but would be visible in the task list. From the graph we can clearly see that there were around 600 tasks overall but out of them only ~10 were in runnable state waiting for CPU. (runq-sz)

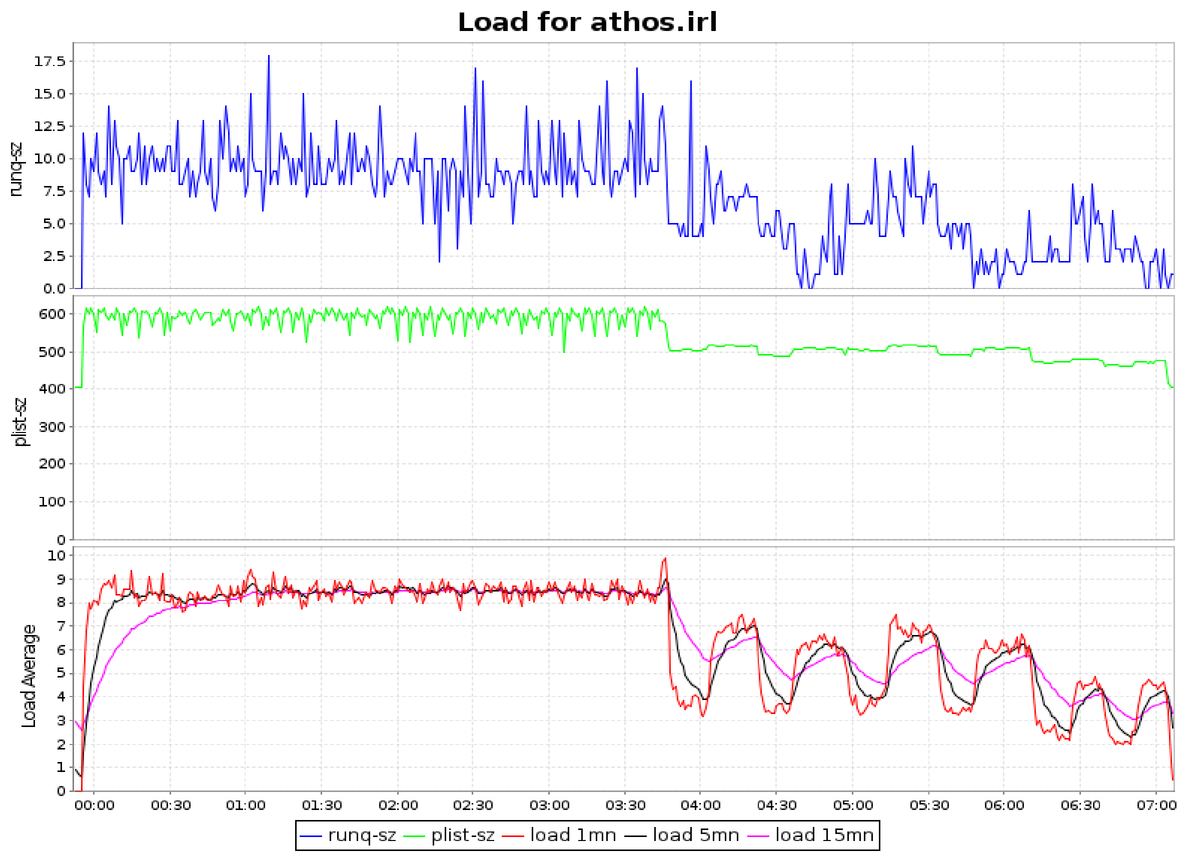


Figure 7: Queue length and load average

The third figure shows the load average for 1, 5 and 15 minutes respectively. That shows the average number of processes in runnable or running state, plus the number of tasks in uninterruptable sleep for the given interval. There can be spikes visible in the runq-sz graph what are smoothed out in the 5 minute load average so this is a better indication if there is a CPU bottleneck.

### iv. Processes created per second

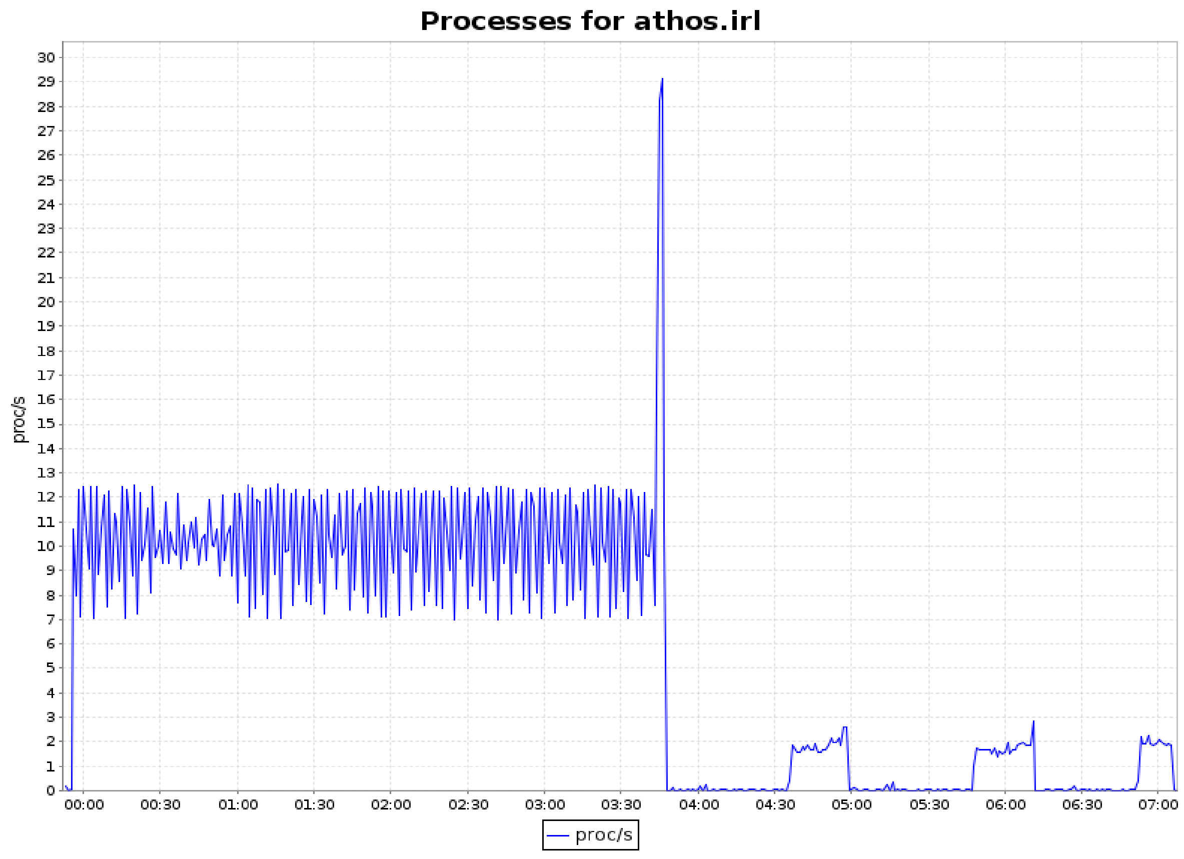


**Figure 8: Number of processes created per second**

The above graph is showing the number of processes created per second. There is a huge spike clearly visible at the start of the Reduce phase. Generally speaking the Mapper part is busier due to higher number of Mapper processes created. In some cases I have seen a drop in proc/s values at some stage in the Mapper phase which always led to bad results.

### v. Disk I/O statistics

In Figure 9 I present disk I/O related statistics. In the first graph the number of transfers can be seen, in other words this is the IOPS value. In the third graph we can see the number of read and write operations, the sum of these gives us the total IOPS in every given second. As we can see the Mapper phase is mostly read-intensive only writing the key-value pairs every now and then. On the other hand in the Reduce phase we store the results in multiple write spikes.

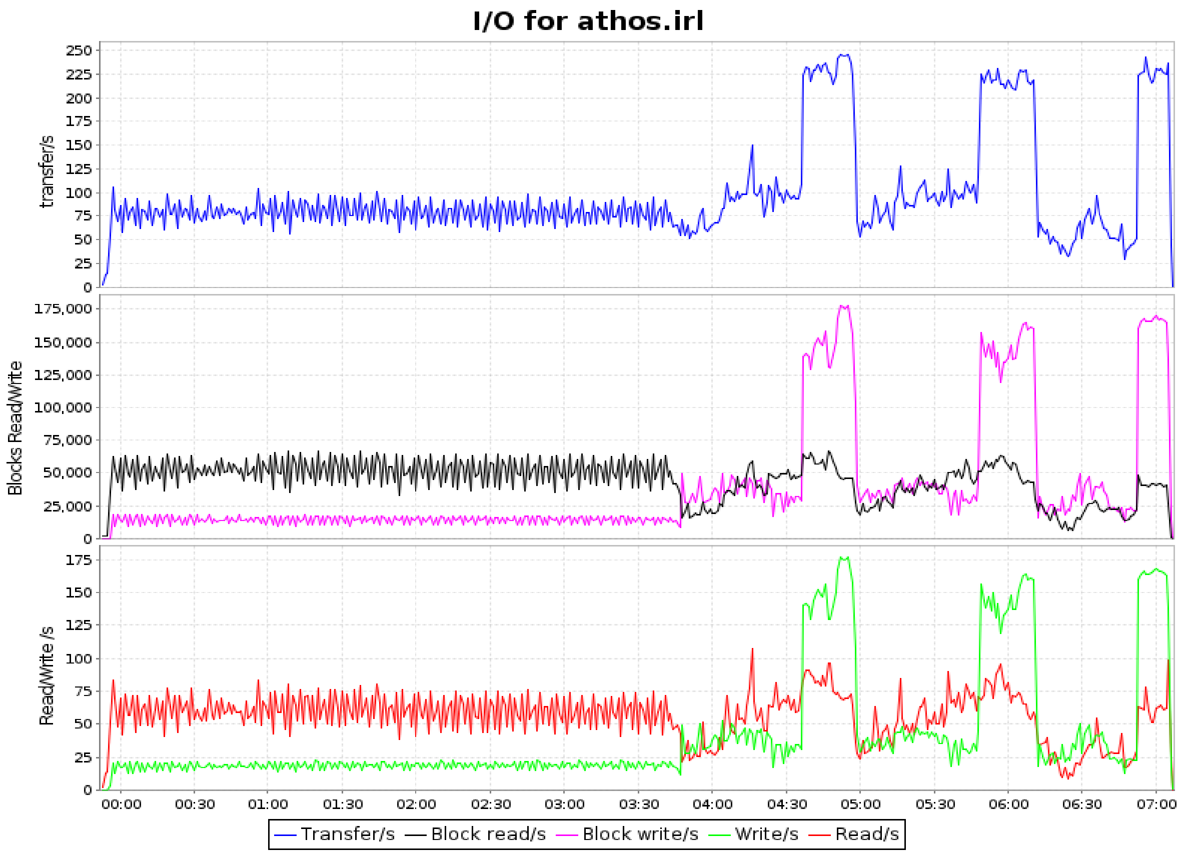


Figure 9: Disk I/O graph generated by ksar

The second graph shows the blocks read and written per second. A block is 512 bytes (sector size). Remember the formula shown earlier at the beginning of this chapter, this is basically the throughput which equals to the product of the IOPS and I/O size. Based on the above values we can calculate the I/O size for the above TeraSort job. Here are the sar results in plain ASCII format:

```
10:19:35          tps     rtps     wtps   bread/s   bwrtn/s
10:20:35         2.77     2.50     0.27   2057.20      3.87
10:21:35        13.28    13.20     0.08   1792.93      0.93
10:22:35        14.13    14.03     0.10   1838.80      1.33
10:23:35        49.38    39.92     9.47  33144.93   6636.80
10:24:35        89.50    69.05    20.45  64269.73  17852.40
10:25:35        72.73    58.57    14.17  54001.87  10180.00
```

Let’s calculate with the last line! The IOPS value was 72.73 which concluded to 64181.87 blocks transferred (bread + bwrtn). The quotient of 64181.87/72.73 equals to 882.46, that is the number of blocks transferred. Multiplying it by 512 bytes (block size used in sar) we get 452 kbytes as average block size for the TeraSort job. That is completely normal as we also have the

NameNode on this host, logging is enabled etc and all these processes use different I/O sizes. For more info on the TeraTest characteristics see [29]. As opposed to that, a TeraValidate is a 100% read and for TeraGen I've found almost a pure 100% write with an I/O size of 512k (~300 IOPS with 300k blocks written) and hence a throughput of 150 Mbyte/sec:

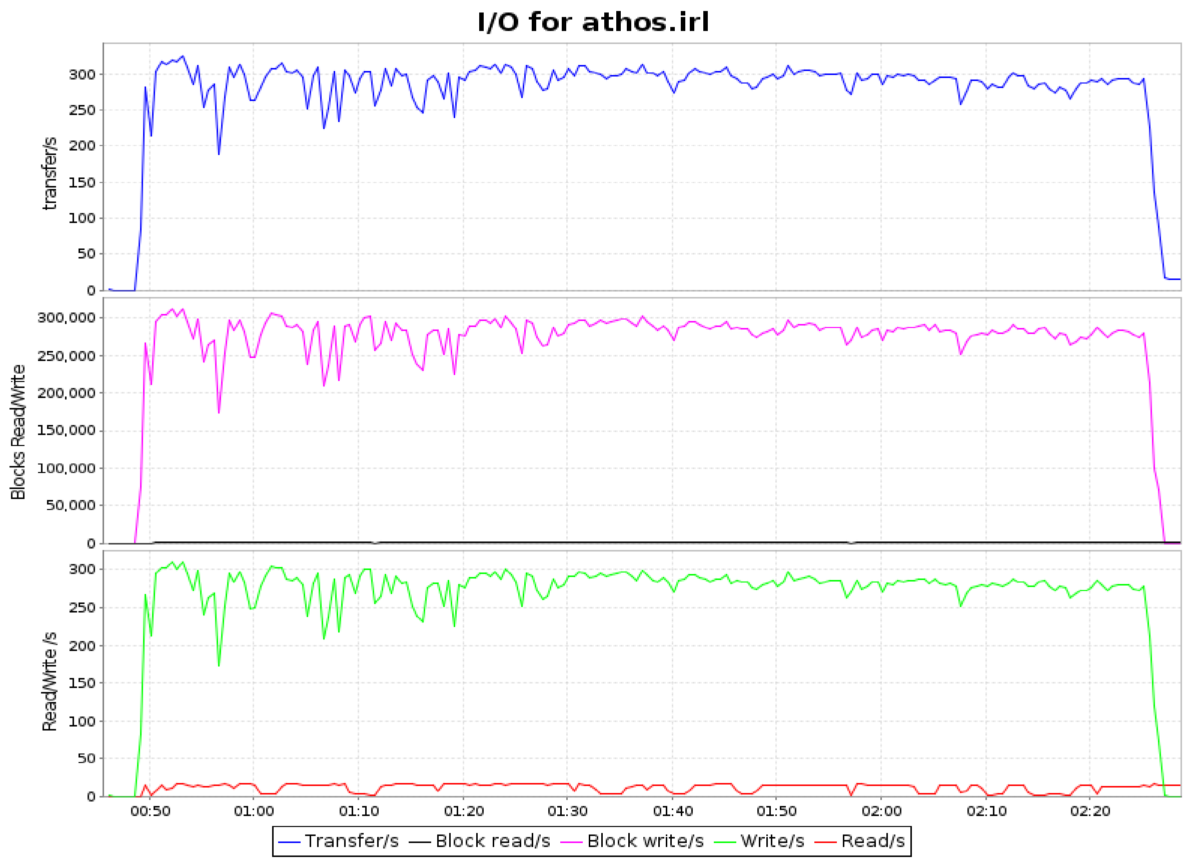


Figure 10: I/O statistics for TeraGen_1T @ 2014-10-02T00-45-40

## vi. Memory statistics

The below graph displays the memory statistics throughout a TeraSort job with one Terabyte of data sorted. The *memfree* value shows the amount of free memory. Note that this value highly depends on the state of the buffer cache. On a recently booted system there is a higher amount of memory available. The subsequent I/O operations are being cached in the RAM and the contents of the buffer gets written to disk in regular intervals, so in the end the amount of free RAM tends to be zero. However this is normal functionality and the part of the buffer cache will be flushed automatically, should there be a memory constraint.

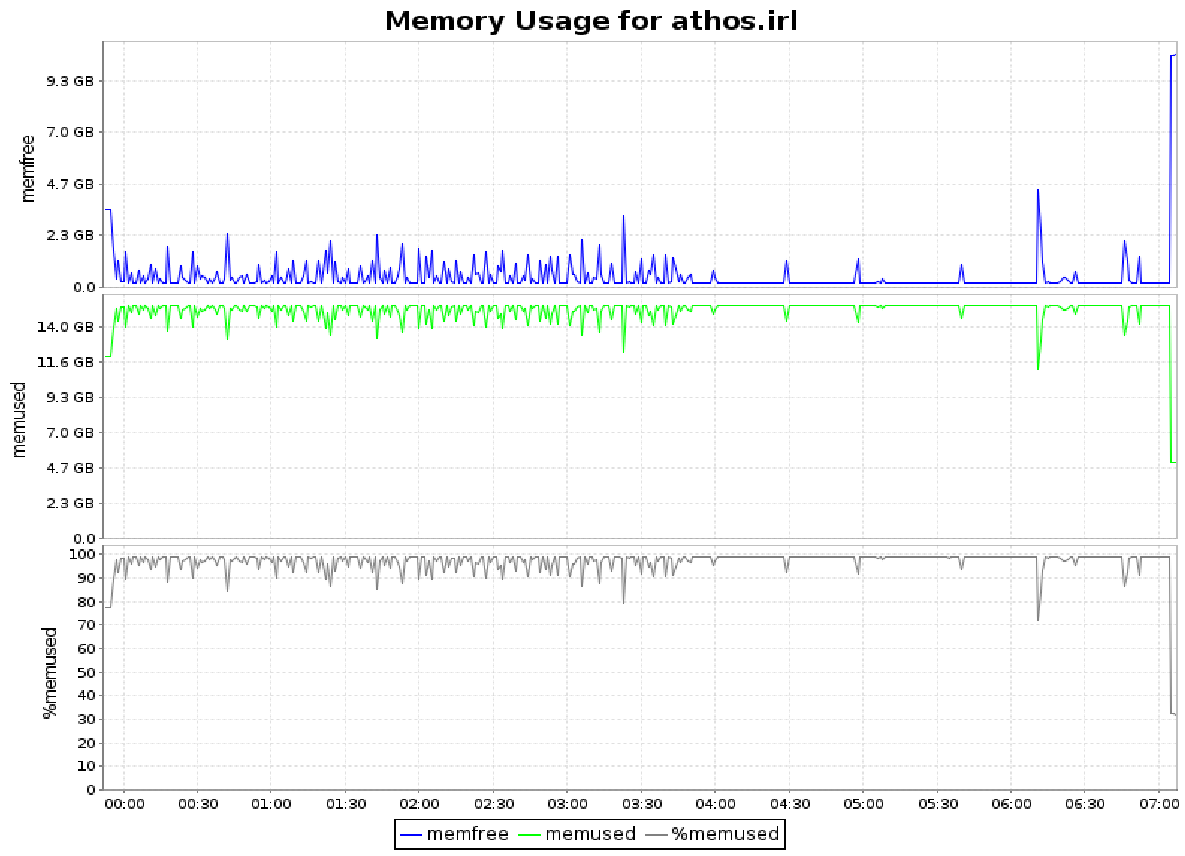


Figure 11: Free and used memory - graph generated with ksar

The two graphs below are showing the amount of used memory in absolute and relative values (as percentage). As we can see the above TeraSort job was not the very first job in the batch processing hence the remaining available RAM was already being used by the buffer cache.

This can also be seen in Figure 12 where the first graph represents the amount of buffer space used by the kernel and the second graph shows the amount of RAM being used for caching data. The buffers contain the filesystem metadata while the cache is used for storing the actual contents of the files. [30] It is clearly visible that due to the higher memory demand from the processes the cache had to be decreased for the Reduce phase.

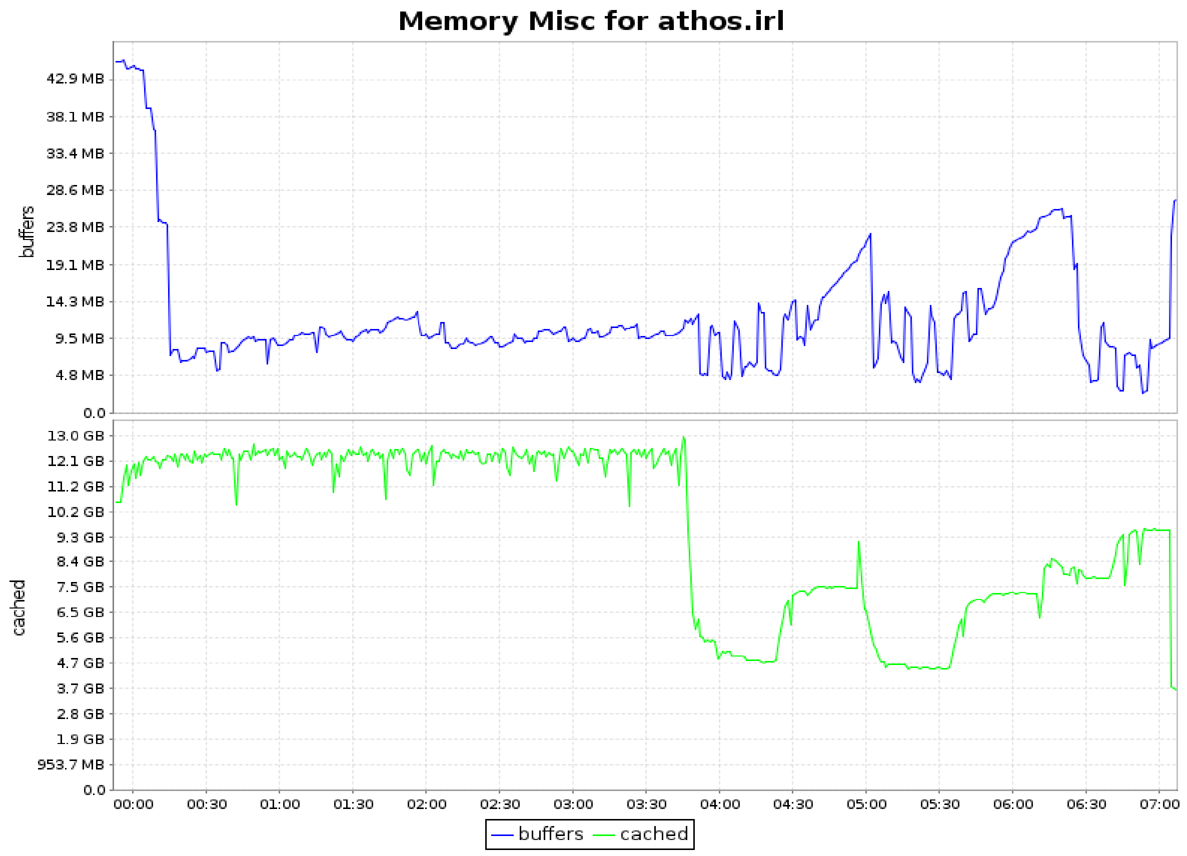


Figure 12: Amount of memory used as buffer and cache

In Figure 13 paging related information is shown. The *frmpg/sec* graph is representing the number of memory pages freed per second. If the value is negative then the system had to allocate new pages. A single memory page is of 4 Kbytes in size for a 64bit Linux architecture, unless *hugepages* were in use.

The next two graphs in green and red are showing the amount of page changes for buffers and cache respectively. Note that these are independent from the *frmpg/sec* value as these are showing additional memory pages. Generally one can observe a higher cache activity in the Mapper phase. Again positive numbers mean that pages have been freed while a negative number indicates newly allocated pages.

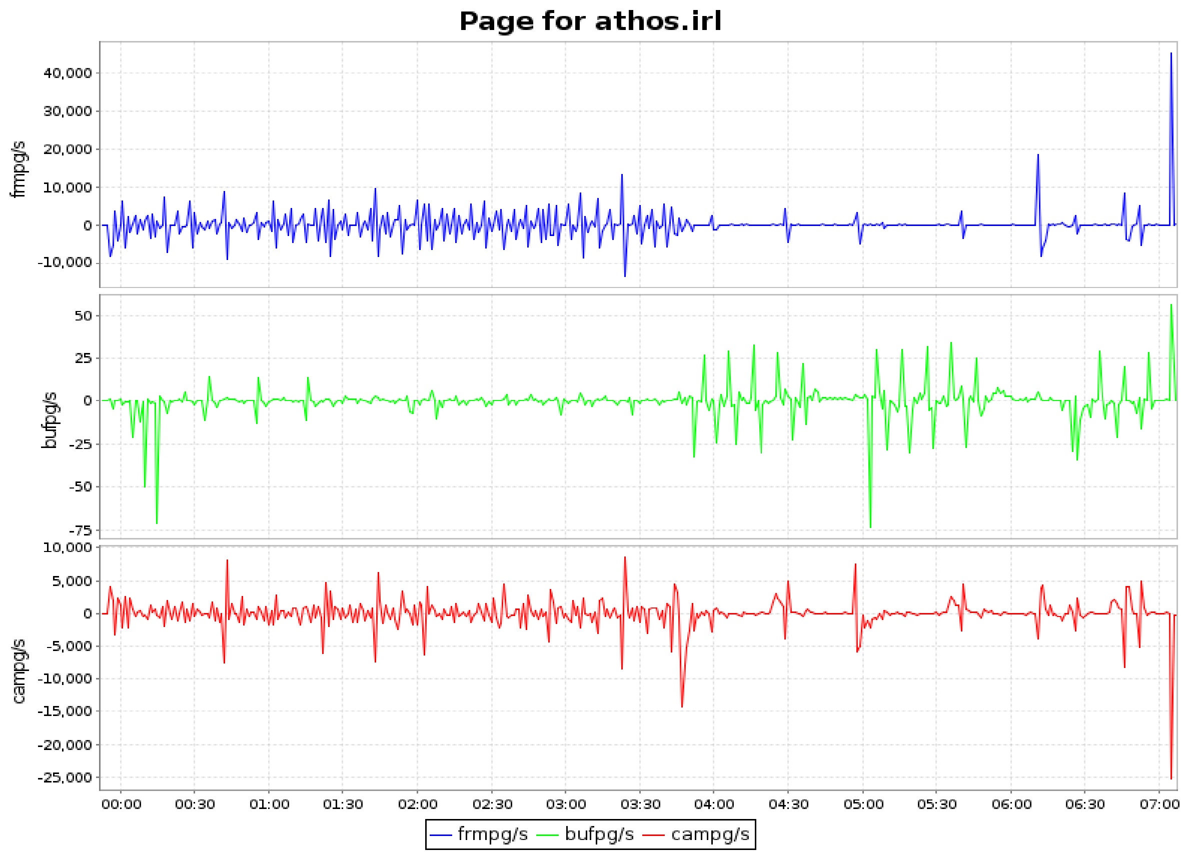


**Figure 13: Change in the number of memory pages / buffers / cache**

The first graph in Figure 14 displays the number of kilobytes the system paged in and out from- and to disk per second. During the Mapper phase more and more pages got read from the disk whereas later in the Reduce phase those pages got flushed to the disk at the end of each Reduce process.

The second graph in red shows the overall number of page faults (minor + major) per second. A page fault occurs when a process would like to access a page which is not loaded into memory. A minor page fault does not necessary generate disk I/O. The graph would show the number of major page faults with *majflt/s* for which a disk I/O is inevitable. This is not visible due to compatibility issues between ksar and sar though the values can be read by opening the sar ASCII file.

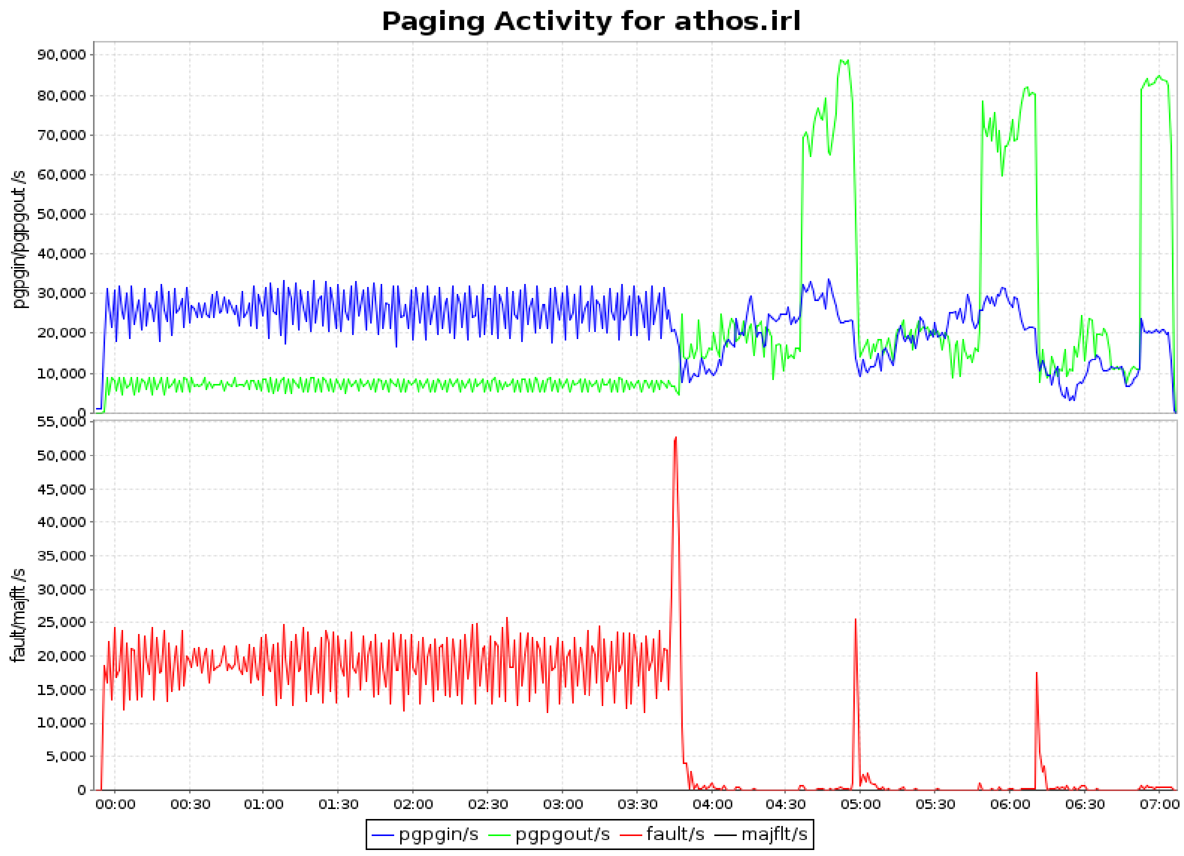


Figure 14: Paging activity and number of page faults per seconds

## vii. Ethernet statistics

The ML310e has two onboard NICs out of which one was used for connecting the nodes with each other. That interface was enumerated as eno1 at the Linux boot process. In the following I will discuss the various metrics of this eno1 adapter and statistics for the loopback adapter are also useful to look at.

The first graph in Figure 15 is showing the sent and received number of Ethernet packets per second for the loopback adapter. The second graph shows the same transmitted and received packets but in kilobytes. The loopback interface is a local interface connected to the IP stack and ensures connectivity between the processes without having to use UNIX sockets. The third graph in Figure 15 is showing compressed packets received and sent. Since the loopback connectivity is only constrained by the CPU resource, there is no gain to compress the local traffic so this is shown as constant zero for the loopback adapter. The last graph shows the number of multicast packets received per second; that is also zero for the loopback adapter. The Java processes of Hadoop are highly utilizing the IP stack even for local connections; that is clearly visible from the graph.

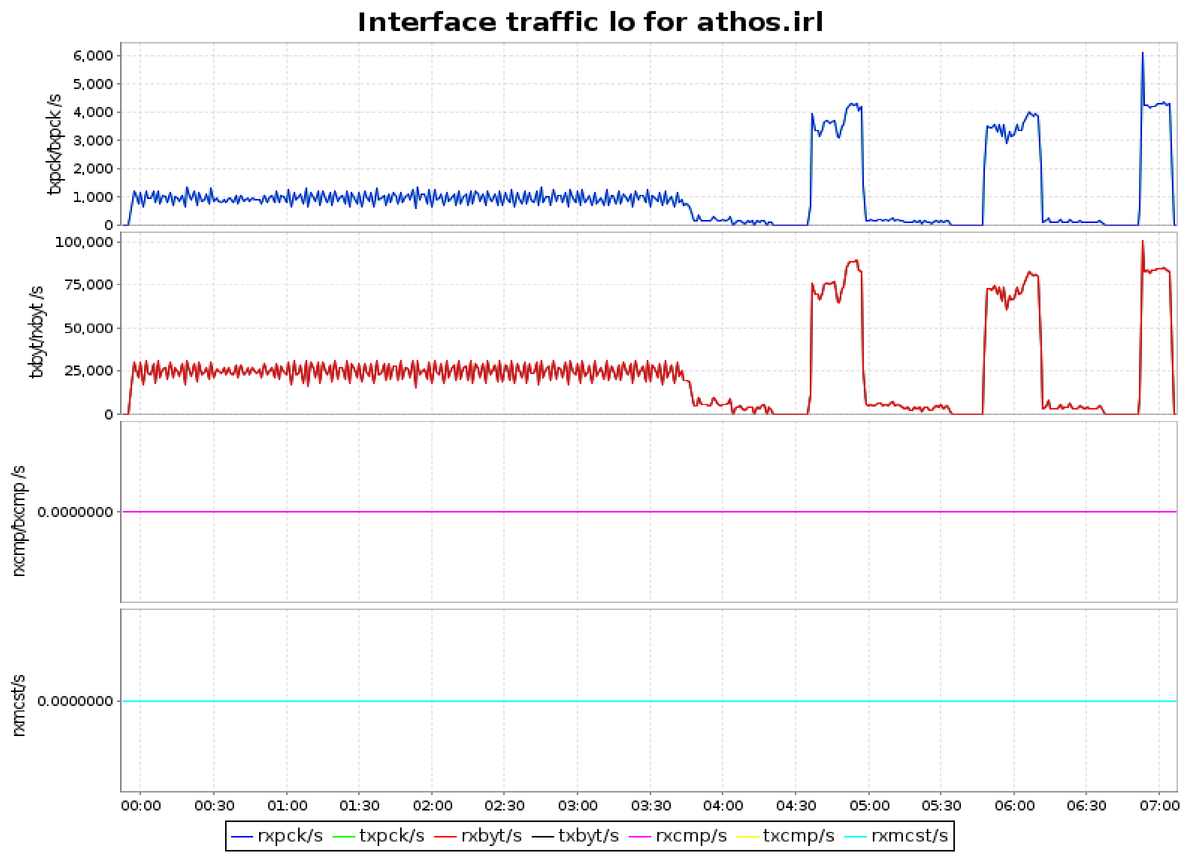


Figure 15: Loopback interface traffic

In Figure 16 the same metrics can be seen for the onboard Ethernet adapter called eno1. While the loopback statistics are strictly showing connections within the same host; behind the eno1 metrics there are mostly external connections. Note that for the Mapper phase Hadoop is considering data locality so the HDFS blocks local to the worker node are preferred for processing. On the other hand in the Reduce phase the results have to be sent to the corresponding Reducer which generates higher inter-node traffic.

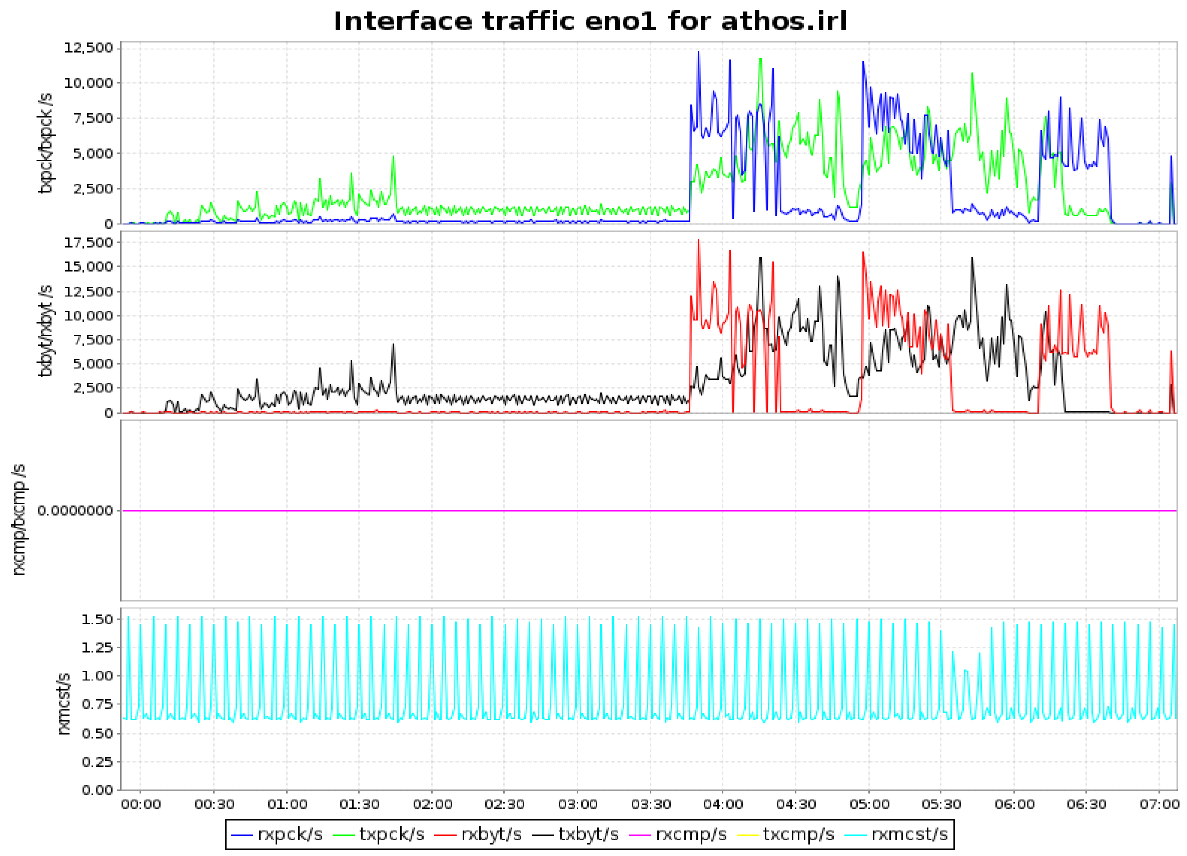


Figure 16: Interface traffic for the default Ethernet adapter

According to the third graph in the above figure there were no compressed packets. Regarding the light multicast traffic received from outside (fourth graph) unfortunately I have no idea from where these are originated. From the documentation I've found for Hadoop I presume that these do not have to do with Hadoop itself at all. For a further analysis one would have to capture the actual Ethernet packets and see the origin of the multicast payload.

I have encountered the best test results for jobs where the amount of sent and received kilobytes were nicely aligned – this is probably a sign of data locality where the underlying processes were still communicating over eno1 instead of the loopback adapter.

In the first graph of Figure 17 the total number of sockets can be seen. The number of IP fragments was shown as zero for every test. This indicates that every payload could fit into the standard 1500 bytes MTU (Maximum Transmission Unit) size. The green graph with *tcpsck* is showing the number of TCP sockets which were currently in use on the system. There seems to be no correlation between this metric and the amount of time the job was running.

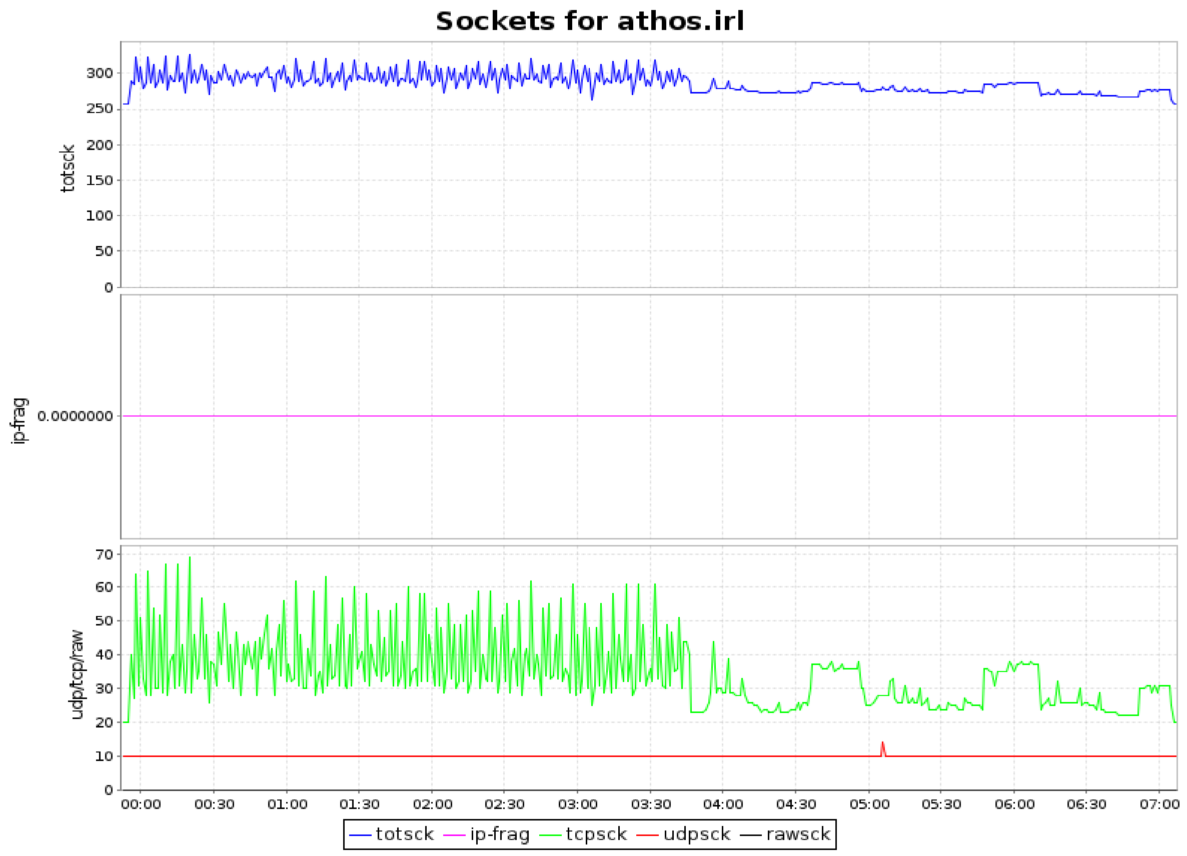


Figure 17: Total number of opened sockets and number of sockets by protocol

## viii. NFS client statistics

In Figure 18 the NFS client related statistics are shown. These are only interesting because the servers were running with a read-only NFS root mounted from the orchestration server. In case of NFS all the filesystem operations are conducted via RPC calls (Remote Procedure Calls). A Remote Procedure Calls is an atomic operation by which the client issues a request to the server. The graphs are showing the number of RPC calls: total number, *read*, *write*, *access* and *getattr*. As the root filesystem was mounted read-only there were no write operations at all.

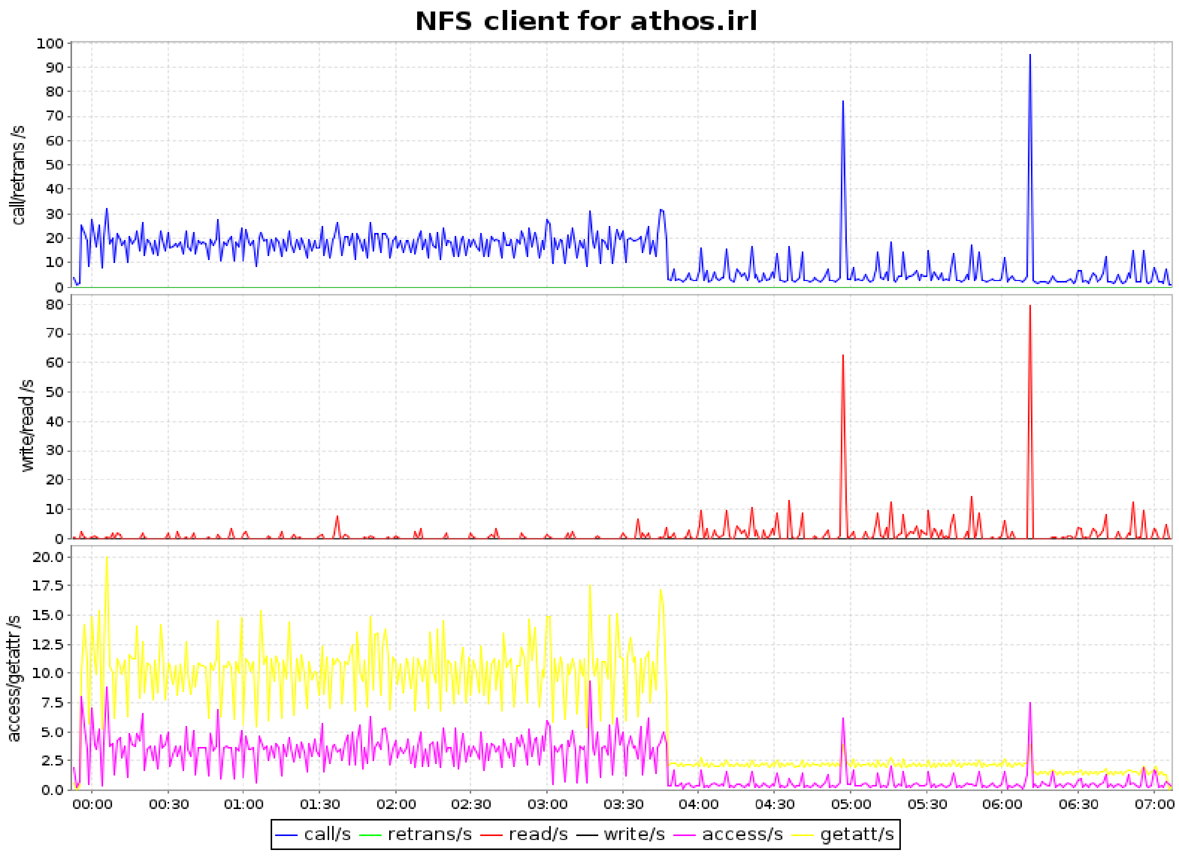


Figure 18: Number of RPC calls by type and total RPC calls

## ix. Results

The results of the actual tests can be found in Appendix D – TeraTest results with 16 Gbyte dataset and in Appendix E – TeraTest results with 1 Tbyte dataset. The following tests have been conducted over the course of 3 months:

1. Baseline

   resetting BIOS and erasing local disks

2. BIOS tweaking

   Power Management was set to Maximum Performance

   Power Regulator was set to Dynamic Power Savings Mode

3. Build102

   cleaned kernel tree: removed unused device drivers

   turned off debug features

auditing & SELinux turned off

virtualization support has been disabled

module signing disabled

4. Build103

   changed from the default deadline to the noop I/O scheduler

5. Build104

   changed to CFQ (Completely Fair Queueing) I/O scheduler

6. Notranshuge

   kernel build103 used

   transparenthugepage compaction disabled see [27]

   test invalidated due to datanode issues on athos

7. Fsopts

   tweaking the mount options of the backing store

   noatime,nobarrier,commit=60,data=writeback

8. Notranshuge2

   realized issues in test no. 6 so running again

9. Further tests with only 16 Gbytes of dataset

For every test case I have recorded the various metrics for the system resources and logged the elapsed time for the jobs. My goal was to get the lowest possible elapsed time. I have defined a runbook with the same set of tests 5 times in a loop. The following jobs were running in every single iteration:

```
BBP_1_16384_16
BBP_1_16384_64
BBP_1_100k_8
BBP_1_100k_32
TeraGen_16G
TeraSort_16G
TeraValidate_16G
TeraGen_1T
TeraSort_1T
TeraValidate_1T
```

After running the tests I’ve reviewed the logs. For some jobs I’ve found that one or more container has failed with NotReplicatedYet exception. This is due to resource shortage and resulted in a resubmit of the container. Such failures did not have an effect on the overall Hadoop job yet these caused prolonged runtime so I decided to mark these cases as outliers and do not take them into account while evaluating the results. Jobs encountering this NotReplicatedYet exception have been marked with bold font on grey background in the appendix.

I also wanted to measure the quality of the tests, by this I mean the consistency of the results. For that I have calculated the relative standard deviation which shows how densely the values are located around the mean value. [31] I got fairly consistent execution times for the TeraSort jobs (generally less than 5%) however the TeraGen and the TeraValidate jobs were showing a higher relative standard deviation. In those cases the resulting dataset does not seem to align according to the normal distribution. Those runtimes rather seem to be clustered around some key values. In order to prove this I would have to conduct a significant amount of tests and apply K-means clustering to the dataset with e.g Rapidminer.

In the initial TeraValidate tests the frequency of the data collection was higher which turned out to be overkill. After raising the SADC_INTERVAL parameter from one second to 30 seconds I still got valuable data and it turned to be just enough to get meaningful results. However I did not want to retake the initial tests so this is the reason for the “hairy” baseline TeraValidate test.

Regarding the two dataset sizes; normally the expected effect of a change is that it would cause roughly the same performance increase for both the 16G and the 1 Terabyte jobs. However we should take into account that the cluster has an overall 64 Gbytes of RAM so in case of the 16 Gbytes tests the whole dataset could fit into the memory. This explains why we can see some differences in the results of the tests between the two dataset sizes.

As I highlighted earlier the aim of the research was to reach the lowest possible runtime specifically for the TeraSort job as that means a similar load pattern to real-world usecases. From the storage perspective the TeraGen means a pure write workload while the TeraValidate phase is a 100% read workload. On the other hand the TeraSort is a complex job, stressing with both read and write characteristics. By the end of the job the same amount of data gets written as the input data read.

The elapsed time for the TeraSort jobs can be seen in Figure 19 and Figure 20, for an overview of all three TeraTests including TeraGen and TeraValidate see Appendix F – Overview of results.

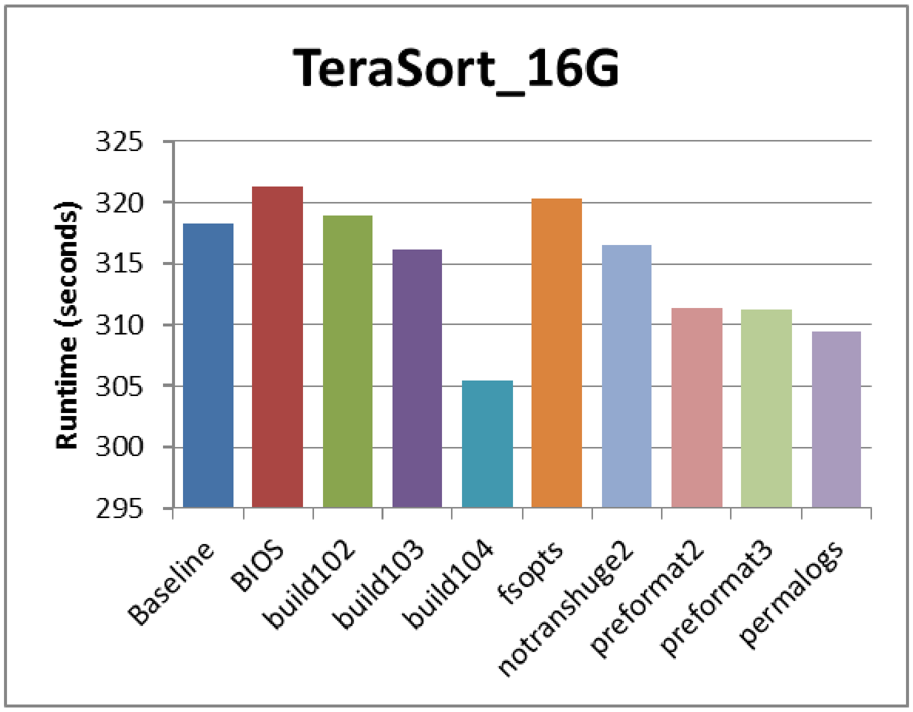


Figure 19: TeraSort elapsed time with 16Gbytes dataset

As it can be seen above; in case of 16 Gbytes of data I've got the best results with the test case called "build104", the improvement was 4% (see Appendix D) compared to the baseline. In that scenario I've used the CFQ (Completely Fair Queueing) I/O scheduler. In the latter "performat" tests I have only extended the time for settling of the services before starting the Hadoop jobs, in any other aspects it was the same as build103fsopts. (System running with build103 kernel and filesystem options applied). The extended time has significantly improved the stability and reproducibility of the tests. The tests with the kernel build103 have delivered the second best results.

In Figure 20 the same tests can be seen, however I have skipped the latter "preformat" tests due to time constraints. It is visible that the tests with the kernel build103 and build104 resulted in lower runtime and therefore show an improvement compared to the baseline tests. With this bigger dataset I got better results with the build103 where the noop I/O scheduler was in use. As

I noted earlier this could be due to the fact that in case of 16Gbytes the whole dataset could fit into RAM.

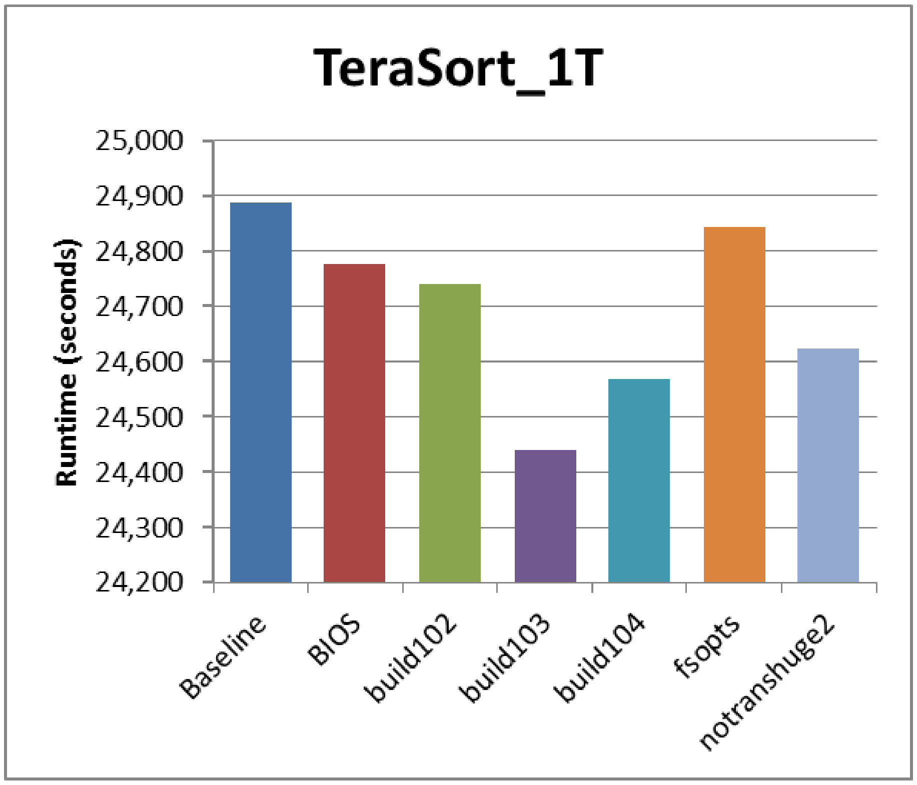


**Figure 20: TeraSort elapsed time with 1Tbytes dataset**

# Chapter 6) Conclusion and Future Outlook

The standard inbox kernel is set to the deadline scheduler. According to my results we can achieve even up to 4% performance improvement by changing the I/O scheduler to either noop (build103) or to the CFQ (build104) scheduler. This improvement could be achieved on a 16Gbyte dataset. By crunching a larger dataset (1 Tbyte) these improvements were 1.8% and 1.28% respectively. I feel that the expected 2-3% improvement has been achieved as I referred to in Chapter 1).

For data generation (TeraGen) the baseline seemed to deliver better results than the subsequent optimizations. This is probably due to the different nature of the workloads. The TeraGen job means a 100% write for the system whereas the TeraValidate is 100% read and the TeraSort being a mixed workload. For a typical Hadoop workload the data loading either takes place gradually over time or there is a single load at the beginning, therefore I would propose the settings from build104.

Interestingly enough I could not reproduce the benefits of the filesystem mount parameter changes initially. The “preformat” tests conducted later with a 16Gbyte dataset could indicate some improvements over the build103 test; however I could not see this improvement at the very first fsopts test. This is probably due to the relatively low number of iterations, a loop of 5 tests is not enough to draw consequences and it can merely show us the directions for further testing.

Overall this research project turned out to be very useful at least from my own personal perspective as I’ve gained a lot of experience in Hadoop and performance tuning in general. I hope that the proposed architecture and the scripting framework could be useful for other Hadoop related projects. One key takeaway from my research is that with less powerful or lower number of nodes one should better aim for a smaller dataset and rather take more tests (even 100s). That would help identifying outlier tests and the consistency of the results would be improved.

Regarding consistency I have seen that in both the TeraGen and TeraValidate phases with the 1 Terabyte dataset the relative standard deviation was 10% which is very high. However in most cases I had a feeling that the execution time does not really align to the normal distribution;

I have seen potential groupings along specific values. To further reseach this area I would have to conduct more tests potentially over 100 iterations.

Now that I have the hardware I will definitely do further experiments on the system. The storage subsystem turned out to be the bottleneck as I have expected. So a major improvement could be reached by populating all the 4 disk bays in each server and that would make it a meaningful Hadoop setup. As the workload is nearly sequential, an SSD backend is not required at all. This study has only focused on the OS specific tunables however I would expect a significant improvement by tuning the Hadoop configuration itself.

As another direction for future tests I could imagine experimenting with multiple parallel jobs; that would probably spoil the sequentiality of the overall workload and might even justify the use of SSDs. In case of SSDs I would also test a system with multiple HDDs and a single SSD for caching. Thinking about a bigger Hadoop cluster I could separate the NameNode and the TaskTracker from the workers and I could focus on tuning them separately.

## References

[1] Aida, Kento, Omar Abdul-Rahman, Eisaku Sakane, and Kazutaka Motoyama, 'Evaluation On The Performance Fluctuation Of Hadoop Jobs In The Cloud', IEEE Computer Society, (2013), 159-166.

[2] Cho, Joong-Yeon, Hyun-Wook Jin, Min Lee, and Karsten Schwan, 'On The Core Affinity And File Upload Performance Of Hadoop', DISCS-2013, (2013), 25-30.

[3] Dede, Elif, Madhusudhan Govindaraju, Daniel Gunter, Richard Shane Canon, and Lavanya Ramakrishnan, 'Performance Evaluation Of A Mongodb And Hadoop Platform For Scientific Data Analysis', Sciencecloud '13, (2013), 13-20.

[4] Ding, Mengwei, Long Zheng, Yanchao Lu, Li Li, Song Guo, and Minyi Guo, 'More Convenient More Overhead: The Performance Evaluation Of Hadoop Streaming', RACS '11, (2011), 307-313.

[5] Guo, Shumin. Hadoop Operations And Cluster Management Cookbook, Packt Publishing, 1st ed., 2013, ISBN: 1782165169

[6] Herodotou, Herodotos, 'Hadoop Performance Models', Arxiv Preprint Arxiv:1106.0940, (2011).

[7] Ishii, Masakuni, Jungkyu Han, and Hiroyuki Makino, 'Design And Performance Evaluation For Hadoop Clusters On Virtualized Environment', IEEE Computer Society, (2013), 244-249.

[8] Joshi, Shrinivas B, 'Apache Hadoop Performance-Tuning Methodologies And Best Practices', ICPE '12, (2012), 241-242.

[9] Kim, Shingyu, Junghee Won, Hyuck Han, Hyeonsang Eom, and Heon Y Yeom, 'Improving Hadoop Performance In Intercloud Environments', ACM SIGMETRICS Performance Evaluation Review 39, iss 3 (2011): 107-109.

[10] Li, Jack, Qingyang Wang, Deepal Jayasinghe, Junhee Park, Tao Zhu, and Calton Pu, 'Performance Overhead Among Three Hypervisors: An Experimental Study Using Hadoop Benchmarks', IEEE Computer Society, (2013), 9-16.

[11] Lin, Xuelian, Zide Meng, Chuan Xu, and Meng Wang, 'A Practical Performance Model For Hadoop Mapreduce', IEEE Computer Society, (2012), 231-239.

[12] Liu, Xuhui, Jizhong Han, Yunqin Zhong, Chengde Han, and Xubin He, 'Implementing Webgis On Hadoop: A Case Study Of Improving Small File I/O Performance On HDFS', IEEE Computer Society, (2009), 1-8.

[13] Loewen, Gabriel, Michael Galloway, and Susan Vrbsky, 'On The Performance Of Apache Hadoop In A Tiny Private Iaas Cloud', IEEE Computer Society, (2013), 189-195.

[14] Moh, Neethu as, and Sabu M Thampi, 'Improving Hadoop Performance In Handling Small Files', Springer, (2011), 187-194.

[15] Perera, Srinath, and Thilina Gunarathne, Hadoop Mapreduce Cookbook, Packt Publishing, 1st ed., 2013, ISBN: 1849517282

[16] Shafer, Jeffrey, Scott Rixner, and Alan L Cox, 'The Hadoop Distributed Filesystem: Balancing Portability And Performance', IEEE Computer Society, (2010), 122-133.

[17] Wlodarczyk, Tomasz Wiktor, Yi Han, and Chunming Rong, 'Performance Analysis Of Hadoop For Query Processing', IEEE Computer Society, (2011), 507-513.

[18] Xie, Jiong, Shu Yin, Xiaojun Ruan, Zhiyang Ding, Yun Tian, James Majors, Adam Manzanares, and Xiao Qin, 'Improving Mapreduce Performance Through Data Placement In Heterogeneous Hadoop Clusters', IEEE Computer Society, (2010), 1-9.

[19] http://www.cloudera.com/content/cloudera/en/documentation/cdh5/v5-1-x/CDH5-Requirements-and-Supported-Versions/cdhrsv_jdk.html

[20] http://h10025.www1.hp.com/ewfrf/wc/document?cc=uk&lc=en&docname=c03089646

[21] http://www.cloudera.com/content/cloudera/en/documentation/core/latest/topics/cdh_qs_mrv1_pseudo.html

[22] http://docs.hortonworks.com/HDPDocuments/HDP2/HDP-2.1-latest/bk_using-apache-hadoop/content/running_mapreduce_examples_on_yarn.html

[23] http://docs.hortonworks.com/HDPDocuments/HDP2/HDP-2.0.6.0/bk_installing_manually_book/content/rpm-chap1-11.html

[24] http://ercoppa.github.io/HadoopInternals/AnatomyMapReduceJob.html

[25] https://coda.jlab.org/wiki/index.php/CentOS6_Linux_Diskless_Setup

[26] http://www.cpubenchmark.net/cpu_list.php

[27] http://structureddata.org/2012/06/18/linux-6-transparent-huge-pages-and-hadoop-workloads/

[28] http://alumni.cs.ucr.edu/~jdou/misco/figs/mapreduce.png

[29] http://blogs.vmware.com/cto/analyzing-hadoops-internals-with-analytics

[30] https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/5/html/Tuning_and_Optimizing_Red_Hat_Enterprise_Linux_for_Oracle_9i_and_10g_Databases/chap-Oracle_9i_and_10g_Tuning_Guide-Memory_Usage_and_Page_Cache.html

[31] http://msdn.microsoft.com/en-us/library/bb924370.aspx

## Appendix A – CDH5 packages

```
# find * -type f -iname \*rpm -print
noarch/avro-doc-1.7.5+cdh5.1.0+30-1.cdh5.1.0.p0.31.el6.noarch.rpm
noarch/avro-libs-1.7.5+cdh5.1.0+30-1.cdh5.1.0.p0.31.el6.noarch.rpm
noarch/avro-tools-1.7.5+cdh5.1.0+30-1.cdh5.1.0.p0.31.el6.noarch.rpm
noarch/bigtop-tomcat-0.7.0+cdh5.1.0+0-1.cdh5.1.0.p0.25.el6.noarch.rpm
noarch/bigtop-utils-0.7.0+cdh5.1.0+0-1.cdh5.1.0.p0.30.el6.noarch.rpm
noarch/crunch-0.10.0+cdh5.1.0+14-1.cdh5.1.0.p0.25.el6.noarch.rpm
noarch/crunch-doc-0.10.0+cdh5.1.0+14-1.cdh5.1.0.p0.25.el6.noarch.rpm
noarch/flume-ng-1.5.0+cdh5.1.0+10-1.cdh5.1.0.p0.26.el6.noarch.rpm
noarch/flume-ng-agent-1.5.0+cdh5.1.0+10-1.cdh5.1.0.p0.26.el6.noarch.rpm
noarch/flume-ng-doc-1.5.0+cdh5.1.0+10-1.cdh5.1.0.p0.26.el6.noarch.rpm
noarch/hbase-solr-1.5+cdh5.1.0+12-1.cdh5.1.0.p0.41.el6.noarch.rpm
noarch/hbase-solr-doc-1.5+cdh5.1.0+12-1.cdh5.1.0.p0.41.el6.noarch.rpm
noarch/hbase-solr-indexer-1.5+cdh5.1.0+12-1.cdh5.1.0.p0.41.el6.noarch.rpm
noarch/hive-0.12.0+cdh5.1.0+369-1.cdh5.1.0.p0.39.el6.noarch.rpm
noarch/hive-hbase-0.12.0+cdh5.1.0+369-1.cdh5.1.0.p0.39.el6.noarch.rpm
noarch/hive-hcatalog-0.12.0+cdh5.1.0+369-1.cdh5.1.0.p0.39.el6.noarch.rpm
noarch/hive-jdbc-0.12.0+cdh5.1.0+369-1.cdh5.1.0.p0.39.el6.noarch.rpm
noarch/hive-metastore-0.12.0+cdh5.1.0+369-1.cdh5.1.0.p0.39.el6.noarch.rpm
noarch/hive-server-0.12.0+cdh5.1.0+369-1.cdh5.1.0.p0.39.el6.noarch.rpm
noarch/hive-server2-0.12.0+cdh5.1.0+369-1.cdh5.1.0.p0.39.el6.noarch.rpm
noarch/hive-webhcat-0.12.0+cdh5.1.0+369-1.cdh5.1.0.p0.39.el6.noarch.rpm
noarch/hive-webhcat-server-0.12.0+cdh5.1.0+369-1.cdh5.1.0.p0.39.el6.noarch.rpm
noarch/kite-0.10.0+cdh5.1.0+120-1.cdh5.1.0.p0.30.el6.noarch.rpm
noarch/llama-1.0.0+cdh5.1.0+0-1.cdh5.1.0.p0.25.el6.noarch.rpm
noarch/llama-doc-1.0.0+cdh5.1.0+0-1.cdh5.1.0.p0.25.el6.noarch.rpm
noarch/llama-master-1.0.0+cdh5.1.0+0-1.cdh5.1.0.p0.25.el6.noarch.rpm
noarch/mahout-0.9+cdh5.1.0+11-1.cdh5.1.0.p0.23.el6.noarch.rpm
noarch/mahout-doc-0.9+cdh5.1.0+11-1.cdh5.1.0.p0.23.el6.noarch.rpm
noarch/oozie-4.0.0+cdh5.1.0+249-1.cdh5.1.0.p0.28.el6.noarch.rpm
noarch/oozie-client-4.0.0+cdh5.1.0+249-1.cdh5.1.0.p0.28.el6.noarch.rpm
noarch/parquet-1.2.5+cdh5.1.0+130-1.cdh5.1.0.p0.26.el6.noarch.rpm
noarch/parquet-format-1.0.0+cdh5.1.0+6-1.cdh5.1.0.p0.30.el6.noarch.rpm
noarch/pig-0.12.0+cdh5.1.0+33-1.cdh5.1.0.p0.23.el6.noarch.rpm
noarch/pig-udf-datafu-1.1.0+cdh5.1.0+8-1.cdh5.1.0.p0.21.el6.noarch.rpm
noarch/search-1.0.0+cdh5.1.0+0-1.cdh5.1.0.p0.36.el6.noarch.rpm
noarch/sentry-1.3.0+cdh5.1.0+155-1.cdh5.1.0.p0.59.el6.noarch.rpm
noarch/sentry-store-1.3.0+cdh5.1.0+155-1.cdh5.1.0.p0.59.el6.noarch.rpm
noarch/solr-4.4.0+cdh5.1.0+231-1.cdh5.1.0.p0.32.el6.noarch.rpm
noarch/solr-doc-4.4.0+cdh5.1.0+231-1.cdh5.1.0.p0.32.el6.noarch.rpm
noarch/solr-mapreduce-1.0.0+cdh5.1.0+0-1.cdh5.1.0.p0.36.el6.noarch.rpm
noarch/solr-server-4.4.0+cdh5.1.0+231-1.cdh5.1.0.p0.32.el6.noarch.rpm
noarch/spark-core-1.0.0+cdh5.1.0+41-1.cdh5.1.0.p0.27.el6.noarch.rpm
noarch/spark-history-server-1.0.0+cdh5.1.0+41-1.cdh5.1.0.p0.27.el6.noarch.rpm
noarch/spark-master-1.0.0+cdh5.1.0+41-1.cdh5.1.0.p0.27.el6.noarch.rpm
noarch/spark-python-1.0.0+cdh5.1.0+41-1.cdh5.1.0.p0.27.el6.noarch.rpm
noarch/spark-worker-1.0.0+cdh5.1.0+41-1.cdh5.1.0.p0.27.el6.noarch.rpm
noarch/sqoop-1.4.4+cdh5.1.0+55-1.cdh5.1.0.p0.24.el6.noarch.rpm
noarch/sqoop-metastore-1.4.4+cdh5.1.0+55-1.cdh5.1.0.p0.24.el6.noarch.rpm
noarch/sqoop2-1.99.3+cdh5.1.0+26-1.cdh5.1.0.p0.22.el6.noarch.rpm
noarch/sqoop2-client-1.99.3+cdh5.1.0+26-1.cdh5.1.0.p0.22.el6.noarch.rpm
noarch/sqoop2-server-1.99.3+cdh5.1.0+26-1.cdh5.1.0.p0.22.el6.noarch.rpm
noarch/whirr-0.9.0+cdh5.1.0+9-1.cdh5.1.0.p0.21.el6.noarch.rpm
Oracle_JDK/jdk-7u60-linux-x64.rpm
x86_64/bigtop-jsvc-0.6.0+cdh5.1.0+488-1.cdh5.1.0.p0.25.el6.x86_64.rpm
x86_64/bigtop-jsvc-debuginfo-0.6.0+cdh5.1.0+488-1.cdh5.1.0.p0.25.el6.x86_64.rpm
x86_64/hadoop-0.20-conf-pseudo-2.3.0+cdh5.1.0+795-1.cdh5.1.0.p0.58.el6.x86_64.rpm
x86_64/hadoop-0.20-mapreduce-2.3.0+cdh5.1.0+795-1.cdh5.1.0.p0.58.el6.x86_64.rpm
x86_64/hadoop-0.20-mapreduce-jobtracker-2.3.0+cdh5.1.0+795-1.cdh5.1.0.p0.58.el6.x86_64.rpm
x86_64/hadoop-0.20-mapreduce-jobtrackerha-2.3.0+cdh5.1.0+795-1.cdh5.1.0.p0.58.el6.x86_64.rpm
x86_64/hadoop-0.20-mapreduce-tasktracker-2.3.0+cdh5.1.0+795-1.cdh5.1.0.p0.58.el6.x86_64.rpm
```

```
x86_64/hadoop-0.20-mapreduce-zkfc-2.3.0+cdh5.1.0+795-1.cdh5.1.0.p0.58.el6.x86_64.rpm
x86_64/hadoop-2.3.0+cdh5.1.0+795-1.cdh5.1.0.p0.58.el6.x86_64.rpm
x86_64/hadoop-client-2.3.0+cdh5.1.0+795-1.cdh5.1.0.p0.58.el6.x86_64.rpm
x86_64/hadoop-conf-pseudo-2.3.0+cdh5.1.0+795-1.cdh5.1.0.p0.58.el6.x86_64.rpm
x86_64/hadoop-debuginfo-2.3.0+cdh5.1.0+795-1.cdh5.1.0.p0.58.el6.x86_64.rpm
x86_64/hadoop-doc-2.3.0+cdh5.1.0+795-1.cdh5.1.0.p0.58.el6.x86_64.rpm
x86_64/hadoop-hdfs-2.3.0+cdh5.1.0+795-1.cdh5.1.0.p0.58.el6.x86_64.rpm
x86_64/hadoop-hdfs-datanode-2.3.0+cdh5.1.0+795-1.cdh5.1.0.p0.58.el6.x86_64.rpm
x86_64/hadoop-hdfs-fuse-2.3.0+cdh5.1.0+795-1.cdh5.1.0.p0.58.el6.x86_64.rpm
x86_64/hadoop-hdfs-journalnode-2.3.0+cdh5.1.0+795-1.cdh5.1.0.p0.58.el6.x86_64.rpm
x86_64/hadoop-hdfs-namenode-2.3.0+cdh5.1.0+795-1.cdh5.1.0.p0.58.el6.x86_64.rpm
x86_64/hadoop-hdfs-nfs3-2.3.0+cdh5.1.0+795-1.cdh5.1.0.p0.58.el6.x86_64.rpm
x86_64/hadoop-hdfs-secondarynamenode-2.3.0+cdh5.1.0+795-1.cdh5.1.0.p0.58.el6.x86_64.rpm
x86_64/hadoop-hdfs-zkfc-2.3.0+cdh5.1.0+795-1.cdh5.1.0.p0.58.el6.x86_64.rpm
x86_64/hadoop-httpfs-2.3.0+cdh5.1.0+795-1.cdh5.1.0.p0.58.el6.x86_64.rpm
x86_64/hadoop-libhdfs-2.3.0+cdh5.1.0+795-1.cdh5.1.0.p0.58.el6.x86_64.rpm
x86_64/hadoop-libhdfs-devel-2.3.0+cdh5.1.0+795-1.cdh5.1.0.p0.58.el6.x86_64.rpm
x86_64/hadoop-mapreduce-2.3.0+cdh5.1.0+795-1.cdh5.1.0.p0.58.el6.x86_64.rpm
x86_64/hadoop-mapreduce-historyserver-2.3.0+cdh5.1.0+795-1.cdh5.1.0.p0.58.el6.x86_64.rpm
x86_64/hadoop-yarn-2.3.0+cdh5.1.0+795-1.cdh5.1.0.p0.58.el6.x86_64.rpm
x86_64/hadoop-yarn-nodemanager-2.3.0+cdh5.1.0+795-1.cdh5.1.0.p0.58.el6.x86_64.rpm
x86_64/hadoop-yarn-proxyserver-2.3.0+cdh5.1.0+795-1.cdh5.1.0.p0.58.el6.x86_64.rpm
x86_64/hadoop-yarn-resourcemanager-2.3.0+cdh5.1.0+795-1.cdh5.1.0.p0.58.el6.x86_64.rpm
x86_64/hbase-0.98.1+cdh5.1.0+64-1.cdh5.1.0.p0.34.el6.x86_64.rpm
x86_64/hbase-doc-0.98.1+cdh5.1.0+64-1.cdh5.1.0.p0.34.el6.x86_64.rpm
x86_64/hbase-master-0.98.1+cdh5.1.0+64-1.cdh5.1.0.p0.34.el6.x86_64.rpm
x86_64/hbase-regionserver-0.98.1+cdh5.1.0+64-1.cdh5.1.0.p0.34.el6.x86_64.rpm
x86_64/hbase-rest-0.98.1+cdh5.1.0+64-1.cdh5.1.0.p0.34.el6.x86_64.rpm
x86_64/hbase-thrift-0.98.1+cdh5.1.0+64-1.cdh5.1.0.p0.34.el6.x86_64.rpm
x86_64/hue-3.6.0+cdh5.1.0+86-1.cdh5.1.0.p0.36.el6.x86_64.rpm
x86_64/hue-beeswax-3.6.0+cdh5.1.0+86-1.cdh5.1.0.p0.36.el6.x86_64.rpm
x86_64/hue-common-3.6.0+cdh5.1.0+86-1.cdh5.1.0.p0.36.el6.x86_64.rpm
x86_64/hue-doc-3.6.0+cdh5.1.0+86-1.cdh5.1.0.p0.36.el6.x86_64.rpm
x86_64/hue-hbase-3.6.0+cdh5.1.0+86-1.cdh5.1.0.p0.36.el6.x86_64.rpm
x86_64/hue-impala-3.6.0+cdh5.1.0+86-1.cdh5.1.0.p0.36.el6.x86_64.rpm
x86_64/hue-pig-3.6.0+cdh5.1.0+86-1.cdh5.1.0.p0.36.el6.x86_64.rpm
x86_64/hue-plugins-3.6.0+cdh5.1.0+86-1.cdh5.1.0.p0.36.el6.x86_64.rpm
x86_64/hue-rdbms-3.6.0+cdh5.1.0+86-1.cdh5.1.0.p0.36.el6.x86_64.rpm
x86_64/hue-search-3.6.0+cdh5.1.0+86-1.cdh5.1.0.p0.36.el6.x86_64.rpm
x86_64/hue-server-3.6.0+cdh5.1.0+86-1.cdh5.1.0.p0.36.el6.x86_64.rpm
x86_64/hue-spark-3.6.0+cdh5.1.0+86-1.cdh5.1.0.p0.36.el6.x86_64.rpm
x86_64/hue-sqoop-3.6.0+cdh5.1.0+86-1.cdh5.1.0.p0.36.el6.x86_64.rpm
x86_64/hue-zookeeper-3.6.0+cdh5.1.0+86-1.cdh5.1.0.p0.36.el6.x86_64.rpm
x86_64/impala-1.4.0+cdh5.1.0+0-1.cdh5.1.0.p0.92.el6.x86_64.rpm
x86_64/impala-catalog-1.4.0+cdh5.1.0+0-1.cdh5.1.0.p0.92.el6.x86_64.rpm
x86_64/impala-debuginfo-1.4.0+cdh5.1.0+0-1.cdh5.1.0.p0.92.el6.x86_64.rpm
x86_64/impala-server-1.4.0+cdh5.1.0+0-1.cdh5.1.0.p0.92.el6.x86_64.rpm
x86_64/impala-shell-1.4.0+cdh5.1.0+0-1.cdh5.1.0.p0.92.el6.x86_64.rpm
x86_64/impala-state-store-1.4.0+cdh5.1.0+0-1.cdh5.1.0.p0.92.el6.x86_64.rpm
x86_64/impala-udf-devel-1.4.0+cdh5.1.0+0-1.cdh5.1.0.p0.92.el6.x86_64.rpm
x86_64/zookeeper-3.4.5+cdh5.1.0+29-1.cdh5.1.0.p0.31.el6.x86_64.rpm
x86_64/zookeeper-debuginfo-3.4.5+cdh5.1.0+29-1.cdh5.1.0.p0.31.el6.x86_64.rpm
x86_64/zookeeper-native-3.4.5+cdh5.1.0+29-1.cdh5.1.0.p0.31.el6.x86_64.rpm
x86_64/zookeeper-server-3.4.5+cdh5.1.0+29-1.cdh5.1.0.p0.31.el6.x86_64.rpm
#
```

## Appendix B – Kickstart file

```
install
url --url http://repo.irl/centos/7/os/x86_64/
lang en_US.UTF-8
keyboard --vckeymap=us --xlayouts='us'
timezone Europe/Dublin --isUtc
network --noipv6 --onboot=yes --bootproto dhcp --device=eth0 --hostname=goldenimage
network --noipv6 --onboot=no --device=eth1
network --noipv6 --onboot=no --device=eth2
network --noipv6 --onboot=no --device=eth3
network --noipv6 --onboot=no --device=eth4
network --noipv6 --onboot=no --device=eth5
auth --enableshadow --enablemd5
rootpw --iscrypted putpasswordhashhere_censored.
firewall --disabled
firstboot --disable
selinux --disabled
services --disabled="kdump,iprdump,iprinit,iprupdate,auditd,irqbalance,postfix,tuned,systemd-
readahead-collect,systemd-readahead-drop,systemd-readahead-replay,hadoop-0.20-mapreduce-
jobtracker,hadoop-0.20-mapreduce-tasktracker,hadoop-hdfs-datanode,hadoop-hdfs-namenode, rhel-
autorelabel-mark, systemd-random-seed" --enabled="sshd,chronyd"
skipx
text
eula --agreed
poweroff
%include /tmp/include.me
#do not reboot to prevent second install
#reboot

%packages --nobase --ignoremissing
@core
yum
openssh-server
chrony
kernel-firmware
wget
sudo
perl
psmisc
bind-utils
nfs-utils
cachefilesd
nfstest
nfsometer
sysstat
screen
netstat
zip
unzip
bzip2
bzip2-1.0.6-12.el7.x86_64
gzip
lsof
lm_sensors
net-tools
gcc
gcc-c++
autoconf
automake
binutils
bison
flex
```

```
gettext
libtool
make
patch
pkgconfig
redhat-rpm-config
rpm-build
rpm-sign
patchutils
git
subversion
ncurses-devel
hmaccalc
zlib-devel
binutils-devel
elfutils-libelf-devel
-*firmware
-b43-openfwwf
-efibootmgr
-fcoe*
-iscsi*
%end

%pre --log=/root/install-pre.log
#!/bin/bash

incfile=/tmp/include.me
> $incfile

# Check physical disks
MAXSIZE=16384  # in Megabytes
for disk in /sys/block/sd*
do
        dsk=$(basename $disk)

        if [[ `cat $disk/ro` -eq 1 ]];
        then
                echo "Skipping disk $dsk: READONLY"
                continue;
        fi

        if [[ $((`cat $disk/size`*512/1024/1024)) -gt ${MAXSIZE} ]];
        then
                echo "Skipping disk $dsk: larger than ${MAXSIZE} megabytes"
                continue;
        else
                echo "Using disk $dsk"
                chosen=$dsk;
                break;
        fi
done

if [[ -n "$chosen" ]];
then
        echo "zerombr" >> $incfile
        echo "bootloader --location=mbr --timeout=3 --boot-drive=$chosen --driveorder=$chosen
--append=\"nomodeset\"" >> $incfile
        echo "ignoredisk --only-use=$chosen" >> $incfile
        echo "clearpart --all --initlabel --drives=$chosen" >> $incfile
        echo "part / --fstype=ext4 --asprimary --ondisk=$chosen --size=6144 --grow" >>
$incfile
else
        echo "" > $incfile
fi
```

```
%end

%post --log=/root/install-post.log
#!/bin/bash

echo "Configuring host keys"
cat <<EOF > /etc/ssh/ssh_host_dsa_key
-----BEGIN DSA PRIVATE KEY-----
somekeyhere_censored
-----END DSA PRIVATE KEY-----
EOF
chmod 600 /etc/ssh/ssh_host_dsa_key
cat <<EOF > /etc/ssh/ssh_host_dsa_key.pub
ssh-dss somekeyhere_censored
EOF
chmod 644 /etc/ssh/ssh_host_dsa_key.pub
cat <<EOF > /etc/ssh/ssh_host_ecdsa_key
-----BEGIN EC PRIVATE KEY-----
somekeyhere_censored
-----END EC PRIVATE KEY-----
EOF
chmod 600 /etc/ssh/ssh_host_ecdsa_key
cat <<EOF > /etc/ssh/ssh_host_ecdsa_key.pub
ecdsa-sha2-nistp256 somekeyhere_censored
EOF
chmod 644 /etc/ssh/ssh_host_ecdsa_key.pub
cat <<EOF > /etc/ssh/ssh_host_rsa_key
-----BEGIN RSA PRIVATE KEY-----
Somekeyhere_censored
-----END RSA PRIVATE KEY-----
EOF
chmod 600 /etc/ssh/ssh_host_rsa_key
cat <<EOF > /etc/ssh/ssh_host_rsa_key.pub
ssh-rsa somekeyhere_censored
EOF
chmod 644 /etc/ssh/ssh_host_rsa_key.pub

echo "Setting up key-based SSH for the root user"
mkdir /root/.ssh
echo 'ssh-rsa somekeyhere_censored' > /root/.ssh/authorized_keys
chmod 700 /root/.ssh
chmod 644 /root/.ssh/authorized_keys

echo "SSH config setup"
cat <<EOF > /root/.ssh/config
GSSAPIAuthentication no
ForwardAgent yes
StrictHostKeyChecking no
EOF
chmod 644 /root/.ssh/config

echo "Creating bash profile"
/usr/bin/perl -npe '/alias/ && s/^/#/' -i /root/.bashrc
cat <<EOF >> /root/.bashrc
set -o vi
alias grep='grep --colour=auto'
alias less='less -i'
alias ll='ls -la --color'
alias ls='ls --color=auto'
alias shutdown='echo SHUTDOWN!?!?'
alias syslog='tail -n 55 -f /var/log/messages'
alias dfs='/usr/bin/df -h | /usr/bin/grep -v ^none'
alias jps='/usr/java/default/bin/jps'
export HISTFILE=/hdfs/scratch/.bash_history_root
```

```
EOF

echo "Setting up timekeeping with chrony"
/usr/bin/perl -npe '/^server/ && s/^/#/' -i /etc/chrony.conf
echo "server time.irl iburst" >> /etc/chrony.conf

echo "Marking root read-only"
/bin/sed -i 's/^READONLY.*/READONLY=yes/' /etc/sysconfig/readonly-root
/bin/sed -i 's/^TEMPORARY_STATE.*/TEMPORARY_STATE=yes/' /etc/sysconfig/readonly-root
echo "" >> /etc/rwtab
echo "dirs  /var/cache/yum" >> /etc/rwtab
echo "dirs  /var/lib/systemd" >> /etc/rwtab
echo "dirs  /etc/hadoop/conf.irl" >> /etc/rwtab
echo "dirs  /kernel" >> /etc/rwtab
echo "files /etc/hostname" >> /etc/rwtab
echo "files /etc/chrony.keys" >> /etc/rwtab
echo "files /etc/sysconfig/network" >> /etc/rwtab
echo "files /etc/rwtab" >> /etc/rwtab
echo "files /etc/sysconfig/readonly-root" >> /etc/rwtab
echo "files /var/lib/chrony/drift.tmp" >> /etc/rwtab

echo "Linking kernel modules"
ln -s /lib/modules/3.10.0-123.el7.x86_64 /lib/modules/3.10.0-123.4.4.el7.x86_64
mkdir /kernel

echo "Disabling IPv6"
echo "net.ipv6.conf.all.disable_ipv6 = 1" >> /etc/sysctl.conf
echo "net.ipv6.conf.default.disable_ipv6 = 1" >> /etc/sysctl.conf
/bin/sed -i 's/^#\(Listen.*0.0.0.0\)$/\1/' /etc/ssh/sshd_config
/bin/sed -i '/^::.*/d' /etc/hosts
/bin/sed -i -e 's/^tcp6/#tcp6/' -e 's/^udp6/#udp6/' /etc/netconfig

echo "Setting up rc.local"
echo "goldenimage" > /etc/hostname
cat <<EOF >> /etc/rc.d/rc.local
MYIPADDR=\$(ip addr show eno1 | sed -n 's/\sinet \([0-9]*\.[0-9]*\.[0-9]*\.[0-9]*\)\/[0-9]*
brd.*/\1/p')
MYHOSTNAME=\$(/usr/bin/dig -x \${MYIPADDR} +short | sed 's/\.$//')
/usr/bin/hostnamectl set-hostname \$MYHOSTNAME
echo \$MYHOSTNAME > /etc/hostname
echo HOSTNAME=\$MYHOSTNAME >> /etc/sysconfig/network

# to get correct hostname in syslog
systemctl restart rsyslog.service

# Loading Hadoop configuration
/root/reload_config.sh
EOF
chmod +x /etc/rc.d/rc.local

echo "Creating reload_config.sh"
cat <<EOF > /root/reload_config.sh
#!/bin/bash
# Populating Hadoop config from central repo

/usr/bin/rm /etc/hadoop/conf/* 2>/dev/null
/usr/bin/mount -o ro raspi:/btrfs/hadoop_conf /mnt
/usr/bin/cp -p /mnt/* /etc/hadoop/conf/
/usr/bin/umount /mnt

printf "%-20s%s\n" \$(hostname) "Hadoop configuration [RELOADED]"
EOF
chmod +x /root/reload_config.sh
```

```
echo "Creating get_sources.sh"
cat <<EOF > /root/get_sources.sh
#!/bin/bash

/usr/bin/mount -o ro raspi:/btrfs/kernel /mnt
/usr/bin/xz --decompress --keep --stdout /mnt/linux-3.17-rc4.tar.xz | /usr/bin/tar -xf - -C
/kernel/
/usr/bin/umount /mnt

EOF
chmod +x /root/get_sources.sh

echo "Setting default JAVA_HOME"
echo 'export JAVA_HOME=/usr/java/default' >> /etc/default/bigtop-utils

echo "Adding Cloudera repository"
/usr/bin/curl http://repo.irl/cdh5/cdh5.repo > /etc/yum.repos.d/cdh5.repo
echo "Installing Hadoop"
/usr/bin/yum -y install hadoop-0.20-mapreduce-jobtracker hadoop-hdfs-namenode hadoop-hdfs-fuse
hadoop-0.20-mapreduce-tasktracker hadoop-hdfs-datanode hadoop-mapreduce hadoop-yarn hadoop-
yarn-nodemanager hadoop-yarn-resourcemanager hadoop-mapreduce-historyserver hadoop-hdfs-nfs3
jdk hue oozie hive

echo "Updating YUM Repositories"
# disabling mirrorlist for os and updates
/usr/bin/perl -npe '/mirrorlist=.*repo=os/ && s/^/#/' -i /etc/yum.repos.d/CentOS-Base.repo
/usr/bin/perl -npe '/mirrorlist=.*repo=updates/ && s/^/#/' -i /etc/yum.repos.d/CentOS-
Base.repo
# enabling baseurl pointing to a single http server
/usr/bin/perl -npe '/^#baseurl=.*\/os\// && s/^#//' -i /etc/yum.repos.d/CentOS-Base.repo
/usr/bin/perl -npe '/^#baseurl=.*\/updates\// && s/^#//' -i /etc/yum.repos.d/CentOS-Base.repo
# setting my own repo
/usr/bin/perl -npe '/^baseurl=.*\/os\// && s/mirror.centos.org/repo.irl/' -i
/etc/yum.repos.d/CentOS-Base.repo
/usr/bin/perl -npe '/^baseurl=.*\/updates\// && s/mirror.centos.org/repo.irl/' -i
/etc/yum.repos.d/CentOS-Base.repo
# turning off keycheck
/usr/bin/perl -npe 's/^gpgcheck=1/gpgcheck=0/' -i /etc/yum.repos.d/CentOS-Base.repo
# and now update
/usr/bin/yum -y update
/usr/bin/yum clean all

echo "Disabling some legacy services and unneeded cron jobs"
# Hadoop roles handled by custom initscripts
/sbin/chkconfig hadoop-0.20-mapreduce-jobtracker off
/sbin/chkconfig hadoop-0.20-mapreduce-tasktracker off
/sbin/chkconfig hadoop-hdfs-datanode off
/sbin/chkconfig hadoop-hdfs-namenode off
/sbin/chkconfig hadoop-hdfs-nfs3 off
/sbin/chkconfig hadoop-mapreduce-historyserver off
/sbin/chkconfig hadoop-yarn-nodemanager off
/sbin/chkconfig hadoop-yarn-resourcemanager off
/sbin/chkconfig hue off
/sbin/chkconfig oozie off
/sbin/chkconfig iprupdate off
# we don't want to store rnd seed on a stateless server
/usr/bin/systemctl disable systemd-random-seed.service
rm /etc/cron.daily/*
rm /etc/cron.hourly/*

echo "Preparing Hadoop configuration"
mkdir /etc/hadoop/conf.irl
rm /etc/alternatives/hadoop-conf
ln -s /etc/hadoop/conf.irl /etc/alternatives/hadoop-conf
```

```
echo "/etc/hadoop/conf.irl" >> /var/lib/alternatives/hadoop-conf
echo "15" >> /var/lib/alternatives/hadoop-conf

echo "Preparing Oozie configuration"
rm -r /var/lib/oozie
ln -s /hdfs/oozie /var/lib/oozie
/usr/bin/curl 'http://archive.cloudera.com/gplextras/misc/ext-2.2.zip' > /tmp/ext-2.2.zip
unzip /tmp/ext-2.2.zip -d /usr/lib/
rm /tmp/ext-2.2.zip

echo "Installing python 2.6 from EPEL - hue prerequisite"
/usr/bin/rpm --quiet -i http://download.fedoraproject.org/pub/epel/5/i386/epel-release-5-
4.noarch.rpm
/usr/bin/yum install python26

echo "Preparing Hue configuration"
cp -r /etc/hue/conf.empty /etc/hue/conf.irl
rm /etc/alternatives/hue-conf
ln -s /etc/hue/conf.irl /etc/alternatives/hue-conf
echo "/etc/hue/conf.irl" >> /var/lib/alternatives/hue-conf
echo "40" >> /var/lib/alternatives/hue-conf
sed -i 's/\(\s\)## \(webhdfs_url=http...\).*\(.webhdfs.*\)/\1\2namenode.irl:50070\3/'
/etc/hue/conf.irl/hue.ini

echo "Setting up HDFS backing store"
echo 'LABEL=HADOOP   /hdfs ext4  defaults 0 2' > /etc/fstab
mkdir /hdfs

echo "Setting profile for user yarn"
echo "set -o vi" >> /var/lib/hadoop-yarn/.bash_profile
echo "alias jps=/usr/java/default/bin/jps" >> /var/lib/hadoop-yarn/.bash_profile
echo "export PS1=\"yarn % \"" >> /var/lib/hadoop-yarn/.bash_profile
echo "export HISTFILE=/hdfs/scratch/.bash_history_yarn" >> /var/lib/hadoop-yarn/.bash_profile
/usr/bin/chown 994:995 /var/lib/hadoop-yarn/.bash_profile
/usr/bin/mkdir -p /var/lib/hadoop-yarn/cache/yarn/nm-local-dir
/usr/bin/chown -R 995:995 /var/lib/hadoop-yarn/cache/yarn

echo "Setting profile for user hdfs"
echo "set -o vi" >> /var/lib/hadoop-hdfs/.bash_profile
echo "alias jps=/usr/java/default/bin/jps" >> /var/lib/hadoop-hdfs/.bash_profile
echo "export PS1=\"hdfs % \"" >> /var/lib/hadoop-hdfs/.bash_profile
echo "export HISTFILE=/hdfs/scratch/.bash_history_hdfs" >> /var/lib/hadoop-hdfs/.bash_profile
/usr/bin/chown 996:995 /var/lib/hadoop-hdfs/.bash_profile

echo "Setting profile for user mapred"
echo "set -o vi" >> /var/lib/hadoop-mapreduce/.bash_profile
echo "alias jps=/usr/java/default/bin/jps" >> /var/lib/hadoop-mapreduce/.bash_profile
echo "export PS1=\"mapred % \"" >> /var/lib/hadoop-mapreduce/.bash_profile
echo "export HISTFILE=/hdfs/scratch/.bash_history_mapred" >> /var/lib/hadoop-
mapreduce/.bash_profile
/usr/bin/chown 995:995 /var/lib/hadoop-mapreduce/.bash_profile

echo "Deploying latest sysstat"
mount -o ro raspi:/btrfs/sysstat /mnt
cd /mnt
find . | cpio -pdm /
cd /
umount /mnt

%end
```

## Appendix C – Switch configuration

```
sw01#show running-config full
Building configuration...

Current configuration : 3362 bytes
!
version 12.2
no service pad
service timestamps debug uptime
service timestamps log uptime
service password-encryption
!
hostname sw01
!
enable secret 5 $1$F3tP$oVyt1grcnUIIqKrrcLsmw0
enable password 7 130D071B051A01243F
!
username root privilege 15 password 7 04531B0F013749401D
aaa new-model
!
aaa session-id common
ip subnet-zero
!
ip domain-name sw01.irl
!
!
crypto pki trustpoint TP-self-signed-433077504
 enrollment selfsigned
 subject-name cn=IOS-Self-Signed-Certificate-433077504
 revocation-check none
 rsakeypair TP-self-signed-433077504
!
!
crypto ca certificate chain TP-self-signed-433077504
 certificate self-signed 01
  3082027F 308201E8 A0030201 02020101 300D0609 2A864886 F70D0101 04050030
  4E312E30 2C060355 04031325 494F532D 53656C66 2D536967 6E65642D 43657274
  69666963 6174652D 34333330 37373530 34311C30 1A06092A 864886F7 0D010902
  160D7377 30312E73 7730312E 69726C30 1E170D39 33303330 31303030 3133355A
  170D3230 30313031 30303030 30305A30 4E312E30 2C060355 04031325 494F532D
  53656C66 2D536967 6E65642D 43657274 69666963 6174652D 34333330 37373530
  34311C30 1A06092A 864886F7 0D010902 160D7377 30312E73 7730312E 69726C30
  819F300D 06092A86 4886F70D 01010105 0003818D 00308189 02818100 E0967ED3
  66A341D4 8343CC10 E32CE2B5 5B8942B0 F832E836 42E4D9C7 A3582F7D CE1DCF2C
  87ED2ABD A9239EE6 957DC82F CBE9A720 F25B496C 31875E93 AF234FC4 09B73325
  8A4EA81A A0364460 0010FF33 CED03A85 78D3E6DA 201B36F4 216B3C9F D5CF831C
  2BFAD056 F6601463 6B986381 B8AD3BC5 D7894B22 6815E22B 2A1F5DEF 02030100
  01A36D30 6B300F06 03551D13 0101FF04 05300301 01FF3018 0603551D 11041130
  0F820D73 7730312E 73773031 2E69726C 301F0603 551D2304 18301680 14731895
  F55D29AE 701060EB 2067B380 E9EC79DD 68301D06 03551D0E 04160414 731895F5
  5D29AE70 1060EB20 67B380E9 EC79DD68 300D0609 2A864886 F70D0101 04050003
  81810032 E21717C1 4F8049BD 6E37B61A B18CBC6A 05701754 17B1F54F 111A53F2
  72F15AC2 EE85BC80 F61ADDE8 29F7C2F0 84B4EFD0 CD437859 ECC84709 4AF1F99C
  87939E5E 846432FD FC2A9102 B5924261 26ACE072 E873EEB4 3ADEB4C1 AF415CEC
  4DC61CC2 898B71DC A578EF82 CCFDE334 E7603F5F 1785E47D C79902CF 5A166C85
  5DABCB
  quit
!
!
no file verify auto
spanning-tree mode pvst
spanning-tree extend system-id
```

```
!
vlan internal allocation policy ascending
!
interface GigabitEthernet0/1
!
interface GigabitEthernet0/2
!
interface GigabitEthernet0/3
!
interface GigabitEthernet0/4
!
interface GigabitEthernet0/5
!
interface GigabitEthernet0/6
!
interface GigabitEthernet0/7
!
interface GigabitEthernet0/8
!
interface GigabitEthernet0/9
!
interface GigabitEthernet0/10
!
interface GigabitEthernet0/11
!
interface GigabitEthernet0/12
!
interface GigabitEthernet0/13
!
interface GigabitEthernet0/14
!
interface GigabitEthernet0/15
!
interface GigabitEthernet0/16
!
interface GigabitEthernet0/17
!
interface GigabitEthernet0/18
!
interface GigabitEthernet0/19
!
interface GigabitEthernet0/20
!
interface GigabitEthernet0/21
!
interface GigabitEthernet0/22
!
interface GigabitEthernet0/23
!
interface GigabitEthernet0/24
!
interface Vlan1
 ip address 192.168.1.38 255.255.255.0
 no ip route-cache
!
ip default-gateway 192.168.1.254
ip http server
ip http secure-server
radius-server source-ports 1645-1646
!
control-plane
!
!
line con 0
```

```
 exec-timeout 0 0
line vty 0 4
 password 7 04531B0F013749401D
line vty 5 15
 password 7 04531B0F013749401D
!
end

sw01#
```

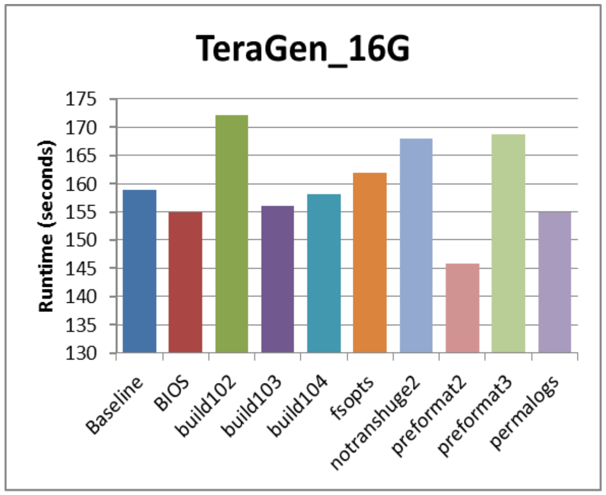
# Appendix D – TeraTest results with 16 Gbyte dataset

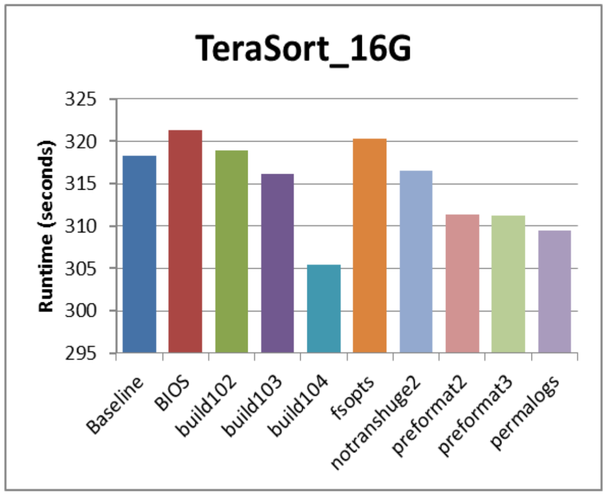
| | All | | Non-failed | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **TeraGen_16G** | **AVERAGE** | **rel.stddev** | **AVERAGE** | **rel.stddev** | **Improvement** | | | | | | | | | | | |
| Baseline | **158.831** | 16.636% | **158.831** | 16.636% | 0.000% | 136.335 | 181.427 | 147.310 | 146.497 | 160.049 | 133.819 | 206.378 | | | | |
| BIOS | **154.832** | 12.823% | **154.832** | 12.823% | 2.518% | 152.403 | 186.713 | 141.368 | 157.911 | 135.763 | | | | | | |
| build102 | **172.175** | 9.585% | **172.175** | 9.585% | -8.401% | 155.079 | 163.357 | 199.406 | 169.031 | 170.049 | 158.335 | 189.967 | | | | |
| build103 | **156.118** | 6.657% | **156.118** | 6.657% | 1.708% | 146.952 | 155.318 | 156.509 | 173.182 | 148.628 | | | | | | |
| build104 | **158.147** | 6.474% | **158.147** | 6.474% | 0.431% | 168.480 | 147.941 | 141.987 | 152.755 | 145.332 | 159.594 | 139.866 | 209.22 | | | |
| fsopts | **161.912** | 7.892% | **161.912** | 7.892% | -1.940% | 161.674 | 144.559 | 180.472 | 159.566 | 163.289 | | | | | | |
| notranshuge2 | **167.928** | 6.632% | **167.928** | 6.632% | -5.728% | 187.030 | 163.357 | 164.215 | 166.884 | 158.154 | | | | | | |
| preformat2 | **145.796** | 3.523% | **145.796** | 3.523% | 8.207% | 154.606 | 144.498 | 143.793 | 151.310 | 139.539 | 143.784 | 148.009 | 145.989 | 145.401 | 141.029 | |
| preformat3 | **168.731** | 11.278% | **168.731** | 11.278% | -6.233% | 167.074 | 178.267 | 192.211 | 202.162 | 162.546 | 147.215 | 162.222 | 165.907 | 155.877 | 153.832 | |
| permalogs | **154.939** | 4.810% | **154.939** | 4.810% | 2.450% | 152.456 | 144.475 | 145.837 | 148.824 | 153.078 | 160.431 | 164.681 | 151.671 | 165.765 | 157.108 | 160.001 |
| **TeraSort_16G** | **AVERAGE** | **rel.stddev** | **AVERAGE** | **rel.stddev** | **Improvement** | | | | | | | | | | | |
| Baseline | **318.308** | 3.460% | **318.308** | 3.460% | 0.000% | 307.999 | 323.126 | 321.954 | 313.243 | 338.543 | 306.114 | 317.179 | | | | |
| BIOS | **321.362** | 4.245% | **321.362** | 4.245% | -0.959% | 298.535 | 326.724 | 322.658 | 335.024 | 323.871 | | | | | | |
| build102 | **318.952** | 4.391% | **318.952** | 4.391% | -0.202% | 308.663 | 308.748 | 342.871 | 317.107 | 331.736 | 303.864 | 319.678 | | | | |
| build103 | **316.104** | 3.170% | **316.104** | 3.170% | 0.693% | 309.823 | 329.860 | 303.943 | 321.090 | 315.802 | | | | | | |
| build104 | **305.423** | 2.669% | **305.423** | 2.669% | 4.048% | 291.721 | 301.951 | 304.852 | 317.939 | 307.965 | 310.752 | 302.781 | 309.656 | | | |
| fsopts | **317.945** | 3.547% | **320.257** | 3.614% | -0.612% | 309.704 | **308.695** | 312.527 | 335.039 | 323.758 | | | | | | |
| notranshuge2 | **316.579** | 4.535% | **316.579** | 4.535% | 0.543% | 339.105 | 304.740 | 309.793 | 322.478 | 306.779 | | | | | | |
| preformat2 | **311.791** | 1.760% | **311.343** | 1.478% | 2.188% | 306.685 | 308.505 | 304.871 | 318.696 | 318.653 | 312.479 | 312.648 | 308.516 | 311.664 | 310.71 | |
| preformat3 | **309.342** | 2.495% | **311.214** | 2.364% | 2.229% | 302.664 | 316.694 | 321.803 | 309.872 | 305.724 | 299.873 | 308.764 | 309.974 | 319.031 | 317.745 | |
| permalogs | **309.692** | 1.048% | **309.477** | 1.502% | 2.774% | 305.741 | 306.454 | 310.658 | 312.745 | 314.706 | 308.714 | 308.827 | 300.46 | 312.621 | 316.749 | 306.577 |
| **TeraValidate_16G** | **AVERAGE** | **rel.stddev** | **AVERAGE** | **rel.stddev** | **Improvement** | | | | | | | | | | | |
| Baseline | **34.673** | 6.855% | **34.673** | 6.855% | 0.000% | 34.530 | 33.650 | 33.700 | 32.611 | 39.798 | 33.525 | 34.899 | | | | |
| BIOS | **35.337** | 8.166% | **35.337** | 8.166% | -1.913% | 33.517 | 35.561 | 38.630 | 37.459 | 31.516 | | | | | | |
| build102 | **33.988** | 7.862% | **33.988** | 7.862% | 1.977% | 37.520 | 31.339 | 33.414 | 34.514 | 32.314 | 31.277 | 37.537 | | | | |
| build103 | **31.153** | 5.357% | **31.153** | 5.357% | 10.154% | 32.304 | 29.338 | 33.435 | 30.357 | 30.329 | | | | | | |
| build104 | **33.705** | 11.048% | **33.705** | 11.048% | 2.793% | 29.448 | 33.515 | 38.595 | 29.333 | 38.367 | 33.318 | 33.358 | 31.325 | | | |
| fsopts | **33.061** | 2.649% | **33.061** | 2.649% | 4.649% | 32.196 | 32.176 | 34.230 | 33.355 | 33.350 | | | | | | |
| notranshuge2 | **33.962** | 6.525% | **33.962** | 6.525% | 2.052% | 33.367 | 37.343 | 34.425 | 31.250 | 33.424 | | | | | | |
| preformat2 | **31.364** | 3.479% | **31.364** | 3.479% | 9.545% | 30.201 | 31.177 | 31.250 | 32.256 | 33.253 | 31.208 | 30.200 | 32.257 | 31.183 | 33.313 | |
| preformat3 | **35.511** | 7.318% | **35.511** | 7.318% | -2.415% | 38.486 | 34.299 | 38.562 | 32.220 | 37.400 | 34.377 | 33.230 | 31.147 | 32.304 | 39.545 | |
| permalogs | **32.270** | 11.032% | **32.270** | 11.032% | 6.932% | 33.143 | 39.297 | 29.072 | 29.128 | 30.059 | 33.185 | 32.004 | 38.402 | 29.149 | 34.141 | 37.292 |

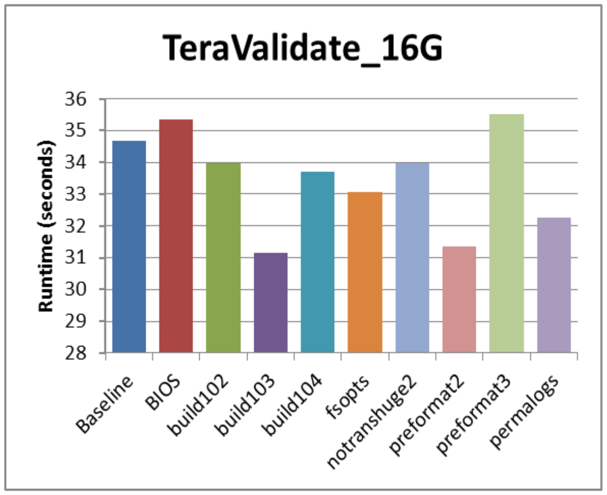
## Appendix E – TeraTest results with 1 Tbyte dataset

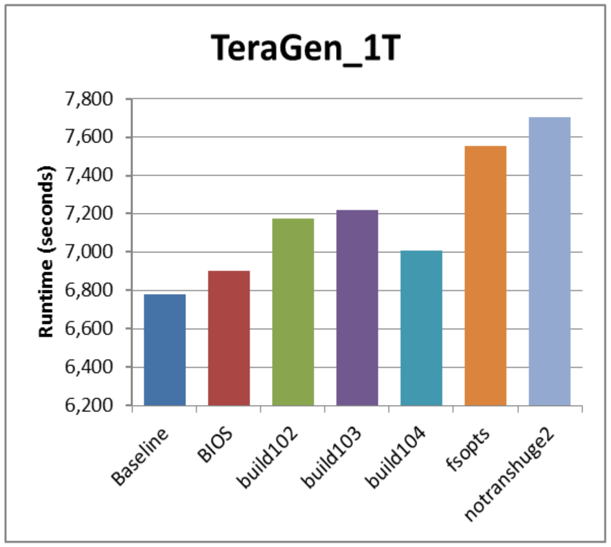
| | All | | Non-failed | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **TeraGen_1T** | **AVERAGE** | **rel.stddev** | **AVERAGE** | **rel.stddev** | **improvement** | | | | | | | | |
| Baseline | **6,779.301** | 13.563% | **6779.301** | 13.563% | 0.000% | 5842.454 | 7900.383 | 5872.387 | 7901.335 | 6211.282 | 6353.998 | 7373.266 | |
| BIOS | **6,901.602** | 10.494% | **6901.602** | 10.494% | -1.804% | 6863.466 | 7865.186 | 6307.869 | 7352.444 | 6119.047 | | | |
| build102 | **7,172.662** | 11.922% | **7172.662** | 11.922% | -5.802% | 7542.650 | 6092.350 | 8163.470 | 6445.050 | 8119.260 | 6397.774 | 7448.079 | |
| build103 | **7,218.132** | 7.416% | **7218.132** | 7.416% | -6.473% | 7495.917 | 7137.524 | 6794.042 | 7986.129 | 6677.046 | | | |
| build104 | **7,005.503** | 10.160% | **7005.503** | 10.160% | -3.337% | 8027.518 | 6949.623 | 6470.400 | 6888.894 | 6499.039 | 7610.749 | 5942.822 | 7654.982 |
| fsopts | **7,555.806** | 8.044% | **7555.806** | 8.044% | -11.454% | 7762.791 | 6735.948 | 8227.453 | 7132.798 | 7920.038 | | | |
| notranshuge2 | **7,704.344** | 4.657% | **7704.344** | 4.657% | -13.645% | 8002.544 | 7374.656 | 7951.191 | 7255.285 | 7938.045 | | | |
| **TeraSort_1T** | **AVERAGE** | **rel.stddev** | **AVERAGE** | **rel.stddev** | **improvement** | | | | | | | | |
| Baseline | **24,887.284** | 5.693% | **24887.284** | 5.693% | 0.000% | 24968.600 | 23281.147 | 25786.129 | 25094.860 | 23363.417 | 27341.413 | 24375.423 | |
| BIOS | **24,777.112** | 4.124% | **24777.112** | 4.124% | 0.443% | 26007.764 | 24036.519 | 25709.582 | 24406.622 | 23725.072 | | | |
| build102 | **24,948.351** | 2.969% | **24741.161** | 2.660% | 0.587% | 24029.427 | **26102.162** | 24245.930 | 25714.404 | 24913.017 | 24803.029 | **24830.488** | |
| build103 | **24,438.467** | 3.175% | **24438.467** | 3.175% | 1.803% | 23676.232 | 25487.552 | 24073.980 | 25020.949 | 23933.622 | | | |
| build104 | **24,675.834** | 4.695% | **24567.464** | 4.690% | 1.285% | 26772.865 | 24244.314 | 24759.633 | 23094.801 | 24975.596 | 23823.902 | **25059.726** | 24301.135 |
| fsopts | **24,800.513** | 3.019% | **24842.674** | 3.452% | 0.179% | 24631.870 | 25531.615 | 23985.241 | 24225.968 | 25627.873 | | | |
| notranshuge2 | **24,744.591** | 3.442% | **24623.198** | 3.785% | 1.061% | 25230.163 | 25410.815 | 23807.672 | 23824.607 | 25449.698 | | | |
| **TeraValidate_1T** | **AVERAGE** | **rel.stddev** | **AVERAGE** | **rel.stddev** | **improvement** | | | | | | | | |
| Baseline | **4,552.510** | 15.155% | **4312.570** | 6.864% | 0.000% | 4599.195 | 4539.023 | 4147.745 | **5992.151** | 4436.654 | 4350.785 | 3802.019 | |
| BIOS | **4,293.203** | 14.740% | **4293.203** | 11.339% | 0.449% | 4484.602 | 3889.211 | 5077.439 | 3449.001 | 4565.761 | | | |
| build102 | **4,307.852** | 13.454% | **4307.852** | 14.695% | 0.109% | 4886.813 | 5231.142 | 3648.406 | 4207.900 | 4249.794 | 4234.767 | 3696.141 | |
| build103 | **4,102.848** | 14.127% | **4102.848** | 10.950% | 4.863% | 3911.592 | 4526.398 | 3737.375 | 4871.358 | 3467.519 | | | |
| build104 | **5,309.626** | 35.743% | **4674.657** | 23.454% | -8.396% | 4765.060 | 3676.659 | 4656.453 | 4133.038 | **8834.079** | 4085.219 | 7016.874 | 4389.294 |
| fsopts | **4,179.044** | 12.266% | **4248.740** | 13.272% | 1.480% | 3598.641 | 4251.059 | 4171.684 | 4973.574 | 3900.262 | | | |
| notranshuge2 | **4,314.332** | 11.762% | **4472.866** | 11.346% | -3.717% | 4973.826 | 4021.914 | 4637.445 | 4258.277 | 3680.198 | | | |

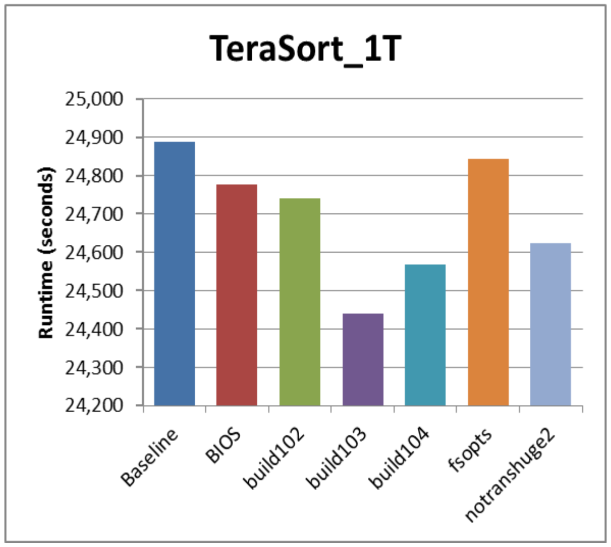
## Appendix F – Overview of results

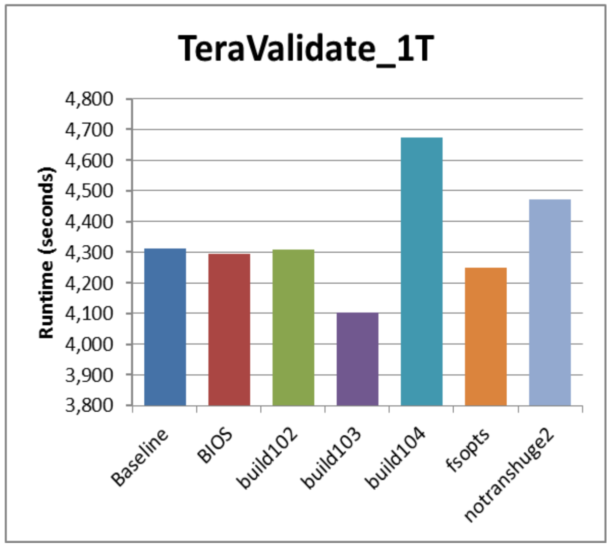

TeraGen_16G
Runtime (seconds)
Baseline
BIOS
build102
build103
build104
fsopts
notranshuge2
preformat2
preformat3
permalogs


TeraSort_16G
Runtime (seconds)
Baseline
BIOS
build102
build103
build104
fsopts
notranshuge2
preformat2
preformat3
permalogs


TeraValidate_16G
Runtime (seconds)
Baseline
BIOS
build102
build103
build104
fsopts
notranshuge2
preformat2
preformat3
permalogs


TeraGen_1T
Runtime (seconds)
Baseline
BIOS
build102
build103
build104
fsopts
notranshuge2


TeraSort_1T
Runtime (seconds)
Baseline
BIOS
build102
build103
build104
fsopts
notranshuge2


TeraValidate_1T
Runtime (seconds)
Baseline
BIOS
build102
build103
build104
fsopts
notranshuge2