I/O Load Scheduler for GRID Mass Storage

CHRISTOS TZIORTZIOS

Christos.Tziortzios(at)os3.nl

February 13, 2012

Abstract

SARA manages a high performance data storage system in order to store data for LHC and LOFAR experiments. The amount of data that are currently stored is more than 5 PB. They are using a Hierarchical Storage Management (HSM) approach in order to achieve that. There is a disk front-end and a tape back-end, in which data are stored permanently and restored when researchers need that data. The Front End Storage (FES) is using the dCache software in order to manage the data and the transfer requests from the research sites. Data are transferred from FES to the tapes and the other way around, through the Grid Mass Storage (GridMS). The data transfers are scheduled on the Front End Storage side in a basic way. In this project we created a prototype solution of implementing a scheduler in the GridMS side, in order to control the data flow between FES and GridMS more efficiently. We used TORQUE resource manager and Maui cluster scheduler in order to achieve this. Putting data from the GridMS disk to tape and getting it from tape to disk is taken care of by the Data Migration Facility, thus this part of the process is not going to be controlled by the scheduler. Last, we propose some Maui and TORQUE configuration characteristics and some modifications to the current working environment that could make it more efficient.

Contents

1 Introduction

SARA [1] manages a high performance data storage system used, among other things, to store data from the LHC [3] (particle accelerator in Switzerland) and LOFAR [2]. SARA uses a disk front end and a tape back end. Data are copied from a remote host to the disk cache (Front End Storage) and then stored on the tape back end, through Grid Mass Storage (GridMS). Reading in data sets from tape to the disk cache and then transporting it back to a remote host also occurs. This process is referred to as data staging. SARA deals with enormous amounts of data. This means that they may need to store and restore many terabytes every day, while a single request could be for a file of even 1TB. Currently, the requests are scheduled in the Front End Storage side, in a non-advanced way.

The research question is:
Is it possible to use an intelligent scheduling mechanism in order to control the data flow between the Front End Storage and Grid Mass Storage more efficiently?

Batch System Schedulers are used in order to give processes access to resources, but normally CPU time and memory. However, our goal is to use schedulers to efficiently control I/O. We will use TORQUE resource manager and Maui cluster scheduler in order to achieve this.

2 Current Infrastructure

2.1 Storage Infrastructure

Hierarchical Storage Management (HSM) [5] is a technique used to move data from high-level expensive storage facilities to cheaper ones in the lower level of hierarchy, while preserving a single view of the filesystem. The storage infrastructure in SARA is organized in a hierarchical way as follows (also see figure 1):

- Front End Storage (FES): Front End Storage receives requests from LHC and other projects such as LOFAR. Data are sent from these project sites to the FES which has a capacity of approximately 4.6 PB, however this disk space is shared with projects other than the one discussed in this paper. Later on, researchers may request to retrieve that data. FES consists of 48 nodes, with 10 GE (10 Gbps) network links to the Grid Mass Storage (GridMS) front-end network. The FES uses dCache, which is a storage middleware system [4], capable of managing the large amounts of data created by projects such as the LHC and LOFAR.
- Next, data are sent to the Grid Mass Storage (GridMS) in order to be stored permanently. GridMS consists of:
  - 4 Data Movers (DM)
  - 5 Tape Movers (TM) with 4 tape drives each
  - the Data Migration Facility and
  - a 33 TB disk

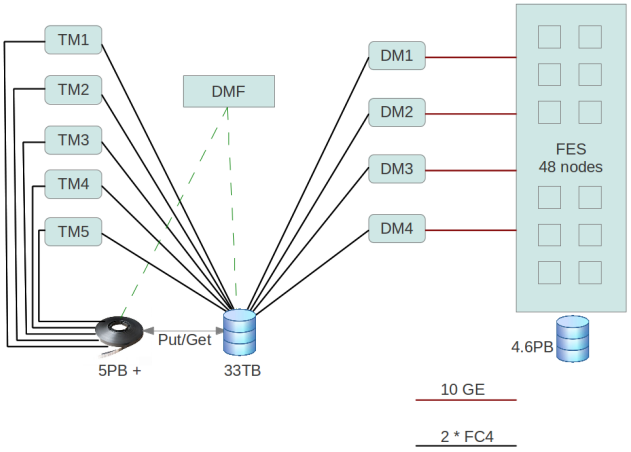

Figure 1: Current Architecture

- each DM node has a 10 GE link to the FES
- each DM and each TM has two Fiber Channels of 4 Gbps links to the GridMS disk

Data are sent from the Front End Storage to the Data Movers to be stored. First, data are sent to the 33 TB disk. Next, the data are put to tape, in which they are stored permanently.

In order to restore data, a request is sent by the FES to the Data Movers. If the data is on the disk, for example because the same file has been requested recently, it is sent to the FES. Otherwise the data needed must be restored from tape to disk. Then the data are made available for the FES. Putting data from disk to tape and getting from tape to disk are both taken care of by the Data Migration Facility (DMF). The DMF is a Hierarchical Storage Management system and its purpose is to efficiently manage the data transfers from disk to tape and the other way around [6].

2.2 Operations

It is extremely important that these operations are flawless, since a minor flaw could potentially lead to loss of data.

2.2.1 Operations between FES and GridMS

Store When a file needs to be stored, the file is copied from a FES node to the GridMS disk, using gridFTP [8]. The protocol used is sshftp, but instead of the standard ssh, hpn-ssh [9] is used. The advantage of hpn-ssh is that there is no encryption of the data, while retaining the strong authentication mechanism. This helps data be transferred at higher rates. When the data are moved to the GridMS, they are checksummed. FES will then check if their checksum is the same as the one in the GridMS and if the file is intact the data may be removed from the FES.

Restore When data need to be restored, the procedure is the following:

1. The user (essentially FES) is connected to one Data Mover through ssh and runs a dmget command. dmget is used by the DMF to recall files that have been stored to tape.
2. Next, the FES repeatedly checks if the dmget call has completed.
3. When the dmget call is found to be completed, the data are copied to the FES.
4. If the FES gets the data successfully, the disk space in the GridMS is freed up since the tape copy is retained.

2.2.2 Checksums

Data are checksummed with the Addler-32 algorithm on the FES side and afterwards on the GridMS side. The checksums are checked when the file is stored and when the files are restored from tape in order to make sure that no errors have taken place throughout the procedure. Adler-32 is not the most reliable checksumming algorithm but it was found sufficiently adequate for our purposes and it performs very fast.

2.2.3 Operations between GridMS disk and Tape

Put When data needs to be put on tape, the Data Migration Facility (DMF) takes care of copying the file from the GridMS disk to tape and freeing up the disk space. The DMF monitors the filesystem and when there are enough data that need to be moved to tape, it takes care of the process at will.

Get When data needs to be retrieved from tape, dmget command is called and the DMF takes care of getting back the file on disk.

2.2.4 Writing to and Reading from disk

The 33TB GridMS filesystem is a striped volume that is built up from RAID6 LUNs. This technique is applied for performance reasons. However, more than one files are being written to or read from that filesystem concurrently and this leads to some drop in performance.

2.3 Performance issues

Bandwidth bottlenecks Front End Storage consists of 48 nodes. However, on the GridMS part there are only 4 nodes (Data Movers). Thus, the GridMS nodes are too few to handle the whole data load that the FES nodes could provide. However this is not the greatest problem, since the GridMS disk has an even lower bandwidth (also see figure 2). The performance of the whole system is limited by the low disk I/O rates, which is the real bottleneck. There is 1 GB/s of bandwidth for I/O for the DM nodes and 1 GB/s for the TM nodes. An other issue is that each tape drive can read from or write to tape at a maximum speed of 120 MB/s. For this reason we need multiple concurrent "puts" or "gets" in order to utilize the bandwidth of the disk.

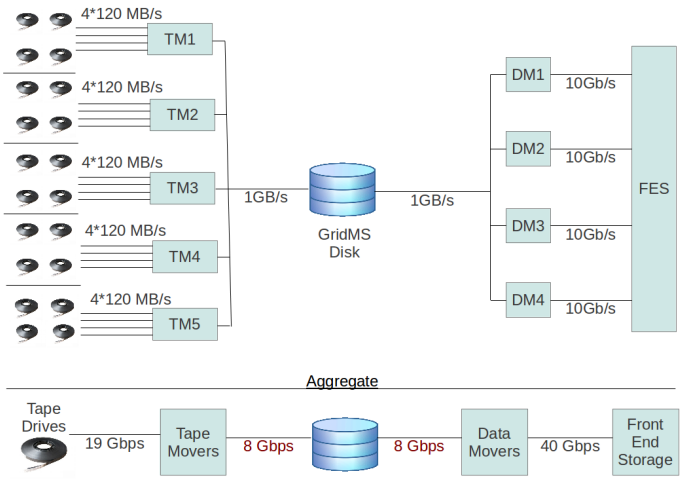

Figure 2: Disk I/O Bottleneck

Disk Input / Output The highest performance achieved between the disk and the DM nodes is approximately 900 MB/s, which is nearly as much as the bandwidth of the disk. However, the average performance is 400 MB/s while performance can drop to even 200 MB/s. Performance depends on the amount of parallel jobs running. On the one hand, we need, for example, multiple concurrent "get" operations, since having one would mean that we could only get 120 MB/s at most (see figure 2). On the other hand, increasing the number of concurrent operations on disk, makes I/O more and more random. There are two different kinds of workload categories, which are sequential and random. In sequential workloads large amounts of data are read at once, while in random access workloads smaller amounts of data are read each time [7] . This leads to worse performance because the data are not parsed at once and the heads need to move a lot from one part of the disk to another. Performance also depends on the size of the files; large files tend to perform better.

Scheduling only on FES-node level Data are sent to the FES from different experiment sites. Later on, researchers all over the world request for files to be restored. Next, the FES requests that data are stored to or restored from tape. As far as the GridMS is concerned, these processes are initiated by the FES. If we allowed each node to send several files to be stored at once, that would lead to sending to the DMs more data than they could handle, thus we allow only one transfer per node at a time. However, using this setup, there is a possibility that there is only one FES node that needs to store lots of data. In this case this node would not be allowed to utilize any other DMs, although they would be idle. Thus, the scheduling performed by dCache in the FES side is not advanced. Scheduling exists only on node level, instead of groups of nodes (e.g nodes dedicated to LHC) or user groups. Using TORQUE resource manager along with Maui Scheduler should provide more options to the storage administrators. These options would let them increase the efficiency of the data

storage. Moreover, it could help providing more "fair" access to the projects and the researchers.

3 Testing

3.1 Testing Environment

3.1.1 Goals

The goal of the following tests is to investigate whether using a batch system would improve the performance of GridMS. We wanted to check if features such as scheduling priorities and resource sharing can be efficiently used in order to schedule the I/O load. Moreover, we want to examine the feasibility of modifying the procedures that are used currently, into procedures that would efficiently use a scheduling mechanism and develop a prototype solution. For these reasons we will be using TORQUE resource manager and Maui cluster scheduler.

3.1.2 Software used for testing

TORQUE TORQUE is a resource manager. Resource managers provide the functionality to start, hold, cancel, and monitor jobs [10]. It is normally used to give processes access to CPU time or memory, but we are mainly interested in scheduling disk I/O and bandwidth.

Maui Although TORQUE has its own scheduler, it is normally configured to use another, more advanced, scheduler; typically Moab or Maui. We are using Maui Cluster Scheduler, which is an open source job scheduler for clusters and supercomputers [11]. Maui is capable of providing scheduling and fairshare options. Maui provides numerous options, through which administrators can give priority to the users that may be more important and make sure that everyone gets what one deserves through fairshare policies.

3.1.3 Setup

Some of the following tests were not done in the optimal way. The main reason is that there was no test environment and we had to use the production environment. Thus we had to be really careful on what we do and we should not perform all of the tests using real store and restore jobs because that would have negative effects to the performance of the working environment. Another side effect is that we would not able to get reliable data if we wanted to calculate the effect of multiple concurrent operations on the disk. We would be able to know how many of our jobs are running but we would not know the number of store and restore operations executing in the production environment. Although we had practically full access to the GridMS nodes, we only had access to one of the FES nodes. All jobs were sent through this node, using different test-users in order to simulate the real requests.

3.1.4 Operations

Restore The Restore procedure of the testing scheme is a simplified version of the one currently used in the working environment (described in the previous

section). The differences are that instead of just running the job, we first submit it in the queue and that we do not copy the file back to the FES.

Store When a file needs to be stored, the procedure is as follows:

1. FES will ssh to one of the nodes of the grid in order to run a job *submitQueue* that will put another job *copyJob* to the queue.
2. When it is time for *copyJob* to execute, depending on the queue, it will run.
3. What *copyJob* does, is run scp, thus copying a file from the FES to the GridMS disk and remove the file from the FES disk upon success of the copy command.

Using scp is not the most efficient way of copying files. In the production environment a gridftp over hpn-ssh command is being run. We did not use the hpn-ssh command because, neither was it installed on the GridMS nodes, nor was there a daemon running on the FES side, in which we had limited access. Moreover, using this set-up, the GridMS would pull the data from the FES instead of having the FES push data to GridMS. This way the GridMS will only pull data when it is ready to. Moreover, in the production environment files are checksummed and the checksums of the FES are compared to the new checksums on the GridMS side as previously mentioned, which was not part of the testing scheme for simplicity.

Put and Get Put and Get operations are, as already mentioned, taken care of by the DMF and we could not adjust the way that the DMF works. Thus, these operations are not part of the testing environment.

3.2 Maui and TORQUE Configuration Tests

The purpose of testing different Maui and TORQUE configurations is to explore the opportunities of scheduling jobs in different ways. Moreover we want to ensure that the scheduler works as expected when different settings are used. In order to achieve this, we first experimented with jobs that would not influence the performance of the working environment. As soon as the first part of testing was successfully conducted, we moved on to storing and restoring random data.

3.2.1 Calculating Job Priority with Maui

"Maui allows jobs to be prioritized based on a range of job related factors. These factors are broken down into a two-level hierarchy of priority factors and subfactors each of which can be independently assigned a weight" [12]. The main components are:

- Job credentials
- Fairshare usage
- Requested job resources
- Current service levels

- Target service levels
- Usage

Each of the main components has several subcomponents. For example, some of the subcomponents of the "Job credentials" component are:

- User
- Group
- Class

3.2.2 User Priorities

In this test we only wanted to check if all jobs sent by users with higher priority would run before the first job of a user with lower priority.

The following is part of the configuration:

```
#The priority of the job will not increase
#depending on when it was queued
QUEUETIMEWEIGHT         0
CREDWEIGHT              1
USERWEIGHT              1
GROUPWEIGHT             1

#User specific priority
USERCFG[testusr1] PRIORITY=500
USERCFG[testusr2] PRIORITY=500
USERCFG[testusr3] PRIORITY=800
USERCFG[testusr4] PRIORITY=1000
```

Numerous jobs were sent to the queue by different users. As expected, the jobs of testusr4, who had the highest priority would run first, the jobs of testusr3 would run next and the jobs of the other users would be the last to run. It should be noticed that, all jobs of testusr1 would run before the first job of testusr2 would, although they had the same priority.

3.2.3 Fairshare and Resource parameters for priority

The purpose of this test is to check the "Fairshare" and the "Requested job resources" components. The Fairshare component is designed to ensure that different users will get the amount of resources that they are entitled to. The Fairshare component uses short term historical data in order to keep track of how much of the resources each user has used. For this test we used short time windows of 10 minutes in order to get to our results faster.

Moreover, the jobs that we sent to the queue were "tagged" with different walltimes in order to give them different priorities. The reason why walltime was chosen is that larger files should take longer to be "processed". We chose to give higher priority to jobs with longer walltime. If we wanted to give higher priority to jobs with shorter walltime, we would have to make sure that the walltime would still be long enough for the jobs to execute.

The following is part of the configuration:

```
#The priority of the job will not increase
#depending on when it was queued
QUEUETIMEWEIGHT         0

#Fairshare weight
FSWEIGHT                1
#longer jobs will have higher priority
RESOURCEWEIGHT          -1
WALLTIMEWEIGHT          1

#All users have the same priority
USERCFG[testusr1] PRIORITY=500
USERCFG[testusr2] PRIORITY=500
USERCFG[testusr3] PRIORITY=500
USERCFG[testusr4] PRIORITY=500

#All groups get the same share of resources
GROUPCFG[testgrp1] FSTARGET=20.0
GROUPCFG[testgrp2] FSTARGET=20.0
GROUPCFG[testgrp3] FSTARGET=20.0
GROUPCFG[testgrp4] FSTARGET=20.0

#Configure time windows
FSPOLICY DEDICATEDPS
FSDEPTH 24
FSINTERVAL 00:10:00
```

In this configuration, all users have the same priority and fairshare target. Thus, the order in which jobs will be processed should only depend on their walltime. After some time, if one user has been running some CPU time consuming jobs we should see that this user's jobs will not be taken into execution any more. When the rest of the users get to their fairshare target, this user should be able to run jobs again. If one user has not been running jobs at all and at some point sends one job, this job should be processed soon. However, this may not be the case if this job has a small walltime and the other jobs have really big ones. testusr1 sent a number of jobs with big walltime and got to run them. However, after 10 minutes of having full access to the resources, as soon as these jobs were executed the rest of the users (testusr2 and testusr3) got their jobs with the biggest walltime running, as expected. When these users got their share of the resources, testusr1 was able to run jobs again. Next we sent a number of jobs for testusr4 and they were put at the top of the queue as expected.

3.2.4 Requesting Resources

When submitting a job to the queue, it is possible to specify the resources that this job is going to need in order to run. Among other things, one can request "Walltime", memory usage and most importantly in our case disk space requirements.

Walltime Walltime stands for the amount of time that takes one job to execute. Time, in our case, is as perceived by humans, not the machine, therefore it is measured in seconds, minutes and hours. This amount of time is estimated by the users. If our estimates are accurate, this may lead to improved performance of the scheduler, since Maui performs backfilling [13] [14]. In our case, we could possibly estimate walltime based on the size of the file and the amount of jobs being run at that time.

Disk Space When submitting a job, one can request for the disk space that one's job is going to need in order to be executed properly. The job will not run, until enough disk space is available. This is useful to us, since the jobs that we want to schedule are related to data transfer. Both store and restore operations require disk space in GridMS and since we can get the size of each file that we need to transfer, we can request for the disk space that we need. This way we would be able to accept any number of jobs from the FES and queue them until enough resources are available. However, when we tried to use this feature, we found out that it did not work properly.

Let there be 1 TB of free disk space. If we request for 1.2 TB when we submit a job, then this job will not run until there is enough space. However, if we send two jobs that request 600 GB each, then both jobs would run, although they do not "fit" in the disk. The result is that one of the operations will return an error.

Other resources Although the rest of the resources are not that important for us for scheduling, we can still use them in order to give different priorities to jobs. For example, we could configure Maui in a way that will favour jobs that need more virtual memory. In this way we could for example request a lot of virtual memory for a job that transfers data from a node that is almost full of data, freeing up space as soon as possible.

Underestimating and Overestimating Resources Underestimating resources may lead to the job starting to execute and being killed as soon as it uses up its resources. For example we estimated the walltime for a job as 1 minute, while in fact our job needed 2 minutes to complete. When our job was the only one running, underestimating resources was not a problem. However, when there were other jobs queued, then at the end of the time that we requested for our job, Maui gave the resources to one of the jobs in the queue, thus killing our job. Overestimating walltime will not cause such problems, but it may lead to less efficient scheduling. It may lead to having low-priority jobs running before high-priority jobs, which may be annoying. For these reasons it could be useful to get accurate estimates of walltime, but it may be risky as well.

3.2.5 Different Queues

When using TORQUE, one can configure multiple queues. This can be useful, since one could assign different jobs to different queues. In our case we could have a queue that can run up to 2 jobs at a time and send all jobs related to LOFAR to that queue. Another set-up could use one queue for store operations

and another queue for restore operations. This way we could have more parallel restore operations, which is more useful than having parallel store operations. However in both cases we could end up having idle nodes. Each queue may have a different priority. However when we tried to send jobs of equal priority to queues of different priority, Maui would override any priority given by TORQUE or the end user.

3.2.6 Test Conclusions

We managed to check the way priorities and the fairshare components work. The "disk space" request feature did not work properly. Moreover, we were able to give higher priority to different jobs by using the "walltime" feature. The procedures that were used in order to store and restore data were not optimal, for example we did not use the most efficient data transfer protocol, however we did manage to use the queuing system successfully in order to do these data transfers.

4 Proposed Architecture

In this section we propose a way in which the operations can be performed after implementing the scheduling mechanism.

4.1 Operations

The store and restore operations will be using the same file transfer protocols as the ones currently used in the production environment (see subsection 2.2). The same applies for the checksum algorithm used, which is still going to be Addler-32. However, instead of executing the "store" and "restore" operations, jobs are going to be submitted to the queue first. Checksum operations can be part of the store and restore operations or be submitted to the queue separately. Currently data are pushed from the FES to the GridMS for the store operations and pulled from the GridMS side to the FES in restore operations. Inversing this could improve the performance of the production environment. Jobs will still be submited from the FES side, however the jobs will first be queued. Next, the scheduler that will be implemented in the side of the GridMS will be running as many concurrent jobs as the GridMS can handle.

Store Regarding the "store" operation, only few changes are needed. The FES, which initiates the request as far as the GridMS is concerned, will need to submit a request for a file transfer to the queue, instead of transferring the file at once, but no further changes are needed.

Restore Splitting the "restore" operation into multiple jobs that will be submitted to different queues can be helpful. As previously mentioned, the way restore operations currently work is by first requesting that the DMF gets the data from tape to the GridMS. Afterwards, the FES repeatedly checks if the GridMS disk has got the data and then moves the file from the GridMS to the FES. A more efficient way could be the following:

1. First, the FES will ssh to GridMS and run a script.

2. When this script runs, it will request that the DMF gets the data from tape (dmget command). This dmget command can either be submitted to queue or run at once.

    - Having the dmget command run at once may increase the efficiency of the DMF. This way the DMF is given more options in order to schedule its operations more efficiently and have multiple tape drives working at the same time.
    - Submitting the dmget command to the queue, can increase the level of control on which transfer is going to take place, based on credential-based priorities and fairshare criteria. However, we can run dmget outside the queue and still have some control on this, by queuing the rest of the commands.

3. Next, instead of having the FES repeatedly check (through ssh) the status of the file, a second job will repeatedly check the status of the file locally. This job does not need to be submitted to the queue.

4. When the previous job succeeds (the file has been put to the GridMS disk), it will submit another job to the queue. This job will copy the data to the FES and remove it from the GridMS disk. Through this job we can apply credential-based priority and fairshare policies as well.

4.2 Time Windows

One common approach, when backing up data is doing your backups at night, while the network is less busy. However we are dealing with an archiving problem. Requests to get data are random and so are the requests to store data, so the pattern of more restore operations during daytime does not apply in our case. Therefore, giving higher priority to restore operations during daytime and higher priority to store operations at night will not necessarily improve the performance.

4.3 Node allocation policy

Besides using the scheduler in order to determine which job is going to execute, we can also choose which node it will be run on. Node allocation policy is one of the parameters of Maui configuration that are important for us. We chose to allocate the jobs to nodes based on the CPU load. This way one node will have to run 2 jobs only if the rest of the nodes already run one job. The most important part is that by balancing the CPU load, we also balance the bandwidth used, since what these jobs mostly do, is transfer data.

4.4 Optimal number of data transfers

We did not have enough FES test nodes available in order to determine the optimal number of concurrent data transfers. Other users would be able to transfer data outside the queuing system, therefore our results would not be reliable. In theory, one single store operation would be able to use the whole bandwidth available on the GridMS disk. However, this does not happen in practice and multiple store jobs should run in order to use the whole bandwidth

that the GridMS offers. It should be noticed that DMF will wait until there are enough data that need to be stored, before it starts putting it to tape. On the other hand, we definitely need multiple restore jobs running at a time. One get operation may use a bandwidth of up to 120 MB/s, thus, in theory we can have up to 8 get jobs running at a time, without fully utilizing the 1 GB/s of bandwidth that the disk has to offer.

5 Conclusion

A prototype solution was created, that used TORQUE and Maui in order to handle the GridMS disk I/O. The scheduling mechanism can increase the efficiency of the current working scheme. First, we would no longer need to limit the concurrent data transfers per FES node to one. Using the scheduler any number of jobs can be submitted to the queue per node. The optimal number of concurrent jobs can be calculated and used in order to increase the efficiency of the scheme. Maui will then decide which job is going to run. Moreover, Maui lets us give different priorities to jobs and lets users share the resources in a more fair way. Furthermore, the restore operation proposed (subsection 4.1) can improve the efficiency of the scheme even more. Instead of repeatedly checking the status of the file through ssh, this operation can be performed more elegantly by having the GridMS check the status of the file locally.

Implementing a scheme that uses a scheduler is feasible. However, any changes to the production environment must be made with care. The script that is currently used includes a lot of code related to logging and error handling. Implementing a new scheme may introduce new bugs, potentially leading to loss of data. For these reasons, a testing environment should be created and a long testing period should be used. Luckily, the FES is capable of storing a lot of data and could possibly handle the data produced during this period of testing.

References

[1] *SARA* www.sara.nl

[2] *LOFAR* www.lofar.org

[3] *Large Hadron Collider* lhc.web.cern.ch/lhc

[4] *dCache* www.dcache.org

[5] *Hierarchical Storage Management*
http://en.wikipedia.org/wiki/Hierarchical_storage_management

[6] SARA *Data Migration Facility*
http://www.sara.nl/systems/shared/software/dmf

[7] Darren Hoch, *Extreme Linux Performance Monitoring Part II* http://www.ufsdump.org

[8] *gridFTP* www.globus.org/toolkit/docs/latest-stable/gridftp

[9] *HPN-SSH* http://www.psc.edu/networking/projects/hpn-ssh/

[10] *TORQUE* www.adaptivecomputing.com/products/torque.php

[11] *Maui* www.clusterresources.com/products/maui-cluster-scheduler.php

[12] *Maui Job Priority Factors, Maui Scheduler Administrator's Guide* www.adaptivecomputing.com/resources/docs/maui/5.1.2priorityfactors.php

[13] V. Subramani, R. Kettimuthu, S. Srinivasan and P. Sadayappan *Distributed Job Scheduling on Computational Grids using Multiple Simultaneous Requests* Proceedings of the 11 th IEEE International Symposium on High Performance Distributed Computing HPDC-11 2002

[14] *Backfill, Moab Workload Manager Administrator Guide* http://www.adaptivecomputing.com/resources/docs/mwm/6-0/8.2backfill.php