VIP ANR-09-COSI-03
Virtual Imaging Platform

D2.2.2: Optimized file access system for results visualisation

	(in alphabetical order)	
Sorina Camarasu-Pop	CREATIS	pop@creatis.insa-lyon.fr
Rafael Ferreira da Silva	CREATIS	rafael.silva@creatis.insa-lyon.fr
Tristan Glatard	CREATIS	glatard@creatis.insa-lyon.fr
Baptiste Grenier	maatG	bgrenier@maatg.fr
David Manset	maatG	dmanset@maatg.fr
Jérôme Revillard	maatG	jrevillard@maatg.fr

Abstract

This deliverable describes how a file access system for results visualisation has been created using the so-called data manager, providing the Virtual Imaging Platform (VIP) with a better reliability by avoiding failures related to data management.

Contents

1 Introduction

This deliverable is part of the Virtual Imaging Platform (VIP) task 2.2.2 - "multiplatform data storage system".

After describing the uses cases addressed by the data manager, the internals and usage of the data manager will be presented.

2 Uses cases

2.1 Introduction

While grid can store and transfer important volumes of data with a high-throughput, it is not suitable for applications manipulating collections of small files that are sensitive to transfer errors and to latency. In fact many experiments designated data transfers as an important cause of job failures [1].

Figure 1 presents an overview of the VIP platform architecture. For data storage, VIP relies on the European Grid Infrastructure (EGI)[1] three-tier data management. At the lowest level, files are stored at different sites on disk or on tape using Storage Elements (SE) such as DPM, dCache, STORM or Castor. All these SEs expose a homogeneous SRM interface through which files can be listed and transfers can be requested. SRM locations are registered in a Logical File Catalog (LFC) that provides a single indexing

[1] http://www.egi.eu

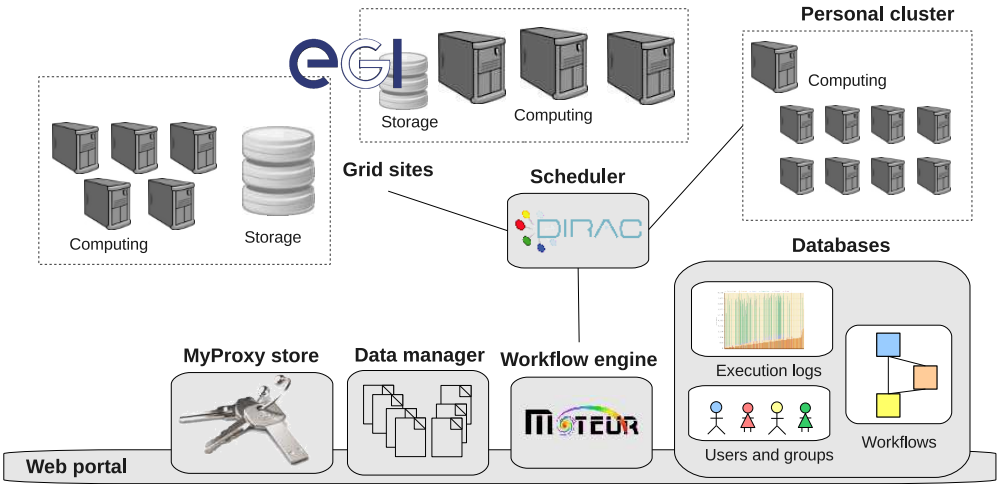

Figure 1: Architecture of the Virtual Imaging Platform.

space for distributed files. File replication is enabled by linking several SRM locations to the same logical name. This architecture can handle replication but it does not provide any mechanism to guarantee file availability. With availability statistics ranging from 80% to 95%, it is critical for applications to set up their own data management system.

In VIP, critical input files such as jGASW bundles are replicated daily on a few SEs to reduce their sensitivity to SE unavailability. Files are also cached by pilot jobs so that tasks executed by the same pilot do not transfer the same file multiple times. To speed-up transfers and reduce the load on the SEs, output files are stored on the site where they are produced; in case the computing site has no SE or it cannot be accessed, files are transferred to a central SE. These measures are useful but not enough. As shown in [1], the job error rate coming from data transfers is still in the order of 5% to 10%.

Inside VIP several use cases were identified: file transfers to and from grid worker nodes, file transfers to and from users and permanent data storage.

Each of the use cases was studied to find an appropriate solution, the data manager, a VIP local storage used as a complement to the grid SEs was implemented as a global solution to address these use cases.

2.2 File transfers to/from grid worker nodes

Different kinds of jobs are running inside VIP but they all face the same issues regarding input and output files.

2.2.1 The different types of jobs

Jobs are executed on grid worker nodes from 100+ computing sites, they have to transfer input files before the execution starts. These input files are stored on a few gLite Storage Elements (SE) and are transferred using `lcg-cp`.

Output files are stored either on the site's default SE or on a central SE when the default SE is not accessible or not properly configured. Output files are transferred using `lcg-cr`.

Two kinds of jobs are represented in simulations, "simulation jobs" and "merging jobs".

Simulation jobs

Simulation jobs compute the core of the simulation.

Input files

- 1 executable (a few KB) ;
- 1 tgz containing dependencies (up to 100MB for Matlab applications) ;
- 1 organ model (from 10KB to 300MB) ;
- parameter files (from 1 to 3 5-KB files).

Output files

- 1 tgz (up to a few MB) ;
- 1 result snapshot (up to a few MB).

Access pattern: up to some 2,000 concurrent jobs, each of them requiring all the input data files.

Merging jobs

Merging jobs assemble the final simulation result from the simulation jobs.

Input files

- 1 executable (a few KB) ;
- 1 tarball (.tgz) containing dependencies (up to 100MB for Matlab applications) ;
- from 50 to 2,000 simulation results (a few MB each).

Output files

- 1 to 3 files, up to a few 10MB each.

Access pattern: 1 to 5 jobs only, each of them requiring all the input data files.

2.2.2 Input files

Input data transfers are hampered by two issues:

- (1.a.i) reliability: data transfers currently account for 85% of the job failures during workflow execution ;
- (1.a.ii) performance: it takes ages for merging jobs to transfer their input data.

Reliability issues are due to:

- Configuration of the computing site, e.g, SE is not in local BDII ;
- Scalability issues of the SE: the SE starts dropping a lot of incoming connections ;

- Temporary downtimes of the SE ;
- Poor connectivity causing connections timeouts.

Performance issues are due to:

- latency issues: transferring 500 files takes several hours, even if the file are very small ;
- throughput issues: some sites have a very low network throughput to/from external SEs, resulting in very long data transfers when files are heavier than 10MB.

Input files are read-only.

Proposed solution

For (1.a.i), input files can be cached by MOTEUR on the platform when workflow execution starts. If input files cannot be transferred to the worker node, the job connects back to the platform and fetches the files. The cache should be large enough to host the complete simulation input and output during the duration of the workflow execution, i.e. up to 10GB. Connections from worker nodes to the cache would only occur in case of download issue, i.e., for about 10% of the jobs. (1.a.ii) is more tricky but a possible solution is that MOTEUR informs the cache that files belong to the same "bundle". Bundles are assembled by the cache in a tgz. Jobs can request bundle transfers to the cache.

2.2.3 Output files

Reliability is the main concern here since output transfer errors are detected after the job finished. If the data cannot be transferred, it often means that a few-hour-long computation has to be resubmitted. An upload test is done before the execution of the job to detect output transfer issues as soon as possible. In spite of this, some lcg-cr errors sometimes occur due to:

- misconfigured site default SE and central SE enters downtime or starts facing problems while the job executes
- file cannot be registered in LFC because of load

A reliable backup solution has to be found to temporarily store the output files. Performance is of course a concern but file bundles are easily created by jobs and uploaded at once to the storage system.

Proposed solution

Uploading results on a cache hosted on the platform could be a solution. Once safely uploaded to the cache, files should be attempted to be transferred to the grid on a short-term. Jobs may require the file before it can be uploaded to the LFC. In this case, transfer from the LFC would fail and the jobs will fall back on the cache. Therefore the file unique identifier in the platform has to remain the LFN. In some cases, upload issues may be permanent (e.g. the proxy does not contain a VOMS extension or the user is not registered in the proper VO group). In this case the cache should be able to send notifications to the platform admin to avoid that files are kept permanently in the cache,

which would result in many jobs connecting to the cache to fetch this file. This requires that at least the LFN can be registerd by the job to ensure that file permissions are defined. Figure 2 shows a sequence diagram for this scenario.

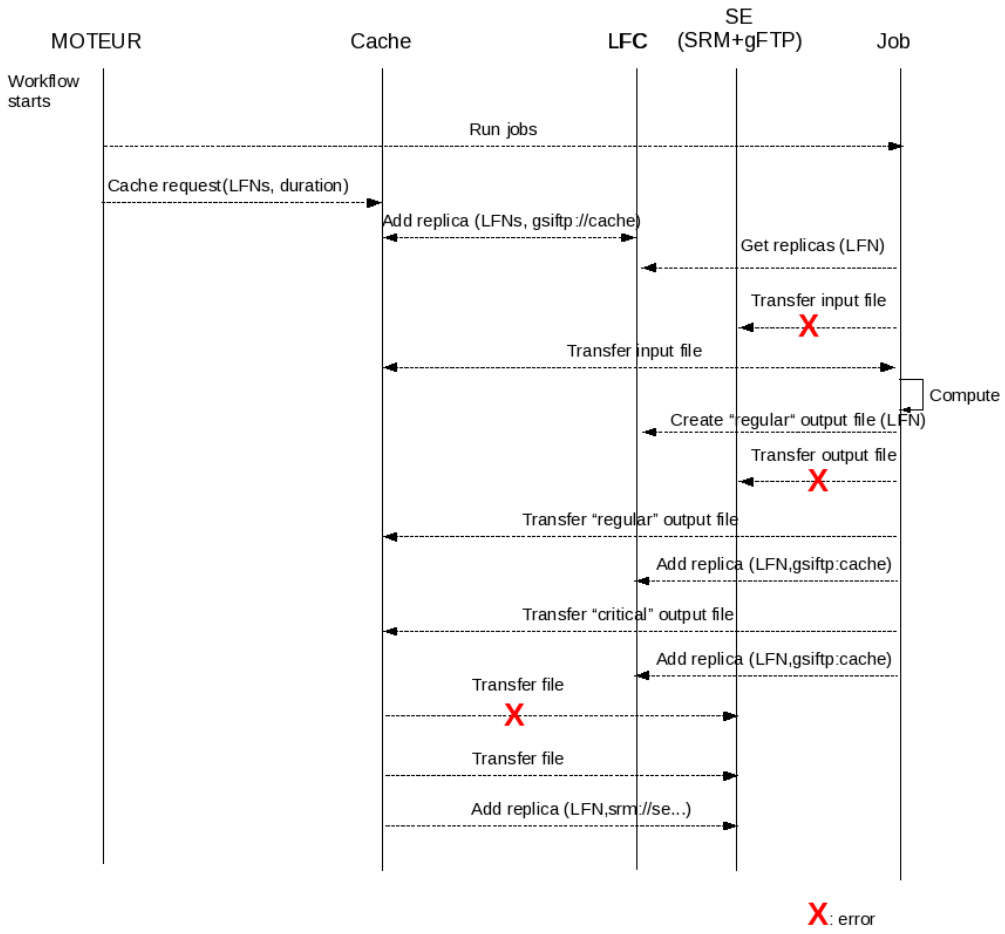

Figure 2: File transfers to/from grid worker nodes

2.3 File transfers to/from users

When a workflow execution completes, the user often wants to inspect the results during the following 24 hours. This means that the results should be quickly transferred to the user host. In most cases, only results of the merging procedure (i.e a few files only) have to be transferred. A few simulation results could also be inspected for consistency. In some cases (e.g. in case of reconstruction issues) all the simulation results (up to 2,000) have to be transferred.

The main issues are:

1. reliability: simulation results may be stored on 100+ different SEs and it is very likely that one of them is not accessible from the user host (or the gateway host) due to SE downtime or connectivity/configuration problem between the local host and the SE.

2. performance: transfers from SEs have a very long latency, often of a few seconds and inspecting a few random simulation results can be painful.

Similarly, users will upload data (e.g. models, simulation input files) to the plaftorm and the cache should ensure interactivity. If files can't be uploaded to the grid, they should remain cached and the system should periodically retry to upload them.

Proposed solution

Grid storage is not meant for user interaction. A more reactive system has to be put between the grid and the users. Caching (some) workflow results for a few days on the platform could allow interactive result inspection. File caching could be triggered by MOTEUR since it can distinguish between results and temporary files. Similarly, file uploads from users could target the cache which would asynchronously try to transfer the data to the grid. A fall back on the SE is done in case the cache is not working properly or down or overloaded (as it is it is a single point of failure). Replica selection has to be changed in the client. Figure 3 shows a sequence diagram for this scenario.

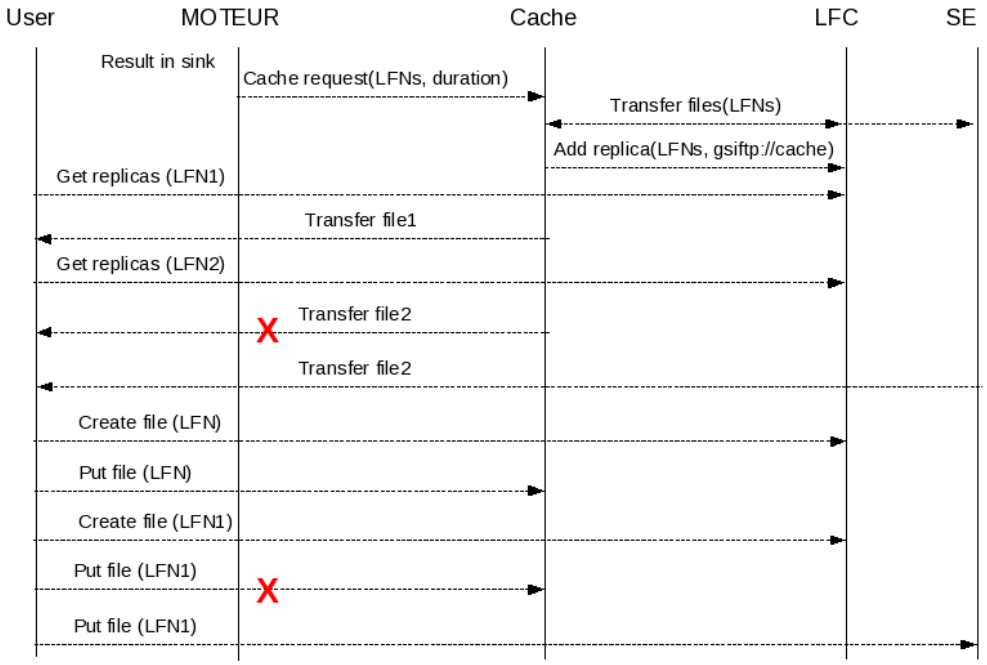

Figure 3: File transfers to/from users

2.4 Permanent data storage

Platform files have to remain stored for a long time, for instance:

- simulator codes and dependencies, to be used in simulations
- organ models, to be used in simulations
- simulation results, to be browsed and retrieved by platform users

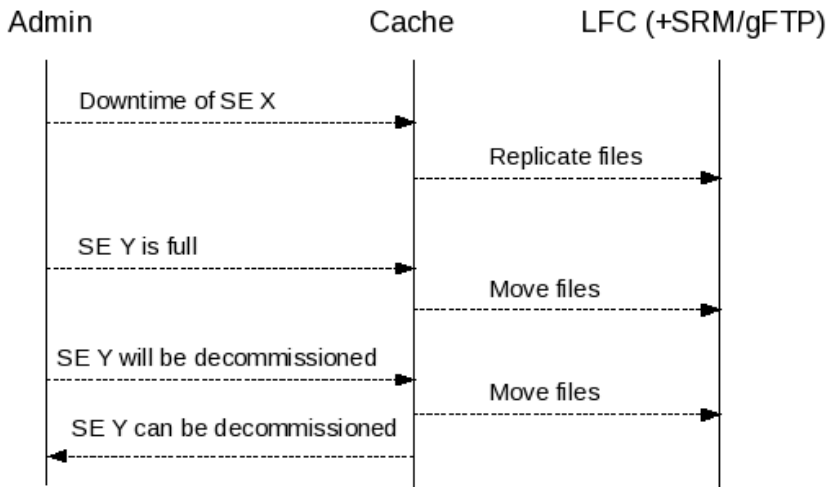

Figure 4: Enabling long-term data storage.

In practice, this requires a dedicated management system to take actions when the following events occur:

- Storage Element is decommissioned: data migration
- Storage Element is full: data migration and/or cleanup
- Storage Element downtime is announced: data migration

Platform data can be considered read-only. Organ models and simulator codes will evolve but this could be handled with versioning.

Proposed solution

The platform could flag some LFC directories as critical. These would be monitored and their files replicated on many SEs to ensure availability. Local backups (in cache) could also be envisaged. Different levels of criticality could be defined. For instance, simulation codes would have a higher criticality than simulation results and workflow intermediate files. Figure 4 shows a sequence diagram for this scenario.

Two policies have to be defined more precisely:

- default policy: how many replicas to make, which files are backed up
- emergency policy: how to react to decommissioning or downtimes? Backup or replicate somewhere else?

2.5 Conclusion: Local storage as a complement to grid SEs

From the previous detailled use cases, a global solution emerged: the solution adopted is to complement grid SEs with a local data manager included in the VIP platform. The data manager is used as a failover storage for files involved in a simulation. It can be accessed both by users and by grid jobs. This use-case is summarized on figure 5.

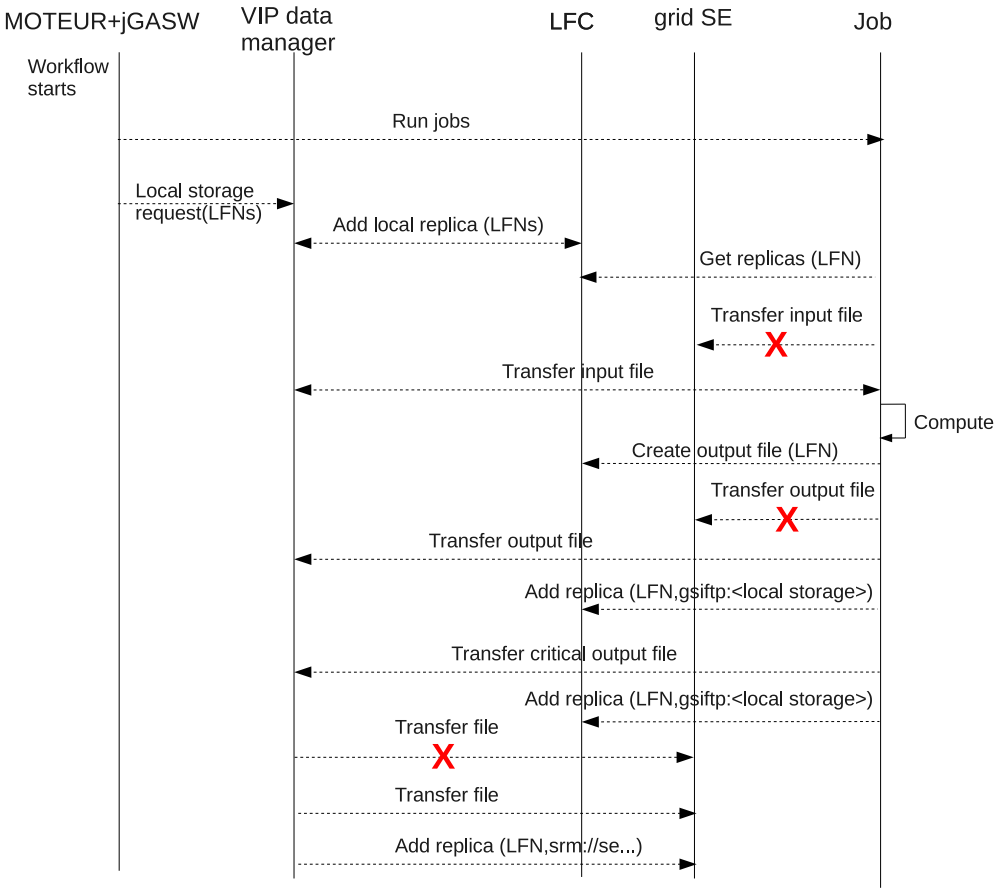

Figure 5: Local storage as a complement to grid SEs. Red crosses indicate transfer errors.

When a simulation starts, MOTEUR replicates the input files on the data manager which is assumed reliable. These local copies are then used as failovers in case input files cannot be transferred.

Jobs can also use it for output transfers when their local SE is not accessible. Periodically, the data manager applies the following replication policy: (i) files are replicated to ensure a minimal number of replicas, (ii) above a certain threshold, replicas are deleted, (iii) files that are properly replicated are removed from the local DPM to ensure that the local storage is always available for critical file transfers, (iv) files that are not referenced in the LFC are deleted from the local storage.

The VIP data manager was implemented as an overlay of the DPM SE. It is seen as a regular SE except that it is not published in the information system. Therefore, it can be used with existing grid data clients and files stored in the local storage can be registered in the LFC as any other grid files[2]. Users can transfer files between their local machine and grid storage by using a file transfer tool. File upload is split in two steps: (i) upload from the local machine to the web portal server and (ii) file transfer to the grid through an asynchronous pool of transfers which is processed sequentially using the VLET library. Similarly, the download process consists in (i) transferring the file from the grid to the portals server machine using the transfer pool and (ii) downloading the file to the user

[2]Checks in the information systems can be disabled in regular data transfer commands (`--nobdii` option).

host. Figure 6 is a screenshot of the portal displaying the on-line monitoring of a workflow and the file transfer tool.

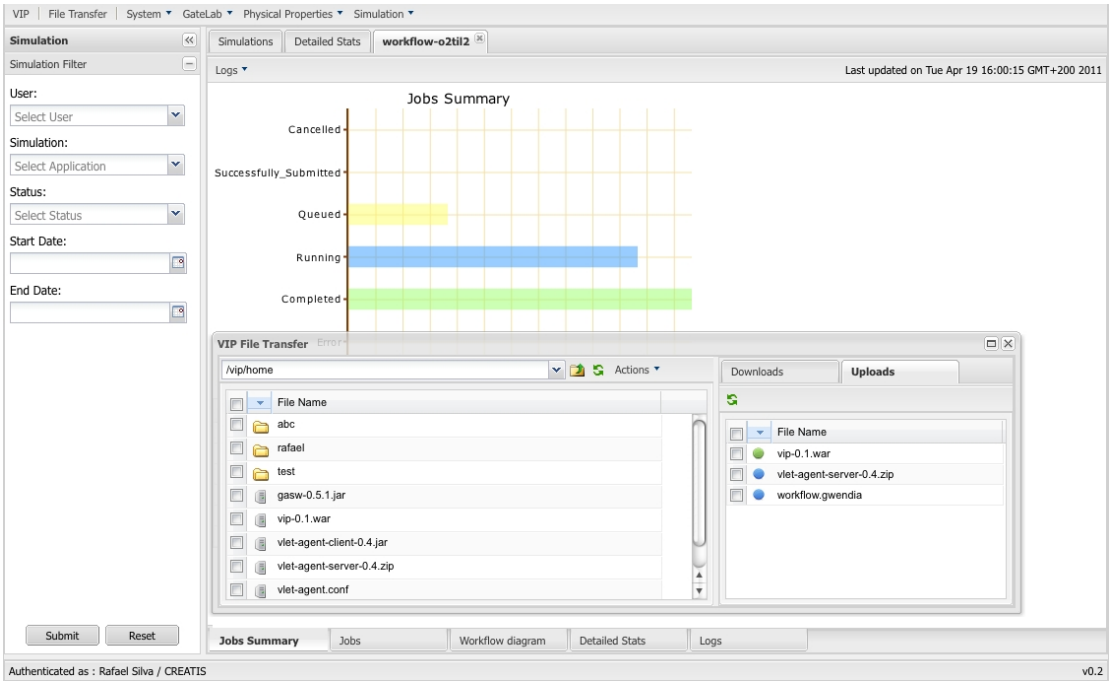

Figure 6: The VIP portal displaying the on-line monitor and file transfer tool.

An experiment was conducted with an ultrasonic simulation to evaluate the impact of the data manager on a simulation running on EGI. Two runs of this simulation were considered. The data manager was enabled in the first one and it was disabled in the second one. The simulation consisted of 128 jobs dominated by data transfers. Each job downloaded 5 input files and uploaded 1 output file. Files were not replicated and an artificial failure rate of 1% was set on the data transfers. It correspond to the realistic case of an execution on the EGI where files are replicated twice[3]. Table 1 shows the impact of the data manager on job reliability. It displays the total number of submitted jobs, the number of data transfer errors and the total number of failed jobs for both runs.

As expected, the data manager zeroed the number of data transfer errors. In this case the remaining 2 failures are due to application errors on the computing nodes. The impact of data transfer errors on the application is important: when the data manager is not used, the observed job failure rate due to data transfer errors is 7.7% (11 failures for 142 submitted jobs). It comes from the fact that the 1% failure probabilities accumulate for all the 6 files transferred by the jobs.

Data manager	Total submitted jobs	Data transfer errors	Total failed jobs
ON	130	0	2
OFF	142	11	14

Table 1: Impact of the data manager on job reliability.

[3]Reported availability/reliability values on EGI are in the order of 90%. If files are replicated twice then the transfer failure rate is 1%.

3 Data manager internals

This section offers an overview of the data manager and how a data management policy is defined.

3.1 Overview

The goal of the `data-mangagement-deamon` is to ease the automatic management of a local DPM used as a cache.

The data-management-daemon inspects recursively a directory of the local DPM to ensure that a policy on files' replicas quantity is applied.

The daemon analyzes recursively a directory of the local DPM, and for each file it checks the number of replicas (by asking the LFC) and then it will apply the policy.

The application use the DPM certificate/key to access the local files, and then it asks MyProxy for a proxy of the file's owner then a VOMS proxy allowing the daemon to interact with the grid on behalf of the file's owner is created using it.

The permissions on the file in the local DPM are copied on the file copied to the remote DPM.

If a file does not have any reference in the LFC(s) (i.e. is orphan) it is deleted from the local DPM.

Detected errors (file unaccessible, no more proxy for the user...) are logged and sent in a summary email at the end of the check.

File access permissions are maintained by the LFC and no further access control is implemented by the data manager.

3.2 Policy

The policy consists of enforcing a number of replica for a file:

- if a file doesn't have enough replicas some will be created until the targeted replica count (`minimum-replicas` option) is reached, with a hard limit of one replica per SE. The replica stored in the cache is not counted as a replica. (it is just a cache copy);
- if the `limit-replicas-number` option is set then the demon will delete the exceeding replicas ;
- if the `cleanup-threshold` value is set and is lower than the space used the application will try to free some space by deleting already well replicated replicas.

4 Data manager usage

The data manager can be launched manually, using its standard Command Line Interface (CLI) or like a unix daemon.

Several files are read by the data manager on start to configure its behaviour:

- `/etc/data-management-daemon/dpmwatcher.properties` is the main configuration file ;
- `/etc/data-management-daemon/mailer.properties` configures the mail-related settings ;

- `/etc/data-management-daemon/log4j.properties` is used to configure the logging system ;
- the initialization script of the daemon additionally reads its configuration in the `/etc/data-management-daemon/daemon.config` file and uses these options as arguments to launch the daemon.

The CLI and the daemon share most of their arguments, but due to its nature the CLI do have some more arguments.

4.1 Command Line Interface arguments

```
-h -help: help
--debug: show debugging output
-v --verbose: verbose output
--version: print version information and exit
-H --dpm-host: hostname of the local DPM
-m --minimum-replicas: minimum number of replicas
-p --monitored-path: path to monitor
-t --cleanup-threshold: cache cleanup threshold
-l --limit-replicas-number: limit replicas number (to minimum-replicas)
```

4.2 Daemon arguments

```
-H --dpm-host: hostname of the local DPM
-m --minimum-replicas: minimum number of replicas
-p --monitored-path: path to monitor
-t --cleanup-threshold: cache cleanup threshold
-l --limit-replicas-number: limit replicas number (to minimum-replicas)
```

4.3 Requirements

The requirements of the data manager, implemented in Java, are quite straight forward.

- a recent JVM (>= 1.5) ;
- should run on a DPM ;
- access to the the DPM's certificate and key ;
- the DPM user certificate should be allowed by MyProxy to access all the proxies.

4.4 Conclusion

The data manager aims at being as simple as possible, by hiding the complexity of its goal. These was achieved by using a gLite CNS - DPNS and LFC compliant - OpenSource Java library for interacting with the DPM's DPNS and with the LFC.

This library, developed by maatG, and freely available, could be found at the gLite Data Management Library project homepage.

5 Conclusion

A data manager addressing the VIP use cases described in the section 2 was presented to complement grid Storage Elements with local reliable storage. The data manager is implemented as an overlay to the DPM Storage Element ; therefore it can be used with the Logical File Catalog and can be accessed using regular grid clients. It holds files involved in the active workflows and it is used as a failover in case grid storage is not available. It includes a daemon that periodically replicate the oldest files to the grid to ensure that incoming transfer requests can always be fulfilled. The tests show that the impact of the data manager is very positive on the number of job failures as explained in section 2.5.

References

[1] T. Li, S. Camarasu-Pop, T. Glatard, T. Grenier, and H. Benoit-Cattin. Optimization of mean-shift scale parameters on the egee grid. In *Studies in health technology and informatics, Proceedings of Healthgrid 2010*, volume 159, pages 203–214, 2010.