Grant agreement no. 211714

neuGRID

A GRID-BASED e-INFRASTRUCTURE FOR DATA ARCHIVING/ COMMUNICATION AND COMPUTATIONALLY INTENSIVE APPLICATIONS IN THE MEDICAL SCIENCES

Combination of Collaborative Project and Coordination and Support Action

Objective INFRA-2007-1.2.2 - Deployment of e-Infrastructures for scientific communities

Deliverable reference number and title: **D11.2 – ACDC2 Test Suite Specification & Report**

Due date of deliverable: **Month 24**

Actual submission date: **January 31st 2010**

Start date of project: **February 1st 2008** Duration: **36 months**

Organisation name of lead contractor for this deliverable: **maat Gknowledge**

Revision: **Version 1**

Project co-funded by the European Commission within the Seventh Framework Programme (2007-2013)		
Dissemination Level		
PU	Public	X
PP	Restricted to other programme participants (including the Commission Services)	
RE	Restricted to a group specified by the consortium (including the Commission	
CO	Confidential, only for members of the consortium (including the Commission Services)	

Table of Contents

EXECUTIVE SUMMARY

One of the main tasks of WP11 is to define a series of validation tests to run within the neuGRID platform, which guarantees its good performance while meeting user requirement specifications. To do so, neuGRID has planned 3 series of Analysis Challenges and Data Challenges (AC/DC1, 2 and 3) as well as 2 series of functional tests called Story Lines (SL1 & 2). In other words, AC/DC challenges aim to measure performances while the SL tests series validate the neuGRID services from the user standpoint. These tests drive and influence the ongoing developments, validating neuGRID's computing model. They are executed at first in the neuGRID PoC environment (development test-bed) and once available, in the PROD environment throughout level 0 and level 1 centres.

This deliverable incorporates AC/DC1 (test code name "Who Made Who ") but focuses on AC/DC2 (test code name "Highway to Hell"). The latter series of tests was applied on a larger infrastructure than the one tested in AC/DC1, with the introduction of the two first DACS sites at level 1. Tests have been carried out between level 0 and level 1 for the sake of refining performance measurements, therefore emphasizing on the same aspects as AC/DC1.

1 Introduction

1.1 Purpose of the Document

This document aims to illustrate how the WP11 team defined a series of validation tests to run in the platform and report on the result of these tests. This work has been carried out the task entitled "*T11.2. Platform Performance Validation, AC/DC Test Series*", which started on month 12 and will finish on month 36 with the following objectives:

«*Specification and execution of the AC/DC Test Series in the infrastructure: This includes the provision of necessary scripting logic to trigger the tests and automate their execution in the system. This task is led by P2 NE in close collaboration with P4 MAAT. A document will be produced by P4 MAAT and P2 NE on M12 describing the AC/DC test series and corresponding results, once applied to the neuGRID infrastructure* »

This current deliverable already updates the D11.1 and will also be updated at M36. Based on the acquired experience, infrastructure recommendations will be provided at the end of the project.

1.2 Document Positioning and Intended Audience

WP11 "*Platform Integration, Performance and Feasibility Tests*" aims (extract from the description of work)

(1) to define a series of validation tests to run in the platform, which guaranty its well performing against users requirement specifications (URS from WP9) (2) to define software releases frequency and policy (3) to provide online collaborative development tools to synchronise partners contributions (4) to setup software deployment repositories, for facilitating deployment, migrations and maintenance (5) to define a gridification model applicable to the existing algorithms, which satisfies the foreseen system architecture and applications' requirements (6) to evaluate the existing algorithms' implementations and requirements both in terms of software and Hardware (7) to design and implement a set of distributed and cooperative optimization methods for facilitating algorithms gridification and their future scheduling within the infrastructure (8) to design and implement a set of interfaces for managing the algorithms in the grid(from algorithm publication, to versioning, to training, to sharing). (9) to gridify, deploy and test the algorithms in

the grid infrastructure, (10) to define adapted scheduling policies for the selected algorithms (11) To benchmark the algorithms execution within the platform and propose optimisations.

This document aims to focus on point (1), which means the definition of a series of tests which will evaluate the neuGRID platform and infrastructure performance.

Thus, this document currently presents the conceived test and the results that were obtained on the grid middleware, meaning that its priority is to serve all protagonists of the Joint Research (JRA) and Services (SA) activities of the project, and in particular, IT researchers and IT developers involved in the following work packages:

Services Activities – SA

WP Id	*WP Title*	*WP10 Contribution*
WP7	Grid Services Provision	To dictate the deployment of necessary underlying grid services and corresponding configurations
WP8	Deployment Services Provision	To dictate the deployment of necessary underlying neuGRID services and corresponding configurations

Joint Research Activities – JRA

WP Id	*WP Title*	*WP Relation*
WP9	User and System Requirements Analysis	To conform with requirements analysis conclusions

1.3 Reference Documents

Prior to reading this document, the reader should be familiar with additional documents/ deliverables produced within the neuGRID project, which have or are considered to potentially have, an impact on the WP11 tasks. The following is a list of such documents sorted by information sources, activities and corresponding work packages (Note: list of available documents at the time of writing):

Services Activities Related Documents

WP Id	*WP Title*	*Documents*
WP7	Grid Services Provision	D7.1. Test-bed Installation and API Documentation
WP8	Deployment Services Provision	D8.2. Ground Truth and Phase 1 & 2 Deployment Test and Validation Report

Other Related Documents

Title	*Documents*
Project Documents	Project Description of Work

2 Definition of AC/DC1 Tests

2.1 Introduction

Testing the entire neuGRID gLite middleware stack in an automatic way reveals a set of problems that WP11 has been facing.

The first work was the determination of which services will have to be tested in the neuGRID project. The gLite middleware provides a wide range of services that may or may not be used inside the neuGRID infrastructure. The answer to this question was found in collaboration with the WP7 team which established the list of gLite services to deploy for the POC environment. This list contains the following gLite services:

- VOMS
- AMGA
- LFC
- CE
- SE (equal to DPM)
- BDII (site and top)
- WMS/LB

The second work done by WP11 was determining how to test each gLite service that was installed. Natively, the gLite middleware provides all the necessary APIs in C/C++ to interact with all the services. It also provides a few java/Python APIs for some services. Usually, gLite is used through what is called a "gLite User Interface" (gLite-UI): this is a suite of clients (binaries) that users and applications can use to access the gLite services. It was obvious then to build all our tests using this interface.

This also allows automating the procedure and generating a semi automatic set of tests, which will rely on the use of the gLite-UI. By successfully running the gLite-UI tests we can ensure that all the relying technology (gLite middleware) is behaving correctly.

EGEE provides a set of scripts that will perform basic tests over the desired infrastructure. After making a study of these scripts, a selection was then performed of the scripts that are suitable for our purposes. These have been summarized in the following section.

2.2 AC/DC1 Tests

The tests have been grouped into 5 areas based on the type of service that is to be tested. This allows for a service-oriented vision of the behaviour of the gLite middleware running in the neuGRID infrastructure. The functionality of the different components will be analyzed with scripts that will report on the correct or the incorrect behaviour of the component. At the end of the script, a measure of the performance will also be provided.

The five different test areas are:

- Security services
- Information system services
- Data management services
- Job management services

In the next sections all these areas will be presented with the list of tests that will be applied on each of them.

2.2.1 Security Related

The aim of this set of tests is to verify the correct operation of the security layer at grid authentication level. These tests are oriented to interact with the VOMS server in the GCC. VOMS serves as a central repository for user authorization information, providing support for sorting users into a general group hierarchy, keeping track of their roles, etc. Its functionality may be compared to that of a Kerberos KDC server.

- **UI-security-voms-proxy-info.sh**: Test voms-proxy-info with the following options.

gLite-UI commands executed
voms-proxy-info
voms-proxy-info -all
voms-proxy-info -text
voms-proxy-info -subject
voms-proxy-info -identity
voms-proxy-info -type
voms-proxy-info –timeleft
voms-proxy-info –strength
voms-proxy-info –path
voms-proxy-info -exists -bits 256
voms-proxy-info -exists -bits 512
voms-proxy-info -exists -bits 1024
voms-proxy-info -exists -valid 1:00
voms-proxy-info -exists -valid 3:00
voms-proxy-info -exists -valid 10:00
voms-proxy-info -exists -valid 24:00
voms-proxy-info –vo
voms-proxy-info –fqan
voms-proxy-info –acissuer

voms-proxy-info –actimeleft
voms-proxy-info –serial
voms-proxy-info -acexists $VO

- o
- o **UI-security-voms-proxy-init-info-destroy.sh**: Test the voms-proxy-init, voms-proxy-info and voms-proxy-destroy chain as follows:

gLite-UI commands executed
voms-proxy-init ${VO_OPTIONS} -verify -debug –limited -valid 1:00 -bits 1024 -out $TMPPROXY
voms-proxy-info -file $TMPPROXY
voms-proxy-destroy -file $TMPPROXY
voms-proxy-info -file $TMPPROXY
voms-proxy-destroy -file $TMPPROXY

- o **UI-security-voms-proxy-init-userconf.sh**: This test ensures that voms-proxy-init really uses the files given with the -userconf and -confile options.

gLite-UI commands executed
voms-proxy-init -voms testvoms -vomses $TMP_VOMS_FILE -out $TMPPROXY
voms-proxy-init -voms testvoms -userconf $TMP_VOMS_FILE -out $TMPPROXY
voms-proxy-init -debug -voms testvoms -confile $TMP_VOMS_FILE -out $TMPPROXY

2.2.2 Information System:

The aim of this set of tests is to verify the correct behaviour of the grid Information System. The Information System (IS) provides information about the status of Grid services and available resources. Job and data management services publish their status through the Grid Resource Information Server (GRIS). GRIS runs on every service node and is implemented using OpenLDAP, an open source implementation of the Lightweight Directory Access Protocol (LDAP). Every grid site also runs one Grid Index Information Server (GIIS). The GIIS queries the service GRISes on the site and acts as a cache storing information about all available site services. Finally, a top-level BDII collects all information coming from site BDIIs and stores them in a cache. The top-level BDII can be configured to collect published information from resources in all sites in a Grid (usually derived from the GOC DB), or just from a subset of them. The site list is normally filtered to include only sites which are currently operational, and VOs can also apply their own filters to exclude sites which are currently failing certain critical tests, so the sites visible in a BDII may fluctuate.

- o **UI-inf-lcg-info-ce.sh**: Run lcg-info with --list-ce.

gLite-UI commands executed
lcg-info $VO_OPTIONS --list-ce --attr "Tag"
lcg-info $VO_OPTIONS --list-ce --attr "OS,OSVersion,OSRelease,Processor,TotalCPUs,FreeCPUs,CEVOs"

- o **UI-inf-lcg-info-se.sh**: Run lcg-info with --list-se.

gLite-UI commands executed
lcg-info $VO_OPTIONS --list-se --attr "SEName,SEArch,SEVOs,Path,Accesspoint,Protocol,UsedSpace,AvailableSpace"

- **UI-inf-lcg-infosites.sh:** Runs lcg-infosites with various options and report failures if any.

gLite-UI commands executed
lcg-infosites --vo $VO sitename
lcg-infosites --vo $VO ce
lcg-infosites --vo $VO ce -v 2
lcg-infosites --vo $VO se
lcg-infosites --vo $VO closeSE
lcg-infosites --vo $VO tag
lcg-infosites --vo $VO lfc
lcg-infosites --vo $VO lfcLocal
lcg-infosites --vo $VO rb
lcg-infosites --vo $VO dli
lcg-infosites --vo $VO dliLocal
lcg-infosites --vo $VO vobox
lcg-infosites --vo $VO fts

- **UI-inf-ldapsearch.sh**: A set of ldapsearch requests with the following different attributes.

gLite-UI commands executed
ldapsearch -x -z $SIZE_LIMIT -H $GIIS -b "mds-vo-name=local, o=grid" 'objectclass=GlueCETop' \ GlueVOViewLocalID GlueCEStateRunningJobs GlueCEStateWaitingJobs GlueCEInfoDefaultSE
ldapsearch -x -H $GIIS -z $SIZE_LIMIT -b "mds-vo-name=local, o=grid" 'objectclass=GlueCESEBindGroup' \ GlueCESEBindGroupCEUniqueID GlueCESEBindGroupSEUniqueID
ldapsearch -x -H $GIIS -z $SIZE_LIMIT -b "mds-vo-name=local, o=grid" 'objectclass=GlueCESEBind' \ GlueCESEBindSEUniqueID GlueCESEBindCEAccesspoint GlueCESEBindCEUniqueID GlueCESEBindMountInfo
ldapsearch -x -H $GIIS -z $SIZE_LIMIT -b "mds-vo-name=local, o=grid" 'objectclass=GlueClusterTop' \ GlueClusterService GlueHostOperatingSystemName GlueHostOperatingSystemRelease GlueHostOperatingSystemVersion \ GlueHostProcessorModel GlueHostProcessorClockSpeed GlueHostProcessorVendor
ldapsearch -x -H $GIIS -z $SIZE_LIMIT -b "mds-vo-name=local, o=grid" \ 'objectclass=GlueSite' GlueSiteLocation GlueSiteWeb GlueSiteSysAdminContact

2.2.3 LCG File Catalog (LFC)

The aim of this set of tests is to verify the correct behaviour of the grid LCG File. The LFC (LCG File Catalog) is a secure catalog containing logical to physical file mappings. In the LFC, a given file is represented by a Grid Unique IDentifier (GUID). A given file replicated at different sites is then considered as the same file, thanks to this GUID, but (can) appear as a unique logical entry in the LFC catalog.

- **lfc-tests-common.sh**: Common functions for the UI LFC tests.
  - **UI-data-lfc-acl.sh**: Create a directory in LFC, list ACL, modify ACL, list ACL, delete directory.

gLite-UI commands executed
lfc-mkdir $TEST_DIR
lfc-getacl $TEST_DIR
lfc-setacl -m $NEWACL $TEST_DIR
lfc-getacl $TEST_DIR
lfc-rm -r $TEST_DIR

  - **UI-data-lfc-comment.sh**: Create a directory in the LFC, set its comment, list, delete comment, delete the directory.

gLite-UI commands executed
lfc-mkdir $TEST_DIR
lfc-setcomment $TEST_DIR "$COMMENT"
lfc-ls -d --comment $TEST_DIR
lfc-delcomment $TEST_DIR
lfc-ls -d --comment $TEST_DIR
lfc-rm -r $TEST_DIR

  - **UI-data-lfc-ln.sh**: Create a directory in the LFC, make a symbolic link to it and clean up.

gLite-UI commands executed
lfc-mkdir $TEST_DIR
lfc-ls -d -l $TEST_DIR
lfc-ln -s $TEST_DIR $LINK_NAME
lfc-ls -l $LINK_NAME
lfc-rm $LINK_NAME
lfc-rm -r $TEST_DIR

  - **UI-data-lfc-ls.sh**: Basic test of lfc-ls.

gLite-UI commands executed
lfc-ls -d $LFC_DIR
lfc-ls -d -l $LFC_DIR

lfc-ls $LFC_DIR
lfc-ls -l $LFC_DIR

- **UI-data-lfc-mkdir.sh**: Create a directory in LFC, list it and remove.

gLite-UI commands executed
lfc-mkdir $TEST_DIR
lfc-ls -d $TEST_DIR
lfc-ls -l -d $TEST_DIR
lfc-rm -r $TEST_DIR

2.2.4 LCG Data Management (SE)

The aim of this set of tests is to verify the correct behaviour of the gLite LCG SE / DPM. A Storage Element provides uniform access to data storage resources; its major functionality is to securely store data in the grid for its subsequent retrieval.

- **lcg-tests-common.sh**: Common functions for the UI LCG data management tests.
  - **UI-data-lcg-alias.sh**: A test of lcg data management tools: Upload a file to the GRID, list alias, create new alias, list again and remove.

gLite-UI commands executed
lcg-cr $VERBOSE $VO_OPTIONS -d $SE_HOST $LOCAL_FILE_URI 2>&1
lcg-la $VERBOSE $VO_OPTIONS $GUID 2>&1
lcg-aa $VERBOSE $VO_OPTIONS $GUID $ALIAS
lcg-la $VERBOSE $VO_OPTIONS $GUID
lcg-ra $VERBOSE $VO_OPTIONS $GUID $ALIAS
lcg-la $VERBOSE $VO_OPTIONS $GUID

  - **UI-data-lcg-cp.sh**: Upload, download and remove a GRID file using lcg data management tools.

gLite-UI commands executed
lcg-cr $VERBOSE $VO_OPTIONS -d $SE_HOST $LOCAL_FILE_URI
lcg-cp $VERBOSE $VO_OPTIONS $GUID file:$LOCAL_FILE_BACK

  - **UI-data-lcg-cr.sh**: Create and register, and then remove, a GRID file using lcg data management tools.

gLite-UI commands executed
lcg-cr $VERBOSE $VO_OPTIONS -d $SE_HOST $LOCAL_FILE_URI

  - **UI-data-lcg-list.sh**: Upload a file to the GRID, list replica, list GUID for the replica, get TURL, and delete the file using lcg data management tools.

gLite-UI commands executed

lcg-cr $VERBOSE $VO_OPTIONS -d $SE_HOST $LOCAL_FILE_URI
lcg-lr $VERBOSE $VO_OPTIONS $GUID
lcg-lg $VERBOSE $VO_OPTIONS $SURL
lcg-gt $VERBOSE $SURL gsiftp

2.2.5 Job Manager

The aim of this set of tests is to verify the correct behaviour of the gLite Job Manager. The three major components that constitute the Job Management Services group are the Computing Element, Workload Management and Accounting. Thus, the Job Manager is the interface used to submit Jobs to the grid.

- **UI-workload-glite-wms-deleg-submit-wait-output.sh**: "Delegate proxy - submit - get status - get output" test for gLite WMS workload system

gLite-UI commands executed
glite-wms-job-delegate-proxy -d $$
glite-wms-job-submit -d $$
glite-wms-job-status
glite-wms-job-cancel --noint
glite-wms-job-output

- **UI-workload-glite-wms-job-list-match.sh**: A job-list-match test for the gLite-WMS submission system.

gLite-UI commands executed
glite-wms-job-list-match -a --rank $JDLFILE

- **UI-workload-glite-wms-submit-wait-output.sh**: The submit - status - get output test for gLite WMS workload system

gLite-UI commands executed
glite-wms-job-submit -a
glite-wms-job-status
glite-wms-job-cancel --noint
glite-wms-job-outp

2.3 Conclusion

This first set of tests intended to test the gLite GRID middleware that we use for the neuGRID infrastructure at a really low level. The different grid services were grouped in five areas and specific tests were applied to each of them.

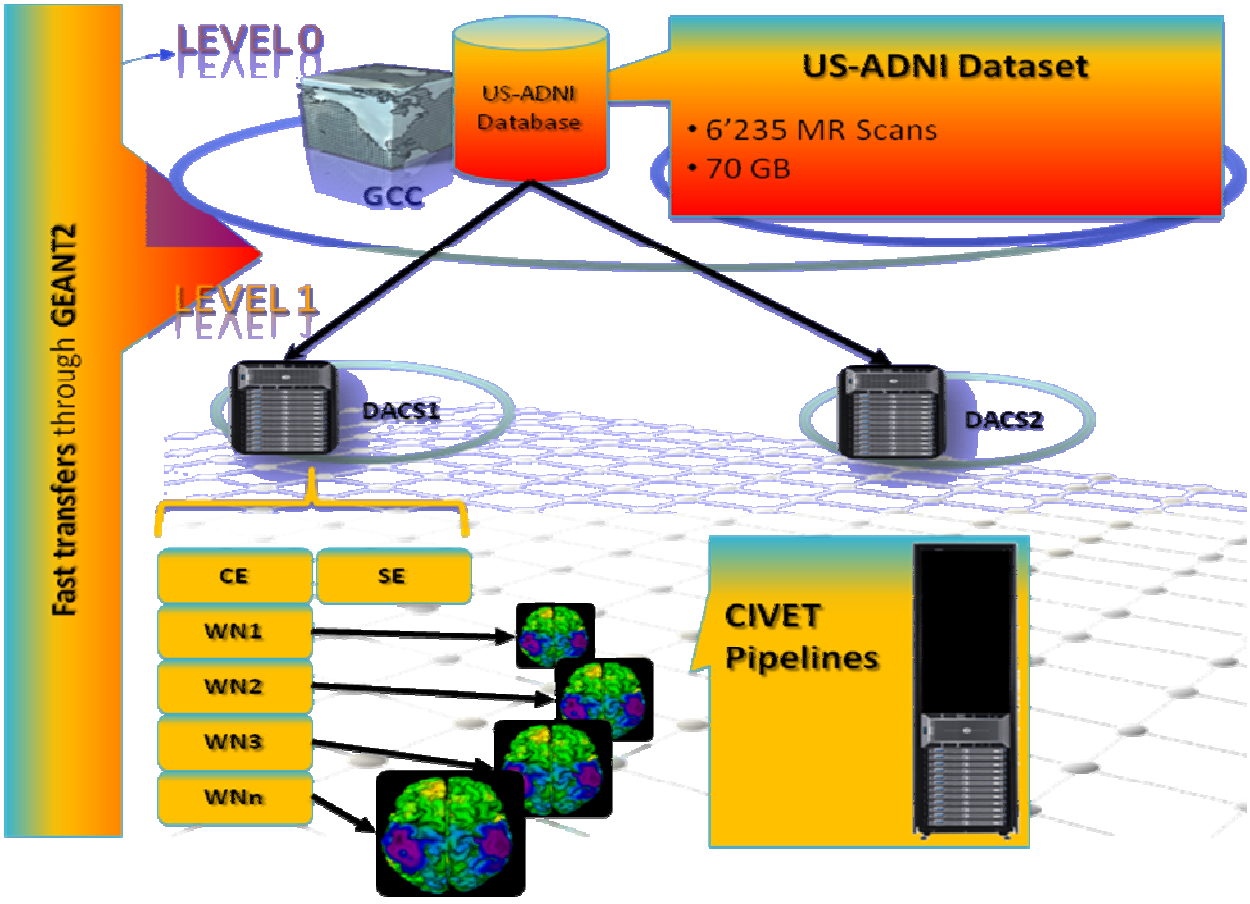
All the results were satisfactory which means that the gLite middleware is behaving properly with no blocking bugs that could interfere with the neuGRID developments. In the next section, more sophisticated and resource consuming test are presented.

3 Definition of AC/DC2 Tests

3.1 Introduction

This AC/DC2 test consists in a data challenge that was performed on the US-ADNI data (715 patient folders, containing in total 6'235 scans - i.e. baseline + 5 to 10 follow-ups per patient- in MINC format, representing roughly 108 GB of data). Each scan is about 10 to 20 MB and can contain from 150 to 250 slices.

The data challenge consisted in analysing this entire dataset - as is, i.e. no data filtering and brute force approach - using the CIVET pipeline. To do so, two out of the three DACS (i.e. Fatebenefratelli (FBF) and Karolinska Institute (KI)) sites have been fully deployed and equipped with 64-bit Worker Nodes and the 64-bit version of the CIVET pipeline has been gridified and propagated within the Production environment. The 2 DACS provide 184 processing cores, 5.3TB of storage capacity and are connected to the GEANT2 network, thus guarantying a good network bandwidth. The following image explains graphically the data challenge configuration.



3-1: Data Challenge configuration

The CIVET pipeline was executed with no optimization in order to maximize the number of parallel job submissions and executions. Multiple CIVET instances will therefore be spawned in the DACS' Worker Nodes. From our tests, CIVET-64 takes about 7 hours to process on a single scan and

generates 10 times the initial data volume as output. Therefore, to analyze the US-ADNI dataset approximately 2 weeks were necessary and almost 1 TB of output was generated.

3.2 AC/DC2 Tests

3.2.1 The data challenge resumed in some numbers

As you can see in the following table the test that was done was quite heavy. The duration was nearly two weeks and the number of analyzed voxels[1] during this period was huge (nearly 9,5 billion).

Experiment duration on the Grid		**< 2 Weeks**
Experiment duration on single computer		**> 5 Years**
Analyzed data	**Patients**	715
	MR Scans	6'235
	Images	~1'300'000
	Voxels	~9'352'500'000
Total mining operations		~286'810
Mining operations throughput per hour		~1'200
Voxels operations throughput per hour		~38'970'000
Max # of processing cores in parallel		184
Number of countries involved		3 (the 2 DACS + 1 central site in France)
Volume of output data produced		1 TB

3-2: The data challenge resumed in some numbers

What is really interesting in those numbers is also the benefit of the GRID in this kind of challenge that can easily be observed. Indeed, if we would have launched the same experiment on a single computer, it would have taken nearly 5 years to complete (of course we speak about a single CPU, single core and this number highly depends on the CPU power).

3.2.2 The data challenge concretely launched on the GRID

Concretely, the data challenge consists in a job that is launched into the gLite GRID middleware. This job is a so called "parametric" job. This kind of job is useful when you want to run similar jobs that only differ in arguments or input/output files. Parametric job type allows you to submit bulk of

[1] A **voxel** (*volumetric pixel*) is a volume element, representing a value on a regular grid in three dimensional space. This is analogous to a pixel, which represents 2D image data in a bitmap (which is sometimes referred to as a pixmap).

jobs as a single job, and then WMS takes over, break your parametric job into many single jobs and submit them separately to CEs on your behalf, thus significantly reducing exectution time. Upon submission, every sub-job will be associated with an individual identifier (job ID), and beside that a common job ID will be assigned to the whole set of jobs. This common id is used to list status or retrieve output of all jobs at once[2].

The job that was launched was the following (truncated - JDL syntax):

```
[
JobType          = "Parametric";
Executable       = "/bin/civet-launch.sh";
Arguments        = "ng-maat-server4.maat-g.com /grid/neugrid/data/US-ADNI/_PARAM_ /grid/neugrid/share/US-ADNI-CIVET ADNI";
StdOutput        = "civet.out";
StdError         = "civet.out";
Requirements     = Member("VO-neugrid-civet", other.GlueHostApplicationSoftwareRunTimeEnvironment);
OutputSandbox    = {"civet.out"};
RetryCount       = 1;
ShallowRetryCount = 10;
Parameters       = {
"ADNI_002_S_0295_MR_MP-RAGE_REPEAT_br_raw_1_S13407_I13721.mnc.gz",
"ADNI_002_S_0295_MR_MP-RAGE_REPEAT_br_raw_1_S21855_I28560.mnc.gz",
"ADNI_002_S_0295_MR_MP-RAGE_REPEAT_br_raw_1_S32679_I55276.mnc.gz",
"ADNI_002_S_0295_MR_MP-RAGE_REPEAT_br_raw_1_S54060_I114209.mnc.gz",
"ADNI_002_S_0295_MR_MP-RAGE_br_raw_1_S13408_I13722.mnc.gz",
"ADNI_002_S_0295_MR_MP-RAGE_br_raw_1_S21856_I28561.mnc.gz",
"ADNI_002_S_0295_MR_MP-RAGE_br_raw_1_S32678_I55275.mnc.gz",
"ADNI_002_S_0295_MR_MP-RAGE_br_raw_1_S54061_I114210.mnc.gz",
"ADNI_002_S_0413_MR_MP-RAGE_REPEAT_br_raw_1_S13894_I14438.mnc.gz",
"ADNI_002_S_0413_MR_MP-RAGE_REPEAT_br_raw_1_S22558_I29706.mnc.gz",
...
...

"ADNI_941_S_1311_MR_MPRAGE_br_raw_98_S56645_I118290.mnc.gz",
"ADNI_941_S_1311_MR_MPRAGE_br_raw_98_S65346_I140246.mnc.gz",
"ADNI_941_S_1363_MR_MPRAGE_Repeat_br_raw_138_S28009_I44496.mnc.gz",
"ADNI_941_S_1363_MR_MPRAGE_br_raw_98_S28008_I44495.mnc.gz"
};
]
```

The executable that is launched by this job is named "civet-launch.sh". This is an helper script that helps the users to launch the CIVET pipeline inside the infrastructure. This script requires four arguments:

- The LFC service hostname (ng-maat-server4.maat-g.com)
- The path to the MINC file that has to be analyzed (/grid/neugrid/data/US-ADNI/_PARAM_)
- The directory where will be stored the result (/grid/neugrid/share/US-ADNI-CIVET)
- The "prefix" that will be used by CIVET (ADNI)

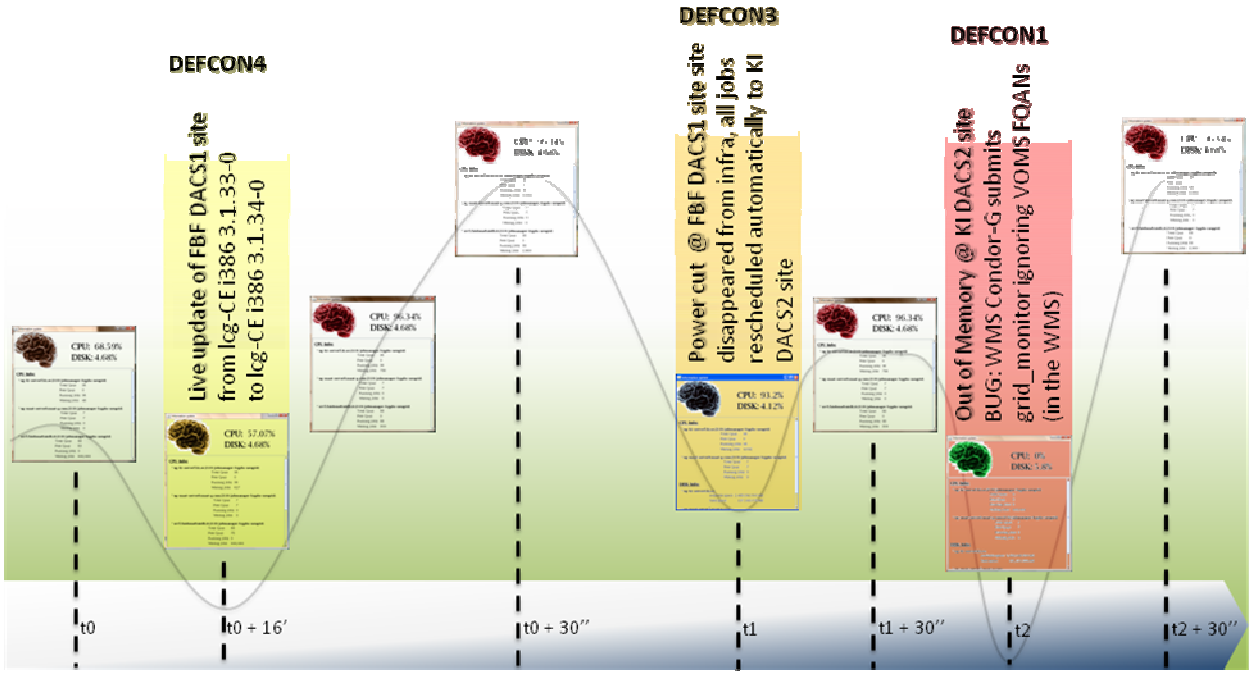
The full "civet-launch.sh" script can be found in annexe 1. The main functions of this script are:

- Initialize the environment variables to be able to use CIVET and the GRID middleware.
- Test the existence of the input data and of the output directory.
- Test if the output data already exists or not. If it already exists the job is stopped.
- Create a "lock" in the output directory to be sure that the job will not be launched more than 1 time.
- Retrieve the input data from the GRID if needed
- Format the input data properly so that it can be analysed by CIVET
- Launch the CIVET pipeline.
- Analyse the CIVET logs to see if everything goes well or not. If not, the script will report the problem in the output file of the job.

[2] For more information about parametric jobs, please refer to : http://wiki.egee-see.org/index.php/Parametric_Jobs

3.2.3 The data challenge lifetime

The first hours of the data challenge were difficult. In the following image, a timeline of the first 24 hours of the data challenge is resumed.



3-3: The first hours of the data challenge

What can be easily seen in this previous image is that some problems appeared during the first hours of the data challenge.
The data challenge was launched Friday 28th of August 2009 at 3pm CET (t0 in the image) with a basic glite-UI command line:

```
glite-wms-job-submit –output civet_us-adni_exec.id –a civet_us-adni_exec.jdl

Connecting to the service https://ng-maat-server10.maat-g.com:7443/glite_wms_wmproxy_server

====================== glite-wms-job-submit Success ======================

The job has been successfully submitted to the WMProxy
Your job identifier is:

https://ng-maat-server10.maat-g.com:9000/LXgmGJfb6ak7Ef2z9ElSiw

The job identifier has been saved in the following file:
/home/jerome/civet_us-adni_exec.id

==========================================================
```

During the first 15 minutes, WMS was generating the 6235 individual jobs from the parametric job that was launched and to schedule everything on the 2 DACs. At this moment all the team was monitoring with a lot of attention everything and this is why a small problem was discovered at FBF. Indeed the job were scheduled to the both DACS properly by WMS but the FBF site BDII service was not reporting properly this information. After a really quick investigation it was figured out that the problem was the version of the site BDII service that was installed at FBF that had a bug. Thus, it was decided to update this service. The update went very well even with the ongoing data challenge and after a few minutes, all the needed information was displayed properly:

```
- CE: ng-ki-server5.ki.se:2119/jobmanager-lcgpbs-neugrid
  - FreeCPUs          0
  - TotalCPUs         96
  - RunningJobs        96
  - WaitingJobs       2579
- CE: srv5.fatebenefratelli.it:2119/jobmanager-lcgpbs-neugrid
  - FreeCPUs          0
  - TotalCPUs         88
  - RunningJobs        88
  - WaitingJobs       2670
```

As one can see in the previous output, all the jobs were not submitted to the infrastructure. This was done after more or less 30 minutes.

Everything was running as expected during more or less 12 hours but at some point during the week-end, the FBF site disappeared from the GRID information system (t1 in the image). After some investigations, it appeared that none of the servers of the site was accessible anymore. At this time, everybody was quite stressed because more than 3000 jobs were already scheduled to this node. Of course, it occurs during the week-end and it was impossible to contact the FBF technicians to find the problem. At this point we discovered that the WMS service did a really good job. Indeed, after some time, WMS saw that the FBF site was done and decided to reschedule the FBF jobs to the KI site. This was a really good reaction from the GRID (t1+2h).

Nevertheless, this good WMS behaviour induced a new problem due to our lack of experience in this kind of data challenge. Indeed, when WMS re-scheduled the FBF jobs on the KI node, this later was completely overloaded by all the jobs that were in the CE queue. We managed to maintain the CE up and running for some time but at some point. We decided to restart the server and to limit the maximum queue length to 3000 jobs (t2). After the reboot, everything went better thanks to this limitation.

After the week end, the FBF technicians explained that there was a power cut in the server room because one of the power switches (16Amp only) didn't guarantee the correct server feeding. It has been modified with a 25 Amp and also the line-power source of the server has been changed.

After those problems everything went well for the rest of the data challenge.

3.3 Conclusion

This new AC/DC2 was created to test the neuGRID grid resources but, this time, compared to AC/DC1, this test aimed to be more computing and data intensive in order to test the production sites. It was decided to create a data challenge on the US-ADNI data. The challenge consisted of the analysis of each scan of the US-ADNI data using the CIVET pipeline.

This test allowed us to see the robustness of the gLite grid infrastructure that is currently used in the neuGRID project. Indeed, despite some difficulties in the first hours of the data challenge, all the jobs completed properly (from a GRID point of view). These difficulties were in fact very positive and allow us to improve the configuration of the DACS sites.

4 Conclusion

The design and execution of tests over large infrastructures (such as neuGRID) is a central task.

The test must ensure the correct operation of the technology upon which the project relies. In the neuGRID project, this means that we must ensure that there will not be problems related to gLite middleware operation nor configuration, this is the role of "AC/DC" tests.

With an in-depth analysis, WP11 has found some key points in which the developed tests must be focused:

- **How to test**: there are two approaches, the first being the "per service", in which all the services are tested in an independent way. The second option is to test from an upper layer, in which high level operations are launched to the grid, to ensure the correct operation of these non-atomic processes. In WP11, the second approach has been selected. The reason for this is that WP11 must test the infrastructure under the point of view of the grid user, which will work with the grid through interfaces, and not directly over the services.
- **Reporting**: Tests should detect malfunctions of the infrastructure, and generate reports to give the correct feedback to WP8. These reports must be clear enough to give key clues to WP8 in order to determine and solve the malfunction of the system.
- **Modularity**: Tests must be developed in a modular way, able to be run in an automatic way, clearly determining the malfunctioning parts, and allowing the reproduction of the encountered errors.
- **Easy to use**: Tests must present an easy interface with few parameters covering all the possibilities being offered to the programmers.
- **Generality**: Tests must be as general as possible, allowing for changes in the infrastructure, should the necessity arise.
- **Performance**: Tests must give a generic performance measure of the basic grid operations, to rapidly detect some problems related to the throughput.

Indeed, a second type of tests called "Story Lines" will be the second challenge for WP11. In this point, a deep mining of the User Requirements specification, which will be released on month 24 too, will give WP8 the guides to develop tests to validate the neuGRID services from the users' point of view.

5 List of Abbreviations

BDII	Berkeley Database Information Index
CE	Computing Element
DPM	Disk Pool Manager
EGEE	Enabling Grids for E-sciencE
gLite	EGEE Grid middleware stack
IS	Information System (grid-level)
LSF	Local Sharing Facility
LB	Logging and Bookkeeping
neuGRID platform	neuGRID services + gLite middleware
POC	neuGRID Proof Of Concept sub-infrastructure – *neuGRID test-bed*
PROD	neuGRID Production sub-infrastructure
SE	Storage Element
UI	User Interface
VOMS	Virtual Organization Membership Service
WN	Worker Node
WMS	Workload Management System

6 ANNEX 1: The civet-launch.sh script

```
#!/bin/bash
# CIVET
# neuGRID 2009
# Author: Jerome Revillard
#
# v1.0: Initial version.
# v1.1:
#       -Add the possibility to specify where to store the result.
#       -Add a lot of checks
# v1.2: 26/06/2009
#       -Launch mincreshape on the mnc file before CIVET execution to be sure to have a well
formatted mnc file
#

set -x

echo "Script launched with the following parameters:"
echo "  ->  $@"
echo "On:"
echo "  ->  `hostname -f`"


export CIVET_PATH=/opt/Quarantines/200906/
SOURCE_DATA=input/
OUTPUT_DATA=output/
CURRENT_DIR=`pwd`

mkdir -p $SOURCE_DATA $OUTPUT_DATA

############################################################
# functions
############################################################

#this function remove the lock before existing
function quit {
        lfc-rm -r $LFC_OUTPUT_DIR/.$CIVET_RESULT
        exit $1
}

############################################################
# Make the different tests before launching the pipeline
############################################################

# Number of parameters
if [ $# -ne 4 ]; then
         echo "Wrong number of parameters:"
        echo "      1. LFC host,"
        echo "      2. LFN of the input data (/grid/neugrid/..../xxxxx.mnc.gz file),"
        echo "      3. LFN directory where to put the ouput data (/grid/neugrid/..../ directory),"
        echo "      4. civet prefix to use."
        echo "Aborting."
         exit 1
fi


# Assign variables.
LFC_HOST=$1
INPUT_DATA=$2
#Remove the slash at the end of the directory name if exists.
case "$3" in
        */)     LFC_OUTPUT_DIR=${3%/} ;;
        *)      LFC_OUTPUT_DIR=$3 ;;
esac
CIVET_PREFIX=$4
FILENAME_INPUT=${INPUT_DATA##*/}
case $FILENAME_INPUT in
        *_t1.mnc.gz | *_t2.mnc.gz | *_pd.mnc.gz)
                        CIVET_PATIENTID=`TMP_ID=${FILENAME_INPUT##${CIVET_PREFIX}_} && echo
${TMP_ID%_*}`
                        CIVET_PRE_OPERATION=0
                        ;;
        *.mnc.gz)
                        CIVET_PATIENTID=`TMP_ID=${FILENAME_INPUT##${CIVET_PREFIX}_} && echo
${TMP_ID%.mnc.gz}`
```

```
                        CIVET_PRE_OPERATION=1
                        ;;
        *_t1.mnc | *_t2.mnc | *_pd.mnc)
                        CIVET_PATIENTID=`TMP_ID=${FILENAME_INPUT##${CIVET_PREFIX}_} && echo
${TMP_ID%_*}`
                        CIVET_PRE_OPERATION=2
                        ;;
        *.mnc)
                        CIVET_PATIENTID=`TMP_ID=${FILENAME_INPUT##${CIVET_PREFIX}_} && echo
${TMP_ID%.mnc}`
                        CIVET_PRE_OPERATION=3
                        ;;
        *)
                        echo "Supported files extentions are: *_t1.mnc.gz | *_t2.mnc.gz | *_pd.mnc.gz
| *.mnc.gz | *.mnc"
                        exit 10
                        ;;

esac
CIVET_RESULT=civet_output_${FILENAME_INPUT%%.*}.tgz

# LFC connection
export LFC_HOST=$LFC_HOST
lfc-ping &>/dev/null
if [ $? != 0 ]; then
        echo "Unable to communicate with the LFC"
        exit 20
fi

# Output directory existence
exec_result=`lfc-ls -ld $LFC_OUTPUT_DIR`
if [ $? == 0 ]; then
         case $exec_result in
                 d*)     ;;

                 *)      echo "$LFC_OUTPUT_DIR is not a directory"
                        exit 30
                        ;;
         esac
else
         echo "$LFC_OUTPUT_DIR does not exist or you cannot access it."
        exit 40
fi

# Create a lock so that this user does not launch 2 times the same job on the same data with the
same parameters...
# (allow also to verify that we can write into $LFC_OUTPUT_DIR)
exec_result=`lfc-mkdir $LFC_OUTPUT_DIR/.$CIVET_RESULT 2>&1`
if [ $? != 0 ]; then
        exec_result=`echo "$exec_result"|grep 'File exists'`
        if [ "$exec_result" != "" ]; then
                echo "You probably already launched this pipeline. The lock file already exists"
                echo "If you are sure that no other pipeline is currently executed with the same
parameters,"
                echo "then delete the following directory: $LFC_OUTPUT_DIR/.$CIVET_RESULT"
                exit 50
        else
                echo "Unable to create the lock directory: $LFC_OUTPUT_DIR/.$CIVET_RESULT"
                echo "Verify that you have write access to the $LFC_OUTPUT_DIR directory"
                exit 60
        fi
fi

# Trap the TERM, SIGINT and SIGTERM signals to properly purge the lock files before leaving the
script
trap "quit 9999" SIGTERM
trap "quit 9999" TERM
trap "quit 9999" SIGINT

# Verify if the output file already exists
exec_result=`lfc-ls $LFC_OUTPUT_DIR/$CIVET_RESULT 2>&1`
if [ $? != 0 ]; then
        exec_result=`echo "$exec_result"|grep 'No such file or directory'`
        if [ "$exec_result" == "" ]; then
                echo "Unable to verify if the output file $LFC_OUTPUT_DIR/$CIVET_RESULT already
exists."
                echo "The process will continue but will fail if it's the case."
        fi
```

```
else
       echo "You probably already launched this pipeline. The output file already exists"
       echo "If you want to launch it again, move or remove the $LFC_OUTPUT_DIR/$CIVET_RESULT file."
       quit 70
fi

############################################################
# Retrieve data or use already existing one
############################################################
if [ -f $INPUT_DATA ]; then
       echo "Processing already available data: $INPUT_DATA"
       mv $INPUT_DATA $SOURCE_DATA/$FILENAME_INPUT || quit 80
else
       cd $SOURCE_DATA
       echo "Retrieving input data: $INPUT_DATA"
       lcg-cp -v -D srmv2 lfn:$INPUT_DATA $FILENAME_INPUT || quit 81
       cd -
fi

############################################################
# Sourcing the CIVET environment
############################################################
source $CIVET_PATH/init.sh

############################################################
# Preparation of the data if needed
############################################################
if [ $CIVET_PRE_OPERATION == 0 ]; then
       # *_t1.mnc.gz | *_t2.mnc.gz | *_pd.mnc.gz
       cd $SOURCE_DATA
       gzip -df $FILENAME_INPUT
       mv ${FILENAME_INPUT%%.*}.mnc ${FILENAME_INPUT%%.*}.mnc.tmp
       mincreshape -transverse +direction  ${FILENAME_INPUT%%.*}.mnc.tmp ${FILENAME_INPUT%%.*}.mnc
|| quit 82
       rm -f ${FILENAME_INPUT%%.*}.mnc
       cd -
elif [ $CIVET_PRE_OPERATION == 1 ]; then
       # .mnc.gz files
       cd $SOURCE_DATA
       gzip -df $FILENAME_INPUT
       mincreshape -transverse +direction  ${FILENAME_INPUT%%.*}.mnc ${FILENAME_INPUT%%.*}_t1.mnc ||
quit 83
       rm -f ${FILENAME_INPUT%%.*}.mnc
       cd -
elif [ $CIVET_PRE_OPERATION == 2 ]; then
       # *_t1.mnc | *_t2.mnc | *_pd.mnc files
       cd $SOURCE_DATA
       mv $FILENAME_INPUT $FILENAME_INPUT.tmp
       mincreshape -transverse +direction $FILENAME_INPUT.tmp $FILENAME_INPUT || quit 84
       rm -f $FILENAME_INPUT.tmp
       cd -
elif [ $CIVET_PRE_OPERATION == 3 ]; then
        # .mnc files
        cd $SOURCE_DATA
       mincreshape -transverse +direction $FILENAME_INPUT ${FILENAME_INPUT%%.*}_t1.mnc || quit 85
        rm $FILENAME_INPUT
        cd -
fi

############################################################
# Pipeline Execution
############################################################
CIVET_CMD="$CIVET_PATH/CIVET/CIVET_Processing_Pipeline -sourcedir $CURRENT_DIR/$SOURCE_DATA -
targetdir $CURRENT_DIR/$OUTPUT_DATA -prefix $CIVET_PREFIX $CIVET_PATIENTID -lsq12 -spawn -granular -
run"

echo "Launching the Civet Pipeline: $CIVET_CMD"
$CIVET_CMD || quit 90

# Test if CIVET was executed properly.
FAILED_OP=`ls $CURRENT_DIR/$OUTPUT_DATA/$CIVET_PATIENTID/logs/ |grep .failed`
ERROR=0
if [ "$FAILED_OP" != "" ]; then
       for failed_op in "$FAILED_OP"; do
              case $failed_op in
                     *.verify_image.failed)
                            ;;
                     *)
```

```
                                ERROR=1;
                                echo
"=========================================================================================="
                                echo "Content of ${failed_op%%.failed}.log"
                                echo
"=========================================================================================="
                                cat
$CURRENT_DIR/$OUTPUT_DATA/$CIVET_PATIENTID/logs/${failed_op%%.failed}.log
                                echo
"=========================================================================================="
                                 ;;
                esac
        done
fi
if [ $ERROR == 1 ];then
        echo "Errors were found in the pipeline execution!"
        quit 100
fi

echo "Civet output available inside $OUTPUT_DATA"

echo "Compressing the output..."
cd $OUTPUT_DATA
tar -h -c *|gzip>$CIVET_RESULT

############################################################
# Pipeline result upload
############################################################
x=1
UPLOAD_DONE=0
while [ $x -le 5 ]; do
        RESULT=`lcg-cr -v -l lfn:$LFC_OUTPUT_DIR/$CIVET_RESULT $CIVET_RESULT 2>&1`
        if [ $? != 0 ]; then
                sleep 30
                x=$(( $x + 1 ))
        else
                echo "Upload error:"
                echo $RESULT
                x=6
                UPLOAD_DONE=1
        fi
done

if [ $UPLOAD_DONE == 0 ]; then
        echo "UNABLE TO UPLOAD THE RESULT!"
        quit 110
fi
cd -

############################################################
# END
############################################################
echo "======================================== Finished ========================================="
echo " You can download the output of the algorithm using:"
echo " export LFC_HOST=$LFC_HOST && \\"
echo " lcg-cp lfn:$LFC_OUTPUT_DIR/$CIVET_RESULT ./$CIVET_RESULT"
echo "==========================================================================================="
quit 0
```