

## The MedIGrid PSE in an LCG/gLite environment\*

A. Murli

University of Naples Federico II  
almerico.murli@dma.unina.it

L. Carracciuolo

Italian National Research Council  
luisa.carracciuolo@cnr.it

G. Laccetti

University of Naples Federico II  
giuliano.laccetti@dma.unina.it

V. Boccia

University of Naples Federico II  
vania.boccia@unina.it

L. D'Amore

University of Naples Federico II  
luisa.damore@dma.unina.it

M. Lapegna

University of Naples Federico II  
marco.lapegna@dma.unina.it

### Abstract

*In this paper we are concerned with improvements and enhancements of a medical imaging grid-enabled infrastructure, named MedIGrid, oriented to the transparent use of resource-intensive applications for managing, processing and visualizing biomedical images. We describe an implementation of the MedIGrid PSE in an LCG/gLite environment. We'll mainly focus on how to exploit the features of the new middleware environment to improve the efficiency and the services reliability of the PSE; further, some comments will be devoted to how to modify, extend and/or improve the underlying numerical components.*

### 1. Introduction

Since 2007, the Italian Ministry of University and Research has promoted the creation of four distributed e-infrastructures, in the South of Italy in the context of CRESCO [1], CyberSar [2], PI2S2 [8] and S.Co.P.E. [9] national projects to support the scientific community involved in the main strategic fields of e-Science. The result is a new unified technical/scientific grid environment, ready to join European Grid Infrastructure (EGI [4]) and to include other partners like SPACI Consortium [10].

The activity, described in this paper, is partially supported by PON S.Co.P.E. project (Italian acronym for high Performance, Cooperative and distributed System for scientific Elaboration). This is a research project with two

aims: the development of applications in fundamental research fields and the implementation of an open and multidisciplinary Grid infrastructure, involving several departments of the University of Naples Federico II distributed in metropolitan scale in the city of Naples.

The S.Co.P.E. architecture provides the integration of all computational and storage resources available in the metropolitan sites involved into the project, using:

- LCG/gLite ([19]) as "low level" middleware and
- the *SCOPE-toolkit* [14] as "high level" middleware.

In this context we were involved in improving and enhancing a medical imaging grid-enabled PSE, named MedIGrid, oriented to the transparent use of resource-intensive applications for managing, processing and visualizing biomedical images (see [20, 15]). Then, the reason of the choice of the LCG/gLite middleware as reference environment is twofold: it provides some general purpose and high level services (Data and Metadata Catalogue, Resource Broker, ...) and it is the de-facto choice for the most of the Italian (S.Co.P.E., PI2S2, CYBERSAR, CRESCO, SPACI, IGI [5], ...) and European (EGEE [3], EGI, ...) grid projects. Moreover, all the work we spent to implement *grid-aware* applications in different environments helps us to propose new solutions for interoperativity/interoperability.

This paper is organized as follows: in Sec. 2 we report the GRID middleware infrastructure, in Sec. 3 we describe the MedIGrid implementation in this environment, focusing on some planning strategies aimed to obtain more efficiency in data management and job submission; in the same section two MedIGrid case study numerical kernels are described, and some details are reported about recent modifications we

\*This work has been partially supported by the 2000-2006 S.Co.P.E. PON Italian National Project

have done to introduce a checkpointing system at application level. Finally in Sec. 4 we report conclusions and some suggestions on possible new middleware enrichment and/or integration.

## 2. The GRID infrastructure

As cited before, the testbed we used is mainly the SCoPE infrastructure, essentially based on the integration of the LCG/gLite and SCOPE-toolkit middlewares.

From the hardware point of view the S.Co.P.E. infrastructure will be constituted, in few months, by up to two thousands computational cores and about 100 TB of disk space. Actually our work has been developed on a prototypical version of SCoPE infrastructure, involving about two hundred of computational cores and one TB of disk space. These resources are distributed among some Departments of the University of Naples Federico II and are connected by a Metropolitan Area Network implemented by a 1 Gbps fibre channel technology.

The S.Co.P.E. GRID is already connected to the GRIDs owned by the projects cited in the previous paragraph (CRESCO, P2S2, CYBERSAR).

The middleware LCG/gLite, developed in the context of EGEE (Enabling Grid for E-Science) European project, provides the user with high level services for scheduling and running computational jobs, accessing and moving data, and obtaining information on the Grid infrastructure, all embedded into a consistent security framework [16]:

### **VOMS (Virtual Organization Management System)**

**Service** is a system to classify users that are part of a Virtual Organization (VO) on the basis of a set of attributes granted upon request and to include that information inside Globus-compatible proxy certificates.

### **WMS (Workload Management System) Service**

that comprises a set of Grid middleware components responsible for the distribution and management of jobs across Grid resources, in such a way that applications are efficiently executed. The core component of the Workload Management System is the Workload Manager (WM), whose purpose is to accept and satisfy requests for job management coming from its clients.

### **LFC (Logical File Catalogue) Service**

that stores the location(s) of files and replicas. LFC will map LFN (Logical File Name) or GUID (Global Unique Identifier) to SURL (Site URL) which specifies a physical instance of the file. It is a high performance file catalogue that builds on the experiences gathered from the EGEE user communities.

The *SCOPE-toolkit* collects some consolidated libraries (i.e. BLAS, LAPACK e ScaLAPACK for linear algebra

and PETSc and FFT for scientific computation) and applications (i.e. GROMACS molecular dynamics) in a self contained package that automizes (by means of standard installation/validation procedures) the dissemination of needed software on the computing resources and it makes applications easily integrated into the S.Co.P.E. GRID infrastructure.

In the last years, much attention has been spent on providing different kinds of “low level” middleware (like Globus, UNICORE, LCG/gLite,...) and on developing related applications. However at each middleware change, applications developers have to modify their work due to the existing gap between middleware and applications. We think that even if, at present, the *SCOPE-toolkit* doesn't contain any grid service or tool, in the future, it could be extended and/or improved to fill that gap.

## 3. The new PSE design

MedIGrid is oriented to the transparent use of resource-intensive applications for managing, processing and visualizing biomedical images. The applications used by MedI-Grid are denoising of 3D echocardiographic images sequence and segmentation of 2D echocardiographic images. Details can be found in [17, 18]. Both applications are based on the *Portable, Extensible Toolkit for Scientific Computation (PETSc)* library [11, 12, 13], actually included in our reference grid environment by means of the S.Co.P.E Toolkit.

In Fig. 1 is shown the web page to access the PSE: after a simple authentication phase, the user has the chance to:

- select the operation (denoising, segmentation, etc.) to be performed on input images and configure some related parameters;
- use some tools, based on standard library as ImageJ [6], to visualize and process input and output image data.

Due to a change in the underlying GRID environment (see Fig. 2), our work on MedIGrid PSE involved:

- A server side modification of the GRID Portal by means of calls to functions of LCG/gLite Java library to interface the PSE with some new collective services:
  - LFC, for a more efficient data management,
  - WMS, for a more complete and general purpose resources discovery and selection,
  - VOMS, for the management of VO access policies to the resources and
  - MyProxy to automatic credential renew,



Figure 1. MediGrid Portal

- The integration of some middleware features and additional code to implement a such kind of job monitoring,
- The introduction of an embryonal checkpointing system implemented at application level.

While we had to modify the “hidden” part of the PSE, we left unmodified the GUI, already complete of a satisfying number of consolidated tools for image processing parameter configuration, image visualization and post-processing.

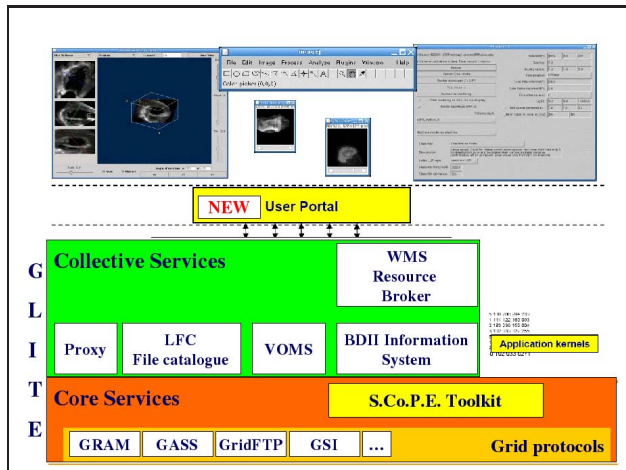


Figure 2. New PSE architecture

Here our attention is focused on some planning strategies aimed to obtain more efficiency in data management and job submission.

In particular our choices allowed us to avoid bottlenecks, due to multiple and unuseful Resource Broker (WMS) transits, and limitation imposed from WMS to data sizes.

To give an idea of problem size we can observe that an input sequence of only 14 ecographic small images is large more than 20MB and the output of the process is  $20MB \times nscales$  (where  $nscales$  is an input parameter related to the output quality). As the images dimension and the number of problems to be solved increase, we have to avoid unuseful paths in data transferring and redundance storage on WMS, to provide an efficient data management.

In MediGrid job execution schema, application binaries and data are stored on Storage Resources and registered on a Logical File Catalogue with symbolic names. On the UI are present only text files containing image processing parameters.

At execution time, parameter files reach computational resources passing through the Resource Broker (WMS), while binaries and input data are transferred directly from Storage to Computational Resource, reducing the execution bottleneck. To reach this aim, we had to modify our numerical kernels, by the means of calls to LFC/GFAL library functions: if LFC functions let us to access data transparently, by means of their logical names, GFAL tools allow efficient data transfer by multistream gridFTP low level protocol.

At the end of the execution output log file reaches the UI through the WMS, while the output image is directly moved on the Storage and registered on the File Catalogue.

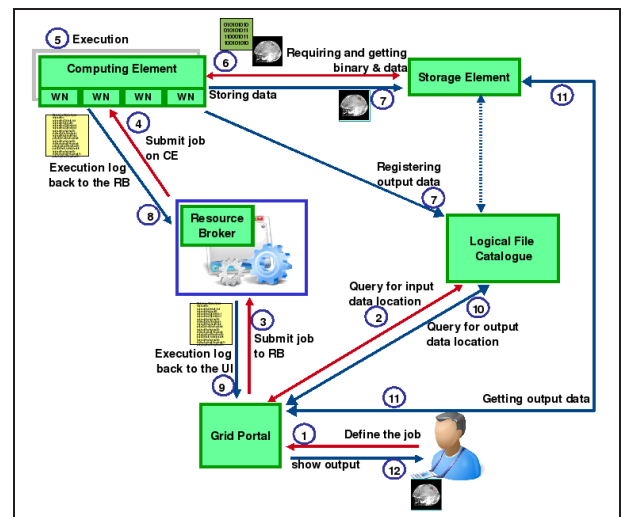


Figure 3. Job Submission: MediGrid schema

Summarizing, looking at the schema in Fig. 3 some main steps are performed:

- Job definition (steps 1-2): The user, by using the Grid

Portal, defines the job parameters (by choosing image processing algorithm, by browsing the File Catalogue to select input data, by fixing the output data logical name, ... ) through the Grafical Interface;

- Job submission (steps 3-4): The submission service on Grid Portal, submits the defined job to the Resource Broker that chooses a Computing Element;
- Job execution (steps 5-6): The first step of the job script asks to LFC for application binary and input data and then gets them directly from the Storage Element;
- Data archivation (step 7): At the end of the process the job script copies the output data directly on the SE and, at the same time, register files on the File Catalogue with the symbolic name chosen at step 1 from the user.
- Output retrieval and visualization (steps 8-12): The execution log file moves from the Working Node to the User Interface through the Resource Broker to be visualized by the user; the user can require output files that, by means of their symbolic names, are moved from the SE directly on the Grid Portal and then visualized.

As reported above, at step 1 (see Fig. 3), the User defines, through the GUI, some requirements and parameters; on server side these one will be used from the *job service* to construct the job in a Job Description Language format (JDL file). The job service is implemented by using objects and methods from native gLite standard library and from extra WMS-Proxy library [7].

```
Type="Job";
JobType="MPICH";
NodeNumber=6;

Executable="ECOCG.sh";
Arguments="6 scope scopelfc01.dsf.unina.it
           scoperb01.dsf.unina.it:2170
           scopese01.dsf.unina.it 2008/February/
           sc 1e-1 1e-4 1 1e-4 150 150 100 14";
StdOutput="ECOCG.out";
StdError="ECOCG.err";
InputSandbox="ECOCG.sh";
OutputSandbox="ECOCG.out","ECOCG.err","Denoise3d.out";
```

**Figure 4. The JDL File generated by using WMS-PROXY Java API**

In Fig. 4 we show an example of JDL file generated. We highlight (see text in bold face) some relevant attributes:

- **JobType** and **NodeNumber** used to identify a parallel job (based on MPI)
- **InputSandbox** that contains only the script file with parameters setting

- **OutputSandbox** that contains only the job execution log files.

```
#!/bin/sh
# ECOCG.sh
CPU_NEEDED=$1
VO=$2
LFC_HOST=$3
LCG_GFAL_INFOSYS=$4
VO_DEFAULT_SE=$5
NAME_PATH=$6
NAME=$7
FILEN=$NAME_PATH/$NAME
# algorithm parameter setting:
# N1, N2, N3, SIGMA, SCALESTEP,
# NSCALES, RCONVERGENCE, NOF.
OUTFILE=$NAME_PATH/$16
PROGARGS=-filename $NAME -scalestep $SCALESTEP
-sigma $SIGMA -nscales $NSCALES
-iz 1 -rconvergence $RCONVERGENCE
-n1 $N1 -n2 $N2 -n3 $N3
-ksp_type gmres -pc_type bjacobi
-sub_pc_type ilu -ksp_max_it 300
-nof $NOF

echo "Downloading binaries and input data"
echo " ====="
lcg-cp --vo $VO
lfn:/grid/$VO/MEDIGRID/SOFTWARE/Denoise3d
file:'pwd'/Denoise3d
lcg-cp --vo $VO
lfn:/grid/$VO/MEDIGRID/ECOCG/Input/$FILEN.tgz
file:'pwd'/$NAME.tgz
tar xzvf $NAME.tgz
echo " ====="

echo "Executing mpirun"
echo " ====="
mpirun -np $CPU_NEEDED -machinefile $HOST_NODEFILE
'pwd'/Denoise3d $PROGARGS ; Denoise3d.out 2;&1
echo " ====="

echo "Uploading Output files"
echo " ====="
tar czvf $NAME.OUT.tgz Stato.txt $NAME*_*
lcg-cr --vo $VO -d $VO_DEFAULT_SE
-lfn:/grid/$VO/MEDIGRID/ECOCG/Output/$OUTFILE.tgz
file:'pwd'/$NAME.OUT.tgz
echo " ====="
```

**Figure 5. An example of execution script**

In Fig. 5, we report the script file code. We highlight (see text in boldface) some relevant lines:

- **PROGARGS** variable that contains all user defined parameters
- the download of input data and binaries, from the storage to the computing resource, using **lcg-cp** command
- the execution of parallel application, using **mpirun** command
- the output data transferring on the storage and its registration on file catalogue, using **lcg-cr** command

Job monitoring system is implemented by combining two mechanisms:



- the *perusal* feature provided by WMS broker, and accessed by means of Java API, to retrieve the job execution log files at runtime
- a custom service that retrieves partial output saved by the application during its execution (see checkpointing mechanism described later on).

With the aim of modify, extend and/or improve the numerical components based on PETSc, we are working on the introduction, at *application level*, of a some kind of checkpointing methods, that will be combined with fault-tolerance mechanisms.

With an *application level* approach, the mechanisms enabling the fault-tolerance features in the grid applications, are implemented in the algorithms of computational kernels and they are not demanded to the middleware, delegating the application developer to deal with the matter of choosing and saving data appropriately. On the other hand, with this strategy, the application is able to gain an higher level of efficiency in storing and retrieving the data from which performing restarting tasks.

Both the denoising and segmentation algorithms are based on iterative schemes and, in order to recover from a fault, they can restart from those "points" corresponding to the last "correct" computed data. So we can use a kind of "*disk based checkpointing*" method, registering those data to the LFC service, at each iteration, so that we could be able to restart with the last computed data; at fault occurrence, application is automatically re-submitted on alternative resource accessing LFC service to obtain last computed data from which perform restarting task.

In Fig. 6 we show the outline of a disk-based checkpointing strategy in an iterative scheme. We highlight some key steps:

- the last "correct" computed data is retrieved from remote storage resource by LFC service (see step 1 in loop block);
- the current data is computed (see step 2 in loop block)
- the computed data is stored on remote storage resource and registered on the LFC (see step 4 in loop block)

We can observe that, such kind of *disk-based* checkpointing, give us the chance to replicate data on more remote storages (as allowed by LFC/GFAL system): data are always available, even if storage and computing resources can become unavailable. This implement a storage service reliability, also paying for overhead in data transfer.

To find and handle faults in Message Passing system, we have to modify appropriately both the application and the "high level" middleware. Thus, to use WMS automatic re-scheduling, the application has to execute the following tasks:

```

% loop over i
for i = last_computed_i + 1, ... do
  1. Retrieve by LFC service ulast_computed_i
  2. compute ui from ui-1
  3. check communication-fault: if present exit(1)
  4. Register to LFC service ui
endfor

```

**Figure 6. outline of disk-based checkpointing in an iterative algorithm.**

1. checking the presence of fault after each communication phase, and,
2. if fault is found, terminating execution with an exit non-zero code.

## 4. Conclusions and Future Works

This paper describes some work done to implement the MediGrid PSE in an LCG/gLite environment. Things are much evolved, starting from our preliminary results, and, although this experience has been realized in economy of thought, features and tools now available in the LCG/gLite middleware open new scenarios for the medical community to collaborate and share resources, information and knowledge.

As mentioned in previous section, to modify, extend and/or improve the underlying numerical components, we have to introduce some checkpointing techniques, to be combined with fault-tolerance mechanisms. To introduce fault-tolerance mechanisms, we have to modify appropriately both the application and the "high level" middleware components (i.e. PETSc).

Then our future work will involve the implementation of mechanisms that handle all communication faults, at each level of "high level" middleware hierarchy, without terminating application execution.

## References

- [1] CRESCO Project Home Page. <http://www.cresco.enea.it/>.
- [2] CYBERSAR Project Home Page. <http://www.cybersar.com>.
- [3] EGEE Project Home Page. <http://www.eu-egee.org/>.
- [4] EGI Project Home Page. <http://web.eu-egi.eu/>.
- [5] IGI Project Home Page. <http://grid.infn.it/igi/partners.html>.
- [6] ImageJ: Image Processing and Analysis in Java. <http://rsbweb.nih.gov/ij/index.html>.
- [7] Java Library for WMPProxy. <https://grid.ct.infn.it/twiki/bin/view/GILDA/ApiJavaWMPProxy>.

- [8] PI2S2 Project Home Page. <http://www.pi2s2.it/>.
- [9] S.Co.P.E. Project Home Page. <http://www.scope.unina.it>.
- [10] SPACI Project Home Page. <http://www.spaci.it/>.
- [11] S. Balay, K. Buschelman, V. Eijkhout, W. D. Gropp, D. Kaushik, M. G. Knepley, L. C. McInnes, B. F. Smith, and H. Zhang. PETSc users manual. Technical Report ANL-95/11 - Revision 2.1.5, Argonne National Laboratory, 2004.
- [12] S. Balay, K. Buschelman, W. D. Gropp, D. Kaushik, M. G. Knepley, L. C. McInnes, B. F. Smith, and H. Zhang. PETSc Web page, 2001. <http://www.mcs.anl.gov/petsc>.
- [13] S. Balay, W. D. Gropp, L. C. McInnes, and B. F. Smith. Efficient management of parallelism in object oriented numerical software libraries. In E. Arge, A. M. Bruaset, and H. P. Langtangen, editors, *Modern Software Tools in Scientific Computing*, pages 163–202. Birkhäuser Press, 1997.
- [14] G. B. Barone, D. Bottalico, V. Boccia, L. Carracciuolo, S. Pardi, M. Scognamiglio, and F. Serio. Middleware Applicativo: lo S.Co.P.E. Toolkit. In Università degli Studi di Napoli “Federico II”, editor, *Proceedings of Italian National Conference of e-Science 2008*, Napoli, 2008. CRUI.
- [15] V. Boccia, L. D’Amore, M. Guarracino, and G. Laccetti. A Grid Enabled PSE for Medical Imaging Applications: Experiences on MediGrid. In *Proceedings of The 18th IEEE Symposium on Computer-Based Medical Systems*, pages 529–536, Boston, 2005. Springer-Verlag.
- [16] S. Burke, S. Campana, P. M. Lorenzo, C. Nater, R. Santinelli, and A. Sciaba. GLITE 3.1 USER GUIDE, 2008.
- [17] L. Carracciuolo, L. D’Amore, and A. Murli. Towards a parallel component for imaging in PETSc programming environment: A case study in 3-D echocardiography. *Parallel Computing*, 32:67–83, 2006.
- [18] L. D’Amore, D. Casaburi, L. Marcellino, and A. Murli. Segmentazione di sequenze di immagini digitali mediante previsione del moto. Preprint 62, Dip. di Matematica e Applicazioni, Università degli Studi di Napoli Federico II, 2006.
- [19] E. Laure and alt. Programming the Grid with gLite. *Computational Methods in Science and Technology*, 12:33–45, 2006.
- [20] A. Murli, V. Boccia, L. Carracciuolo, L. D’Amore, G. Laccetti, and M. Lapegna. Monitoring and Migration of a PETSc-based Parallel Application for Medical Imaging in a Grid computing PSE. In P. Gaffney and J. Pool, editors, *Grid-Based Problem Solving Environments*, pages 421–432, Boston, 2007. Springer-Verlag.