A comparison between ARC and gLite for medical image processing on Grids

Tristan Glatard[1], Xin Zhou[2], Sorina Camarasu–Pop[1], Oxana Smirnova[4] and Henning Müller[2,3]

August 5, 2009

[1] Creatis–LRMN, University of Lyon, France
[2]Medical Informatics, University of Geneva, Switzerland
[3]University of Applied Sciences Western Switzerland, Sierre, Switzerland
[4]NDGF and Institute of Physics, Lund University, Sweden

Abstract

Medical imaging tasks often require large amounts of computing power or they could be improved if more computing power were available. Many medical institutions do not have any dedicated computing infrastructure for research and a way to cope with this is the use of computational Grids. These Grids can be used internally if the data can not leave the hospital network or from external infrastructure providers. Choosing/maintaining a Grid infrastructure can be a tedious tasks for researchers, as well as adapting existing applications for parallel computation on the Grid. Based on medical imaging use-cases, this article compares two widely-used middleware solutions, namely gLite and ARC (Advanced Resource Connector). Interoperability is enabled at the application level and the resulting setup is demonstrated on two use–cases combining resources from both Grids. In addition, experimental results show a simple performance comparison of data transfers and job submissions.

1 Introduction

Medical imaging is an essential part of medical diagnosis and treatment planning but processing large amounts of medical imaging data can be computationally very expensive. Only few medical institutions currently have large-scale computing infrastructures destined for imaging research, which led to the use of computational Grids in the medical imaging field [3, 15]. A variety of Grid middleware projects have been conducted over the past 20 years, from Condor [11], to Globus [6], gLite [7], and ARC (Advanced Resource Connector) [5]. For a researcher not familiar with computational Grids it is difficult to choose a particular middleware among the available solutions and most often the available resources determine this choice. Middleware comparisons, in particular for a concrete task (in this case medical imaging) are rare.

On the other hand, there are many ongoing efforts currently targeting middleware interoperability, so jobs can be exchanged, potentially easing the development of applications [20]. Regarding interoperability, problems range from very low middleware layers (e.g. interoperability among batch queues to build Grids) to higher levels (interoperability among Computing Elements to federate Grids [10]). At the application–level, there are also several motivations for interoperability:

- Sharing of applications to limit the Grid porting effort. Applications ported to a particular Grid platform can be run on another. In particular, this can be useful for widely adopted software tools.
- Sharing of data to enhance the accuracy of applications requiring large amounts of data for really meaningful results (e.g. content–based image retrieval [14]).
- Sharing of resources without an additional maintenance cost (e.g. to access very specific resources such as large clusters or clusters of Graphical Processing Units (GPUs)).

This article presents our early attempts towards application–level interoperability between ARC and gLite. Our goal is to provide a qualitative comparison for medical imaging applications. Experiments reported here are run from execution environments aiming at facilitating Grid access to non–expert users, i.e. medical image analysis researchers. Two specific environments are targeted, one being deployed at the HUG (University Hospitals of Geneva[1]) to interface with an ARC–enabled Grid resource and the other being deployed at CREATIS–LRMN (Centre de REcherche et d'Applications en Traitement de l'Image et du Signal — Laboratoire de Résonance Magnétique Nucléaire)[2] to give access to the EGEE Grid running gLite. In addition to the facilities provided by ARC and gLite, both execution environments include a workflow manager for application porting and an application–level job submitter. Execution environments and methods for application–level interoperability are first presented in section 2. Experiments for data–sharing and job submission are then reported in section 3.

2 Methods

One group (HUG) targeted data sharing and attempted to use data stored on EGEE from ARC resources in a content–based image retrieval (CBIR) application. A second group (CREATIS–LRMN) targeted resource sharing and attempted to run with ARC a radiotherapy simulation application originally ported to a gLite–based environment. This section first presents the two execution environments in 2.1. Setups for interoperability are then detailed in 2.2 (for data) and 2.3 (for jobs).

2.1 Execution environments

Both execution environments are mainly composed of a workflow (WF) description tool and a workflow engine enabling job submission, input selection, and data piping between jobs. *Figure 2.1 summarizes the components adopted by the partners and shows how they interact with the grid middleware* . A more detailed description follows.

HUG Grid setup

The HUG group has a particular Grid setup to assure that computation of data is also possible inside the hospital itself to avoid the transmission of sensitive medical information. Thus a small setup inside the hospital makes available computational power based on virtualization, Condor as computing node software and ARC to manage the jobs [16]. To ease the creation of parallel applications and gridify them the Taverna workflow system is used [17]. This also includes an ARC plugin to automatically submit the created jobs, following XRSL (eXtended Resource Specification Language) [23]. Besides the use of internal submission

[1] http://www.sim.hcuge.ch/medgift/
[2] http://www.creatis.insa-lyon.fr/

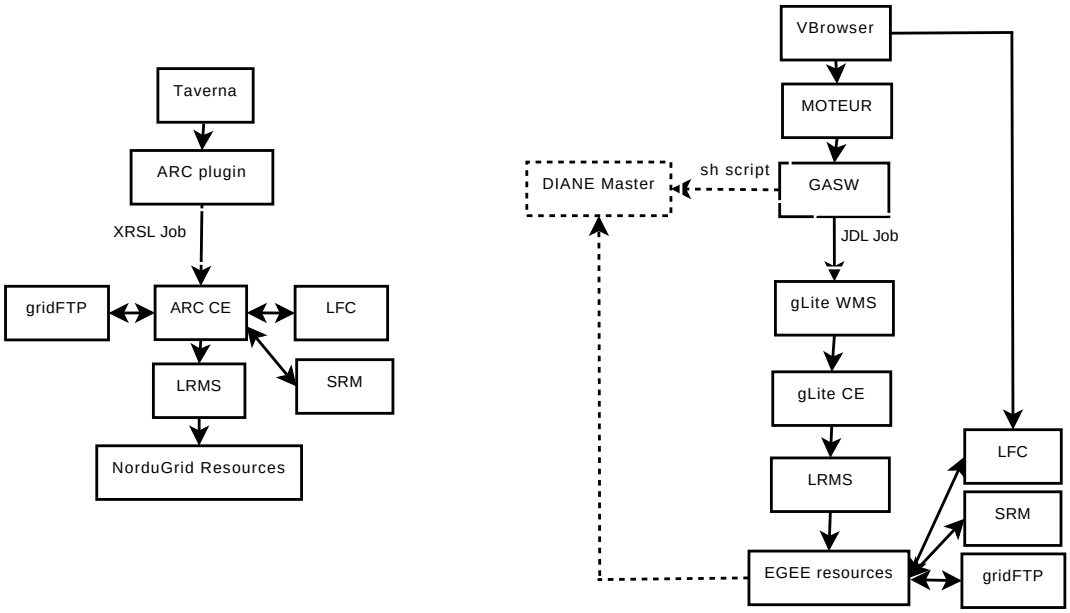

Figure 1: *Overview of the Grid environments used by HUG (left) and CREATIS-LRMN (right).*

interfaces the created applications can simply be submitted to external resources such as the KnowARC[3] virtual organization (VO) *of the Nordugrid*[4] *infrastructure* using the exactly same submission interface. *In this case, jobs are handled by the ARC Computing Element (CE) and delegated to a Local Resource Management System (LRMS) that eventually schedules them on computing resources. In/output data is handled directly by the CE which pre-/post-stages files from/to storage systems. Supported file protocols include gridFTP, Logical File Catalog (LFC) and the Storage Resource Management (SRM).*

CREATIS–LRMN Grid setup

The workflow description in the second setup relies on the Scufl (Simple Concept Unified Flow Language) language, generated using the Taverna workbench as a workflow editor [17]. Code is wrapped into workflow components using the Generic Application Service Wrapper (GASW [8]), which provides a basic command–line description language enabling input file staging, parameter specification, output file naming and transfer, as well as dependency specification. Workflows are then executed on gLite using MOTEUR [9] that generates, submits and monitors jobs on gLite complying to the Job Description Language (JDL) *as figured by plain lines on figure 2.1* . The main difference with ARC regarding job submission is that in ARC the site selection (matchmaking) is performed by the client whereas in gLite it is delegated to a global matchmaker (called Workload Management System — WMS). gLite's strategy is supposed to yield better scheduling while ARC's ensures better scalability. *Moreover, data has to be transferred by the job itself once it reaches the computing resource.* A comparison of ARC and gLite job life–cycles is reported in [10].

Alternatively *(dashed lines on figure 2.1)* , MOTEUR can also submit tasks to the DIANE pilot–job framework [13][5], offering a *pull* execution model supposed to improve performance on high–throughput systems. In DIANE, tasks are no longer pushed to computing resources but generic pilots are submitted. Once run-

[3] http://www.knowarc.eu/
[4] http://www.nordugrid.org
[5] http://cern.ch/diane/

ning, pilots connect back to a central pool, fetching tasks when available and dying otherwise. Workflow input files and results are graphically browsed and selected on Grid storage resources using the Virtual Resource Browser (VBrowser) [18, 19]. Eventually, jobs execute on resources of the `biomed` EGEE VO, external to the institution.

Data is stored on gLite Storage Elements (SE) equipped with the Storage Resource Management interface (SRM) [1]. Data files are indexed in the Logical File Catalog (LFC), which maps application–specific logical file names to their physical locations.

2.2 Data interoperability for content–based image retrieval

Part of the medical data sets used for imaging cannot be deported outside of the hospital network for privacy reasons. On the other hand, CBIR relies on databases that can be stored on external resources, potentially belonging to another Grid and VO. The goal of this subsection is to enable the execution of applications developed on ARC with databases stored on EGEE servers.

Accessing the data stored on EGEE from the protected hospital network is not straightforward. Outbound connections in the HUG have several constraints: connections can only use port 80 (HTTP/HTTPS) and are always passing through a restrictive proxy server. Thus, communication with a Grid server outside of the hospital is hardly possible by default. To help scientific projects we were allowed to circumvent some of these restrictions by using a VPN (Virtual Private Network) connection towards the network of the University of Geneva. This network then has much lower security restrictions. The group also has two servers for data processing on the University network that are used for accessing other Grid networks.

The tested CBIR application can be divided into two parts: (i) downloading the data from the EGEE Grid servers to the University network, and (ii) executing the medical image analysis application on the internal hospital Grid. The latter is not related to data interoperability, we thus focus on the first step with the purpose of evaluating feasibility and possible overhead. Two client tools are tested for data transfer: the Java LFC client of VBrowser [18] and the ARC standalone client, which is also interfaced with LFC. Other related candidate tools for evaluation include the Grid Storage Access Framework (GSAF)[6] and JavaGAT [22].

2.3 Resource sharing for radiotherapy simulation

Computationally expensive simulation experiments often require large amounts of resources that may not be available on a single Grid at a given time. For instance, radiotherapy simulation [2] benefits from hundreds ($\geq$300) of concurrent CPUs (Central Processing Units). The goal of this section is to enable the execution of applications developed for EGEE on ARC resources. *ARC resources under consideration are the ones provided by the Nordugrid infrastructure* .

Two solutions can be envisaged for this execution using the execution environment described in section 2.1: (i) DIANE submits pilot-jobs to ARC (MOTEUR still submits tasks to DIANE), or (ii) MOTEUR directly submits jobs to ARC (DIANE is not used). Solution (i) provides more interoperability since every application relying on DIANE could then be executed both on gLite and on ARC. Besides this, the solution would easily enable a joint exploitation of ARC and gLite resources for a single application. On the other hand, (ii) provides better performance since the job generation can be adapted to a particular middleware. In practice, implementing (i) raises several technical issues.

Firstly, since tasks are only fetched when the job reaches a computing resource (so–called *late binding*),

[6] http://grid.ct.infn.it/twiki/bin/view/PI2S2/GSAF

pre–staging of files cannot be implemented easily in a pilot–job framework. As a consequence, files need to be transferred onto the computing node by the task itself, which not only underexploits the features of ARC but is also technically heavy to implement since neither the data transfer client nor the user proxy are present on the computing node by default. In addition, it may lead to an unnecessary and uncontrollable overload of the storage service.

Secondly, in all cases, task generation by MOTEUR has to be adapted to the execution on an ARC computing node to accommodate, e.g., syntax differences in data clients. This is problematic given the late binding of tasks provided by pilot–jobs.

These reasons led us to implement solution (ii). The GASW was extended to support submission to ARC clusters. Beyond minor changes in job submission, monitoring, and status syntax, this required the adaptation of the job description format (from JDL to XSRL) and of the job content (from explicit to automatic data transfers).

Data transfers from EGEE to NorduGrid were performed using ARC's support for LFC (LFC locations can be specified in XRSL, the files being automatically transferred to/from EGEE resources). Because of the numerous ambiguities, only non–DPM EGEE SEs (Storage Elements) could be used, though[7]. A more important issue is that the VO–specific physical locations are not automatically generated by the generic ARC client, whereas it is done by the gLite LFC client for the registered EGEE VOs. The SRM output directory path thus has to be explicitly suggested in the configuration of the workflow manager that uses ARC, while only the SE host has to be specified for the EGEE infrastructure. This is potentially problematic in case of changes in the configuration of an EGEE SE (e.g. upgrade leading to change of the directory hierarchy or permissions).

Authorization of an EGEE user on NorduGrid clusters was easily performed by registering the X509 certificate in the `knowarc.eu` VO. However, being a member of two VOs led to some technical issues when information about the VOs is stored in the proxy itself (i.e. the proxy contains an extension obtained from the VO Management Service — VOMS). Due to the lack of a relevant specification that would formalize processing of multiple VOMS extensions, proxies containing two or more VOMS extensions are treated in an arbitrary manner by SRM services, often leading to data transfer errors. Since submission to ARC clusters does not require any VOMS extension, we coped with this issue by using a proxy with an EGEE VOMS extension only.

It has to be mentioned that the ease of installation of the ARC client greatly facilitated this implementation. The ARC client and gLite UI (User Interface) are easily able to be installed on a single Linux box. Well–packaged distributions like the one maintained by the Dutch VL–e (Virtual–Lab for eScience) project[8] now allow installing and configuring a gLite UI in ca. 20 minutes, including download and configuration. Because of its reduced dependencies, only 2 minutes were required for installation of an ARC client. This process was also less invasive and several Linux flavors are supported.

[7] SRM standard currently does not allow to identify neither the transfer protocol, nor the necessary end-point details such as port number, leading to incompatible implementations

[8] http://poc.vl-e.nl/

	data on EGEE		data on ARC	
	download	upload	download	upload
VBrowser	4523	1022	X	X
ARC–client	4451	997	4301	911

Table 1: Comparison of transfer speed (KB/s) to ARC computing resources for data stored on EGEE and ARC clusters with a catalog service.

	data on EGEE		data on ARC	
	download	upload	download	upload
VBrowser	339	116	X	X
ARC–client	361	110	345	112

Table 2: Comparison of transfer speed (KB/s) to ARC computing resources for data stored on EGEE and ARC clusters with Catalog service, using VPN in both cases.

3 Experiments and results

3.1 Evaluation of the access to data on EGEE and ARC clusters

The ARC standalone client is a Linux command line tool that offers not just ARC–specific job management functionality, but also some fundamental data transfer commands. Its interoperability with various data management services — either plain GridFTP servers, SRM or LFC — is demonstrated in [12]. However, in HUG, users are used to Windows–like graphical interfaces. VBrowser provides such an interface for data management. It also adapts necessary protocols to access the data on both gLite and ARC.

Two virtual organizations(VO) are used for this test: the `Biomed` VO based on the gLite middleware and the `knowarc.eu` VO of the EU KnowARC project. The test file comprises 40MB of a compressed image collection; physically it is located in Italy (the gLite server, `Biomed` VO) and Hungary (the ARC server, `knowarc.eu` VO). Two different data indexing services were used: the Globus Replica Location Service (RLS) [4] for ARC and the LFC catalog and indexing service for EGEE gLite. ARC can use both LFC and RLS for data indexing, while gLite currently supports only LFC. Incidentally, VBrowser cannot deal with RLS either, thus only the other interoperability possibilities were tested.

Tests are performed both on university network (Table 1) and using a VPN from the hospital network (Table 2). Client tools are installed both on a server located in University of Geneva and a workstation inside the HUG. For reasons explained beforehand, the communication from the HUG has to be through a VPN. A VPN encrypts the communication in both directions, which reduces the download/upload speed. When performing the tests from the university network the difference of speed is not significant and depends on the network speed (EGEE sites are network–wise closer to the HUG than the ARC ones). It should be pointed out that ARC offers a light–weight storage element named Smart Storage Element, which uses the HTTPS protocol for the data transfer. This can help reducing the communication overhead.

Regarding the comparison between VBrowser and ARC client, no significant difference was detected in terms of overhead. *Comparison tests on larger data sets are part of our future work.* Client tools with a windows–like GUI are regarded as more user–friendly. The ARC client, though lacking a GUI, supports more existing URL formats, which can be an advantage for users who want to access different storage systems.

	input archive	GATE release	GATE wrapper script	Output archive
size	1.6 MB	28 MB	7.9 KB	697 B
SE location	DE	GR	NL	BG

Table 3: GATE input/output file sizes and locations.

3.2 Joint execution of radiotherapy simulations on EGEE and ARC clusters

Using the setup described in section 2.3 we were able to execute on ARC resources a workflow initially developed on EGEE. The underlying application is GATE, a Monte Carlo simulation code currently used by more than 1000 users[9] and used here for radiotherapy simulation as described in [21]. Such Monte Carlo simulations are divisible–load problems, i.e. they can be divided into as many tasks as wanted. We here consider a 3h49'-simulation (average on ARC and gLite clusters used for the experiment) split into 50 jobs.

Each of the 50 tasks requires 3 input files and produces 1 output archive wrapping all the results. The job itself is wrapped into a script performing in/output data transfers (for gLite only), checking execution correctness and writing monitoring information such as total run time in the job console. Four gLite SEs spread all over Europe were used. File sizes and locations are reported in Table 3.

The experiment was repeated 5 times (experiments are coined batch 1 to 5 in the following). Each batch was simultaneously submitted to ARC and gLite. To have similar matchmaking conditions, job submission was restricted to 3 NorduGrid sites and 3 EGEE sites. MOTEUR was configured to resubmit failed jobs up to 3 times. It should be noted, however, that although matchmaking conditions were comparable, ARC–enabled sites are voluntarily academic community contributions supported on a best–effort basis, while gLite–enabled sites were of a professional HPC grade, offering higher levels of service.

For each successful job, the submission, matchmaking, queuing, input transfer, running, output transfer, worktime and total round–trip times were measured as shown in Table 4. Some of the times were estimated from the job status reported by the Grid Information System (IS) and others were obtained from job or LRMS (Local Resource Management System) logs. In Table 4, the job states refer to the gLite[10] and ARC[11] user guides.

Because KnowARC and EGEE clusters used for the experiment have a different number of nodes and CPU characteristics, the total round–trip times cannot be compared. In particular, job queuing and running times are expected to be largely affected by those differences. Instead, we remove those two values from the total round–trip time to define a comparable Grid overhead defined as $\{5\}-(\{3\}+\{4.b\})$ referring to the notations of Table 4.

This comparable Grid overhead breaks down to the sum of the submission, matchmaking, input transfer, output transfer and *infrastructure overhead* (*ISO*). The latter measures the difference between the *real* job worktime (i.e. obtained from the job and/or LRMS stdout) and the worktime given by the information system, i.e., using notations of Table 4:

$$ISO_{gLite} = \{4\}-(\{4.a\}+\{4.b\}+\{4.c\}) \quad and \quad ISO_{ARC} = \{4\}-(\{4.b\}+\{4.c\})$$

Table 5 reports the comparable overhead of the successful jobs for the 5 batches and how it breaks down to submission, matchmaking, in/output transfer and ISO. The latter accounts for the largest proportion of the

[9] http://www.fgate.fr/
[10] http://glite.web.cern.ch/glite/documentation/userGuide.asp
[11] http://www.nordugrid.org/documents/ui.pdf

	gLite		ARC	
Measured time	Start state in IS	End state in IS	Start state in IS	End state in IS
{1} - Submission	Not submitted	Successfully subm.	Not submitted	Successfully subm.
{2} - Matchmaking	Submitted	Scheduled	not applicable	
{3} - Queuing	Scheduled	Running	Successfully subm.	INLRMSR
{4.a} - Input transfer	Job stdout		LRMS stdout	
{4.b} - Running	Job stdout			
{4.c} - Output transfer	Job stdout		LRMS stdout	
{4} - Worktime	Running	Completed	Running	Finished
{5} - Total round-trip	Not submitted	Completed	Not submitted	Finished

Table 4: Definition of measured times on ARC and gLite.

	Batch 1		*Batch 2*		*Batch 3*		*Batch 4*		*Batch 5*	
	gLite	*ARC*	*gLite*	*ARC*	*gLite*	*ARC*	*gLite*	*ARC*	*gLite*	*ARC*
Number of jobs	*49*	*50*	*49*	*50*	*48*	*50*	*47*	*49*	*48*	*50*
***Subm.** Mean (s)*	*3.7*	*15.5*	*3.6*	*14.8*	*4*	*16.1*	*4.3*	*14.4*	*3.8*	*13.8*
Stdev (s)	*0.87*	*12.2*	*0.86*	*9.9*	*0.97*	*13.1*	*1.6*	*10.3*	*1.1*	*10.4*
***Matchm.** Mean (s)*	*26.6*	*0*	*637.4*	*0*	*27.9*	*0*	*25.2*	*0*	*28.8*	*0*
Stdev (s)	*6.8*	*0*	*862.7*	*0*	*7.4*	*0*	*6.5*	*0*	*6.5*	*0*
***In. trsf.** Mean (s)*	*47.7*	*26.6*	*44.4*	*22.4*	*44.8*	*25.0*	*42.9*	*22.4*	*42.8*	*22.8*
Stdev (s)	*8.9*	*8.7*	*6.0*	*2.8*	*4.4*	*6.0*	*6.2*	*4.7*	*6.6*	*4.1*
***Out. trsf.** Mean (s)*	*7.9*	*18.3*	*12.5*	*17.4*	*8.7*	*14.2*	*9.4*	*17*	*8.0*	*14.7*
Stdev (s)	*1.0*	*4.2*	*18.4*	*2.8*	*2.3*	*1.5*	*6.2*	*9.4*	*1.8*	*2.2*
***ISO** Mean (s)*	*634.3*	*1255.4*	*333.7*	*1280.6*	*523.5*	*1257.8*	*694.2*	*1321*	*697.2*	*1242.3*
Stdev (s)	*635.3*	*x*	*516.0*	*x*	*531.4*	*x*	*610*	*x*	*537.8*	*x*
***Comp. over.** Mean (s)*	*719.9*	*1289.4*	*1032.1*	*1312.4*	*608*	*1288*	*776*	*1352*	*781*	*1270*
Stdev (s)	*721.4*	*237.4*	*700.1*	*202.5*	*532.1*	*273.9*	*611.4*	*349*	*536.2*	*223.4*

Table 5: *Overhead comparison between ARC and gLite on GATE radiotherapy application. Each batch corresponds to a repetition of the experiment.*

comparable overhead in both cases. In average, it is close to 10 minutes for gLite (576s) and 20 minutes for ARC (1271s). *Data transfers have similar* performance both on ARC and gLite, which confirms the ability of the ARC client to efficiently handle files stored on EGEE, as shown in section 3.1. As explained in section 2.1, ARC does the matchmaking on the client side, i.e., during the submission process, which explains why the perceived submission time on ARC is higher than on gLite (about 4 times on average). However, the main result is that in all cases, the sum of submission and matchmaking times on gLite are significantly higher than on ARC. Those results show that ARC's strategy is globally less penalizing that gLite's in our case. In particular, batch 2 shows that an overloaded WMS dramatically penalizes the experiment, which could not occur on ARC. On the other hand, one should keep in mind that gLite's strategy may lead to better scheduling, thus reducing the job queuing times in LRMS, which is not considered here. Moreover, ARC's strategy may also lead to scalability issues when several experiments are run from the same client.

4 Conclusions

In this work, we successfully implemented data and resource sharing between ARC and gLite. This allowed us to (i) run a CBIR application on ARC resources using data stored on EGEE resources and (ii) easily deploy on ARC an application developed for EGEE. This was tested in high–level graphical execution environments targeting medical imaging researchers. Both ARC and gLite support such solutions that are easy to use and can be of interest for researchers.

Scenarios are different for a research group that is inside a medical institution and research groups being on more open University networks, in particular concerning network connectivity. Data that can be treated inside and outside of medical institutions might also be different. Secondary data use in general is not easy as legal constraints often make it had to acquire data sets.

Beyond application–level interoperability, this setup enabled a comparison between ARC and gLite for medical imaging. Concerning data sharing, the main results are that (i) ARC's data transfer client manages to reach similar performance as gLite's to handle data stored on EGEE, (ii) the performance drop in using the VBrowser Java driver is not significant, (iii) whether data is stored on EGEE or on ARC does not significantly impact transfers and (iv) using VPN solutions to circumvent connectivity limitation in hospitals dramatically penalizes the performance. Regarding resource sharing, results show that (i) both for gLite and ARC, the infrastructure overhead accounts for most of the job latency and (ii) on the tested application, ARC's strategy of implementing matchmaking on the client side yields better performance than gLite's. All in all, this kind of study may be of importance in the current efforts for federating European Grids in a common European Grid Initiative (EGI).

5 Acknowledgements

We thank Piter T. De Boer for technical support on the VBrowser LFC client and David Sarrut for the help given to the porting of the GATE application on EGEE. This work has been funded (supported) by the EGEE–III INFSO–RI–222667 European project. This work was partially supported by the EU 6th Framework Program in the context of the KnowARC project (IST 032691) and by the Swiss National Science Foundation (FNS) in the context of the Talisman–2 project (project 200020 118638).

References

[1] A. Sim, A. Shoshani and others. The Storage Resource Manager Interface (SRM) Specification v2.2. GFD-R-P.129, 2008. 2.1

[2] J. Badel, L. Guigues, and D. Sarrut. ThIS : a Geant4-based Therapeutic Irradiation Simulator. In *1st European Workshop on Monte Carlo Treatment Planning*, 2006. 2.3

[3] V. Breton, R. Medina, and J. Montagnat. DataGrid, Prototype of a Biomedical Grid. *Methods of Information in Medicine*, 42(2):143–148, 2003. 1

[4] A. L. Chervenak et al. Performance and Scalability of a Replica Location Service. In *HPDC'04*, pages 182–191. IEEE Computer Society Press, 2004. 3.1

[5] M. Ellert, M. Grønager, A. Konstantinov, B. Kónya, J. Lindemann, I. Livenson, J. Langgaard Nielsen, M. Niinimäki, O. Smirnova, and A. Wäänänen. Advanced resource connector middleware for lightweight computational Grids. *FGCS*, 23(2):219–240, 2007. 1

[6] I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. *The International Journal of Supercomputer Applications and High Performance Computing*, 11(2):115–128, 1997. 1

[7] F. Gagliardi, B. Jones, M. Reale, and S. Burke. European DataGrid project: Experiences of deploying a large scale testbed for e-science applications. In *Performance Evaluation of Complex Systems: Techniques and Tools, Performance 2002*, pages 480–500, 2002. 1

[8] T. Glatard, J. Montagnat, D. Emsellem, and D. Lingrand. A Service-Oriented Architecture enabling dynamic services grouping for optimizing distributed workflows execution. *FGCS*, 24(7):720–730, 2008. 2.1

[9] T. Glatard, J. Montagnat, D. Lingrand, and X. Pennec. Flexible and efficient workflow deployement of data-intensive applications on grids with MOTEUR. *IJHPCA*, 22(3):347–360, 2008. 2.1

[10] M. Gronager, D. Johansson, J. Kleist, C. Sottrup, A. Waananen, L. Field, D. Qing, K. Happonen, and T. Linden. Interoperability between ARC and gLite . In *eScience*, pages 493–500, 2008. 1, 2.1

[11] M. Litzkov, M. Livny, and M. Mutka. Condor — a hunter of idle workstations. In *Proceedings of the 8th international conference on distributed computing*, pages 104–111, 1988. 1

[12] M. Mambelli. OSG Storage Elements and ATLAS DDM, 2008. 3.1

[13] J. Mosciki. Distributed analysis environment for HEP and interdisciplinary applications. *Nuclear Instruments and Methods in Physics Research A*, 502:426429, 2003. 2.1

[14] H. Müller, N. Michoux, D. Bandon, and A. Geissbuhler. A review of content–based image retrieval systems in medicine – clinical benefits and future directions. *Int. Journal of Medical Informatics*, 73:1–23, 2004. 1

[15] M. Niinimäki, X. Zhou, A. Depeursinge, A. Geissbuhler, and H. Müller. Building a community grid for medical image analysis inside a hospital, a case study. In *MICCAI-Grid'08*, pages 3–12, 2008. 1

[16] M. Niinimäki, X. Zhou, A. Depeursinge, A. Geissbuhler, and H. Müller. Building a community grid for medical image analysis inside a hospital, a case study. *FGCS*, 2009. 2.1

[17] T. Oinn, M. Addis, J. Ferris, D. Marvin, M. Senger, M. Greenwood, T. Carver, K. Glover, M. Pocock, A. Wipat, and P. Li. Taverna: A tool for the composition and enactment of bioinformatics workflows. *Bioinformatics journal*, 17(20):3045–3054, 2004. 2.1, 2.1

[18] S. Olabarriaga, P. de Boer, K. Maheshwari, A. Belloum, J. Snel, A. Nederveen, and M. Bouwhuis. Virtual Lab for fMRI: Bridging the Usability Gap. In *e-Science'06*, 2006. 2.1, 2.2

[19] S. Olabarriaga, T. Glatard, K. Boulebiar, and P. de Boer. From 'low-hanging' to 'user-ready': initial steps into a healthgrid. In *HealthGrid'08*, pages 70–79, 2008. 2.1

[20] M. Riedel, editor. *Int. Grid Interoperability and Interoperation Workshop*. IEEE, 2008. 1

[21] D. Sarrut and L. Guigues. Region-oriented CT image representation for reducing computing time of monte carlo simulations. *Med Phys*, 35(4), 2008. 3.2

[22] R. van Nieuwpoort, T.Kielmann, and H. Bal. User-friendly and reliable grid computing based on imperfect middleware. In *SC'07*, 2007. 2.2

[23] X. Zhou, H. Krabbenhöft, M. Niinimäki, A. Depeursinge, S. Möller, and H. Müller. An easy setup for parallel medical image processing: Using Taverna and ARC. In *HealthGrid'09*, 2009. 2.1