FutureGrid 2012 Project Challenge: Interoperable and Standards-based Distributed Cyberinfrastructure and Applications

Andre Luckow   Andre Merzky   Mark Santcroos   Ole Weidner   Shantenu Jha

Abstract

1 Introduction

The scope, scale and variety of distributed computing infrastructures (DCIs) currently available to Scientists and CS researchers is both an opportunity, and a challenge. An opportunity, because the DCIs' scope is such that they cover the need of a vast number of end users. A challenge, because providing interoperable tools and applications on the diversity of DCIs is an incredible challenge.

Our group has been pursuing the latter path, attempting to provide generic programming abstractions and application frameworks, which are to support scientists and application programmers, and which are intended to simplify and increase the utilization of DCIs. Providing those components and frameworks interoperably is only possible with rigorous verification on a variety of infrastructure types. Also, their validity can only be ensured by demonstrating and testing their viability on different infrastructures.

Against this backdrop, this paper presents a selection of results from FutureGrid's SAGA project[1] which make use of FutureGrid to develop, test, verify and use software components and frameworks. We discuss our work on P*, a model for pilot abstractions, and related implementations which demonstrate (amongst others) interoperability between different pilot job frameworks. We also discuss standards based approaches to software interoperability, and the related development challenges – including SAGA as a standards based generic access layer to DCIs. Finally, we discuss the role of FutureGrid for the education on scientific computing, focusing again on the role of interoperability in that context.

2 Abstraction and Runtime Environments for Distributed Cyberinfrastructure

The seamless uptake of distributed infrastructures by scientific applications has been limited by the lack of pervasive and simple-to-use abstractions at multiple levels – at the development, deployment and execution stages [2]. Pilot-Jobs (PJ) provide an effective abstraction for dynamic execution and resource utilization in a distributed context. Not surprisingly, PJs have been one of the most successful abstractions in distributed computing.

FutureGrid has been an important resource for our research in Pilot-based abstractions and for the development of the BigJob PJ framework[3].

2.1 P* - A Model for Pilot-Abstractions

Pilot-Jobs (PJ) have become one of the most successful abstractions in distributed computing. In spite of extensive uptake, there does not exist a well defined, unifying conceptual model of Pilot-Jobs, which can be used to define, compare and contrast PJ implementations. This presents a barrier to extensibility and

[1] `https://portal.futuregrid.org/projects/42`

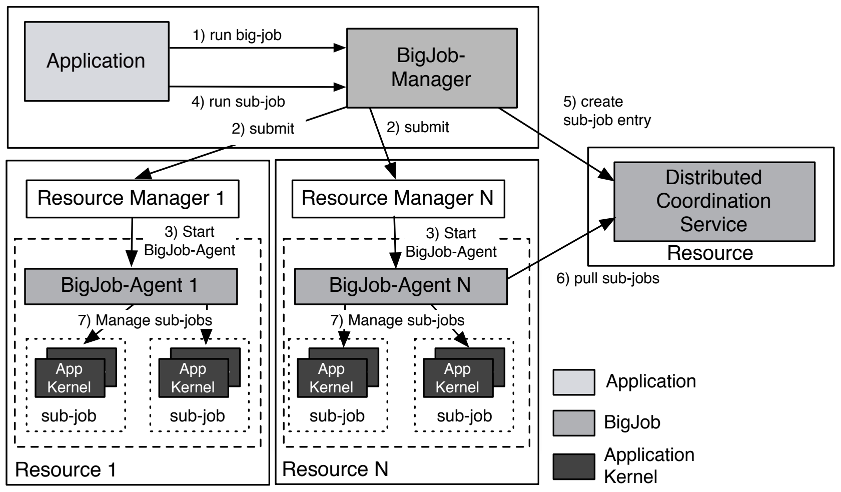


Fig. 1: **BigJob Architecture:** The BJ architecture resembles many elements of the P* Model. The BigJob-Manager is the central Pilot-Manager, which orchestrates a set of Pilots. Each Pilot is represented by a decentral component referred to as the BigJob-Agent.

interoperability. The P* model [4, 5] provide a minimal but complete model of Pilot-Jobs, which has been successfully applied to different Pilot-Job frameworks, e. g. Condor and DIANE.

The P* Model defines the typical elements found in a PJ framework: a *Pilot* is the entity that actually gets submitted and scheduled on a resource. The PJ provides application (user) level control and management of the set of allocated resources. A *Compute Unit (CU)* encapsulates a self-contained piece of work (a task) specified by the application that is submitted to the Pilot-Job framework. The *Pilot-Manager (PM)* is responsible for (i) orchestrating the interaction between the Pilots and CUs and (ii) decisions related to internal resource assignment. The Pilot-API [7] provides an abstract, unified interface to PJ frameworks that adhere to the P* Model.

2.2 BigJob and BigData

FutureGrid has been an important testbed for the development of BigJob [3] and BigData [5, 6]. BigJob (BJ) is a SAGA-based Pilot-Job (PJ) framework that implements the Pilot-API. BJ has been designed to be general-purpose and extensible. While BJ has been originally built for HPC infrastructures, such as FutureGrid and XSEDE, it is generally also usable in other environments, such as OSG. This extensibility mainly arises from the usage of SAGA as a common API for accessing distributed resources interoperably (see section 3).

Figure 1 illustrates the BJ architecture: The BJ-Manager is the Pilot-Manager responsible for coordinating the different components of the frameworks. The BigJob-Agent is the actual Pilot that is submitted to a resource. BigData extends the Pilot-Job concept to data. BigData provides late-binding capabilities for data by separating the storage allocation and application-level [4]. Similar to BigJob, it is comprised of two components: the BD-Manager and the BD-Agents, which are deployed on the physical resources.

We used FutureGrid to evaluate the overheads typical associated with PJ frameworks. For the evaluation of the communication & coordination (c&c) subsystem of BigJob and DIANE we run a different number of very short running (i. e. zero workload) tasks on Alamo/FG concurrently. In general, the c&c systems used are mostly insensitive to the number of coordinated tasks.

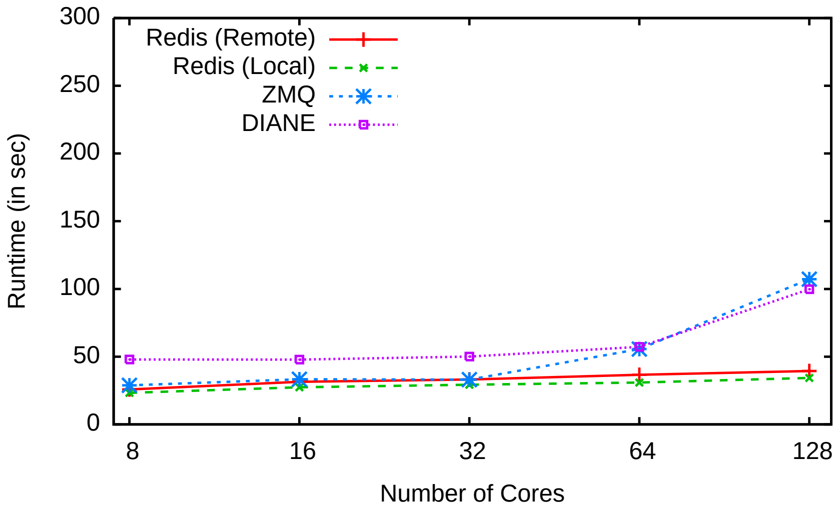


Fig. 2: **Pilot-Job Coordination Mechanism:** The runtime of a workload of 4 tasks per core, i. e. 32 - 512 tasks, using different Pilots and configuration. For BJ-Redis the runtime increases only moderately, the client-server-based implementations BJ-ZMQ and CORBA-based DIANE show particularly a steep increase when going from 64 to 128 cores.

Figure 2 illustrates the scalability of BJ and DIANE with respect to the number of cores and tasks managed by Pilot. For this purpose, we execute 4 tasks per core, i. e. between 32 and 512 tasks. BigJob with Redis (local) shows almost linear scaling up to 128 cores. BigJob with Redis (remote) imposes an increase of about 14 %. BigJob with ZeroMQ performs very well with lower core counts; with larger core counts, the runtimes increase, indicating a potential scalability bottleneck. Due to higher startup overhead, at lower core counts DIANE shows a longer runtime than ZeroMQ or Redis. At higher core counts DIANE behaves similar to BigJob/ZeroMQ, but shows a greater increase in the overall runtime. This increase is likely attributable to the single central manager in DIANE's CORBA-based client-server architecture. Using Redis as central data space for BigJob decouples Pilot-Manager and Agent, yielding better performance in particular with many replicas.

2.3 Pilot-Job Interoperability

In principal, two types of interoperability between Pilot-Jobs and infrastructure exist: the first is the usage of a given PJ framework on different infrastructures; in the scenario examined, BigJob is used on different infrastructures by invoking different SAGA adaptors. The second is the usage of distinct PJ frameworks via the Pilot-API, i.e., interoperability between PJ frameworks.

Figure 3 shows how the Pilot-API and BigJob/SAGA enables the user to run applications interoperably on different production and research infrastructures. For the purpose of this investigation, we evaluate the performance of BFAST, a genome sequencing application. For this purpose we use a problem set consisting of 1.9 GB reference genome and index files as well as 128 read files (each 170 MB). BFAST is very I/O sensitive – we observed for example, an I/O bottleneck if many BFAST CUs are run on the same shared file system. The Pilot-API enables applications to scale to different infrastructures in such cases. As shown in the figure, the FG resource was able to achieve the best performance.

Figure 4 shows how to run multiple (different) Pilots on multiple infrastructures concurrently. In configuration 1 we utilize SAGA adaptors to run BigJob concurrently on FutureGrid:India and XSEDE:Trestles. In configuration 2 and 3, we show PJ framework interoperability by concurrently running BigJob and DIANE on FutureGrid:India and EGI (2), as well as Condor and BigJob on OSG and XSEDE:QueenBee (3). It is

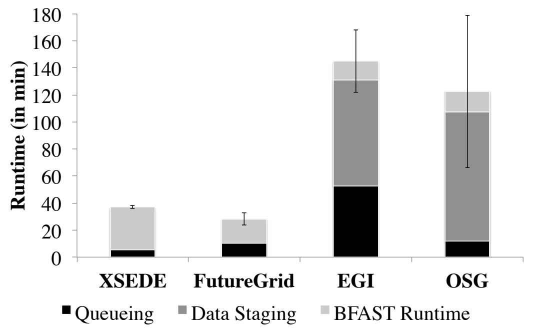


Fig. 3: **BigJob/SAGA Interoperability on XSEDE, FutureGrid, EGI and OSG:** Running 128 BFAST match tasks on 128 cores. The longer runtimes on EGI and OSG are mainly caused by longer queuing times and the necessity to stage all input files.

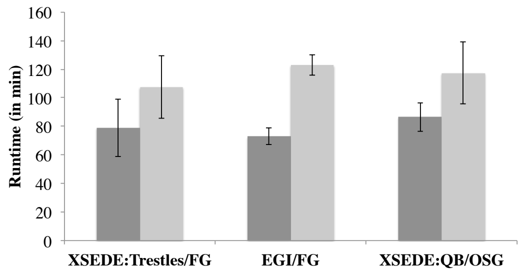


Fig. 4: **PJ Framework Interoperability:** Average runtime of 128 BFAST CUs on different infrastructures. CUs are equally distributed across the two infrastructures.

worth reiterating, that to the best of our knowledge, the latter scenario, wherein different PJ frameworks are utilized concurrently for the same application (whether on the same infrastructure or distinct) has never before been realized. We attribute this to the use of the Pilot-API. For all scenarios, we run the same BFAST scenario as in Figure 3; we run 64 CUs on each infrastructure, i. e. in total 128 CUs.

In configuration 1, one BJ Pilot is submitted to Trestles and one BJ Pilot to FG/India. The overall runtime is the sum of the file staging and the actual compute time on the respective resource. T_R of the distributed run improved more than 50 % compared to previous runs on only one resource, i.e., Trestles or India only, mainly due to the minimization of the incoming bandwidth bottleneck by distributing the load to two sites. Configuration 2 and 3 in Figure 4 demonstrate that two PJ frameworks can be utilized concurrently using the Pilot-API. As previously, the overall performance heavily depends on the time required for staging the files. Further, some performance overhead is induced by the distributed coordination necessary particular in case C2 where the BJ manager, agent and the central Redis database are highly distributed. The last column shows the result of the OSG and XSEDE:QB run. Since QueenBee is an older XSEDE machine, T_R on this machine is much longer than T_R on OSG.

3 Standards-based Interoperability

There are two major approaches to allow applications to interoperably use DCIs: system level interoperability (SLI) and application level interoperability (ALI) [8]. SLI is the (often standards based) approach to federate multiple DCIs on middleware and operational level, and to allow applications to use the combined set of resources as if it were a single DCI. That implies the exchange of accounting, authorization, authentication, logging, brokering and resource management information, and others, on system level. Very often, SLI is implemented as delegation from one DCI to the other.

ALI on the other hand does not require any activity on infrastructure level at all, but instead relies on the ability of the application to utilize multiple independent DCIs at the same time. While ALI is usually easier to achieve, it typical increases the burden on the application layer with the task to abstract the differences between DCIs.

On real-world infrastructures, we usually encounter a need for both approaches: while system-level standardization has found uptake on a variety of infrastructures (for example, BES is available on XSEDE, FutureGrid, EGI, NorduGrid, Naregi and others), the level of SLI varies widely and is often not the default modus operandi (BES again is not, as of yet, the default interface on XSEDE and FutureGrid, and thus not available on all resources).

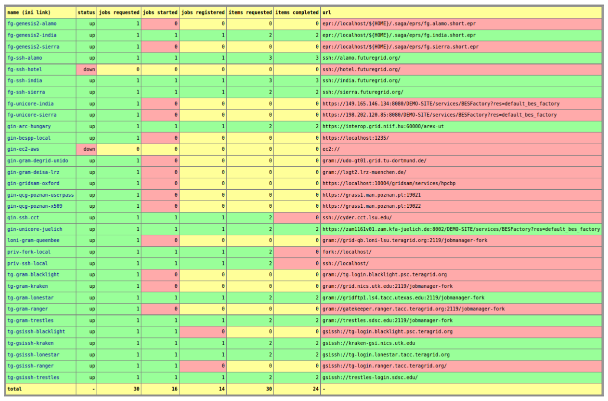
As interoperability is a driving motivator behind the P* research efforts described in the previous sections, we see the ultimate need for application level interoperability, to complement SLI. We use SAGA [1] to interface to a wide set of other standards based backends (such as GridFTP, BES, DRMAA), but also non-standards based ones (such as EC2/AWS/Eucalyptus, Condor, Globus/GRAM) – and thus utilizes SLI where available.

FutureGrid has been used in several critical aspects of the standards-based approach. First, we use FutureGrid as experimental environment to aid in the evolution of standards and for associated interoperability experiments on both the SLI and ALI layer (see sec. 3). Secondly, we use a variety of backends provided by FutureGrid for developing SAGA as the standards based access layer for our application level experiments (see sec. 3). And finally, we use FutureGrid in its capacity as XSEDE targeted test environment to prepare and harden our software, and its related packaging and deployment procedures, towards enrolment on XSEDE, and on other infrastructures such as EGI (see sec. 3).

Standardization and OGF Interop Activities

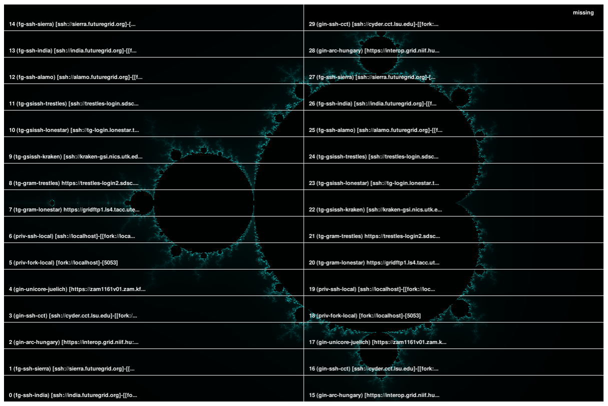
Standardization of interfaces and protocols is a crucial component for interoperability, even more so for SLI than for ALI. For the target infrastructures of our research efforts (XSEDE, EGI, Naregi, OSG, FutureGrid), OGF related standards like BES, JSDL, GLUE, OCCI and SAGA are very relevant, but are largely still evolving and are still implemented and deployed. The OGF Community Group 'GIN' (Grid Interoperation Now) is, for the scope of OGF, the umbrella group to host and coordinate efforts for interop testing and deployment. Resources on FutureGrid have been used both to develop and harden standard implementations as Genesis, Unicore and (from our end) SAGA, and also to test and demonstrate the SLI and ALI capabilities across those implementations, across a wide set of infrastructures.

Within the scope of OGF, we have been showing a proof-of-concept distributed application (master-worker pattern, computing the Mandelbrot set – see fig. 5), which is able to concurrently utilize more than 30 systems distributed world-wide, which run >10 different middleware stacks, for a single application instance[2]. On FutureGrid alone, we have been using >10 submission endpoints on 4 different stacks for those experiments. FutureGrid's capacity as research environment is greatly supporting that type of endeavour.



name (ini link)	status	jobs requested	jobs started	jobs registered	items requested	items completed	url
fg-genesis2-alamo	up	1	0	0	0	0	epr://localhost/${HOME}/.saga/eprs/fg.alamo.short.epr
fg-genesis2-india	up	1	1	1	2	2	epr://localhost/${HOME}/.saga/eprs/fg.india.short.epr
fg-genesis2-sierra	up	1	0	0	0	0	epr://localhost/${HOME}/.saga/eprs/fg.sierra.short.epr
fg-ssh-alamo	up	1	1	1	3	3	ssh://alamo.futuregrid.org/
fg-ssh-hotel	down	0	0	0	0	0	ssh://hotel.futuregrid.org/
fg-ssh-india	up	1	1	1	3	3	ssh://india.futuregrid.org/
fg-ssh-sierra	up	1	1	1	2	2	ssh://sierra.futuregrid.org/
fg-unicore-india	up	1	0	0	0	0	https://149.165.146.134:8080/DEMO-SITE/services/BESFactory?res=default_bes_factory
fg-unicore-sierra	up	1	0	0	0	0	https://198.202.120.85:8080/DEMO-SITE/services/BESFactory?res=default_bes_factory
gin-arc-hungary	up	1	1	1	2	2	https://interop.grid.niif.hu:60000/arex-ut
gin-bespp-local	up	1	0	0	0	0	https://localhost:1235/
gin-ec2-aws	down	0	0	0	0	0	ec2://
gin-gram-degrid-unido	up	1	0	0	0	0	gram://udo-gt01.grid.tu-dortmund.de/
gin-gram-deisa-lrz	up	1	0	0	0	0	gram://lxgt2.lrz-muenchen.de/
gin-gridsam-oxford	up	1	0	0	0	0	https://localhost:10004/gridsam/services/hpcbp
gin-qcg-poznan-userpass	up	1	0	0	0	0	https://grass1.man.poznan.pl:19021
gin-qcg-poznan-x509	up	1	0	0	0	0	https://grass1.man.poznan.pl:19022
gin-ssh-cct	up	1	1	1	2	0	ssh://cyder.cct.lsu.edu/
gin-unicore-juelich	up	1	1	1	2	2	https://zam1161v01.zam.kfa-juelich.de:8002/DEMO-SITE/services/BESFactory?res=default_bes_factory
loni-gram-queenbee	up	1	0	0	0	0	gram://grid-qb.loni-lsu.teragrid.org:2119/jobmanager-fork
priv-fork-local	up	1	1	1	2	0	fork://localhost/
priv-ssh-local	up	1	1	1	2	0	ssh://localhost/
tg-gram-blacklight	up	1	0	0	0	0	gram://tg-login.blacklight.psc.teragrid.org
tg-gram-kraken	up	1	0	0	0	0	gram://grid.nics.utk.edu:2119/jobmanager-fork
tg-gram-lonestar	up	1	1	1	2	2	gram://gridftp1.ls4.tacc.utexas.edu:2119/jobmanager-fork
tg-gram-ranger	up	1	0	0	0	0	gram://gatekeeper.ranger.tacc.teragrid.org:2119/jobmanager-fork
tg-gram-trestles	up	1	1	1	2	2	gram://trestles.sdsc.edu:2119/jobmanager-fork
tg-gsissh-blacklight	up	1	1	0	0	0	gsissh://tg-login.blacklight.psc.teragrid.org
tg-gsissh-kraken	up	1	1	1	2	2	gsissh://kraken-gsi.nics.utk.edu
tg-gsissh-lonestar	up	1	1	1	2	2	gsissh://tg-login.lonestar.tacc.teragrid.org
tg-gsissh-ranger	up	1	1	0	0	0	gsissh://tg-login.ranger.tacc.teragrid.org/
tg-gsissh-trestles	up	1	1	1	2	2	gsissh://trestles-login.sdsc.edu/
total	-	30	16	14	30	24	-

(a) overview over participating backend states.



```
14 (fg-ssh-sierra) [ssh://sierra.futuregrid.org]-[...
13 (fg-ssh-india) [ssh://india.futuregrid.org]-[[f...
12 (fg-ssh-alamo) [ssh://alamo.futuregrid.org]-[[f...
11 (tg-gsissh-trestles) [ssh://trestles-login.sdsc...
10 (tg-gsissh-lonestar) [ssh://tg-login.lonestar.t...
9 (tg-gsissh-kraken) [ssh://kraken-gsi.nics.utk.ed...
8 (tg-gram-trestles) https://trestles-login2.sdsc....
7 (tg-gram-lonestar) https://gridftp1.ls4.tacc.ute...
6 (priv-ssh-local) [ssh://localhost]-[[fork://loca...
5 (priv-fork-local) [fork://localhost]-[5053]
4 (gin-unicore-juelich) [https://zam1161v01.zam.kf...
3 (gin-ssh-cct) [ssh://cyder.cct.lsu.edu]-[[fork:/...
2 (gin-arc-hungary) [https://interop.grid.niif.hu:...
1 (fg-ssh-sierra) [ssh://sierra.futuregrid.org]-[[...
0 (fg-ssh-india) [ssh://india.futuregrid.org]-[[fo...
missing
29 (gin-ssh-cct) [ssh://cyder.cct.lsu.edu]-[[fork:...
28 (gin-arc-hungary) [https://interop.grid.niif.hu...
27 (fg-ssh-sierra) [ssh://sierra.futuregrid.org]-[...
26 (fg-ssh-india) [ssh://india.futuregrid.org]-[[f...
25 (fg-ssh-alamo) [ssh://alamo.futuregrid.org]-[[f...
24 (tg-gsissh-trestles) [ssh://trestles-login.sdsc...
23 (tg-gsissh-lonestar) [ssh://tg-login.lonestar.t...
22 (tg-gsissh-kraken) [ssh://kraken-gsi.nics.utk.e...
21 (tg-gram-trestles) https://trestles-login2.sdsc...
20 (tg-gram-lonestar) https://gridftp1.ls4.tacc.ut...
19 (priv-ssh-local) [ssh://localhost]-[[fork://loc...
18 (priv-fork-local) [fork://localhost]-[5053]
17 (gin-unicore-juelich) [https://zam1161v01.zam.k...
16 (gin-ssh-cct) [ssh://cyder.cct.lsu.edu]-[[fork:...
15 (gin-arc-hungary) [https://interop.grid.niif.hu...
```

(b) resulting mandelbrot set.

Fig. 5: **Distributed Mandelbrot Demo**: the endpoints listed in table (a) contribute to the resulting mandelbrot set in figure (b). Shown is an experimental snapshot which shows a number of endpoints in various error and success states.

It must be noted that the greatest inhibitor of production level application runs on that scale is, at this point, not the engineering level – a combination of ALI and SLI is well able to overcome most of the technical

[2] `http://cyder.cct.lsu.edu/saga-interop/mandelbrot/demo-gin/today/last/`

obstacles. Instead, the security policies and accounting limitations for these efforts are what makes these experiments a continuous, and fragile, challenge.

The test results are fed back to the OGF standardization community, to either improve the respective standards, or to forward any implementation problems to the respective standards development groups.

Development of SAGA as an interoperable DCI Access Layer

The described P* research and implementation efforts, and also the underlying ALI efforts, are mostly based on SAGA as generic abstraction layer for programming distributed applications and frameworks. Our SAGA implementations are adaptor-based, i.e. they expose a generic API towards the application layer, whose semantics is provided by a set of adaptors which translate the API calls to the lower level system's capabilities. As with all adaptor based systems, such an implementation is only as useful and stable as its backend bindings. In particular, code stability is major concern, as SAGA requires integration with several middleware stacks, which in turn are based on a wide set of technologies, and are inherently complex.

FutureGrid has been used as a development and testing environment for SAGA, allowing to implement and harden a wide number of backend adaptors, in particular new ones for BES and EC2/AWS. The current move of FutureGrid toward (newer) OpenStack deployments will allow us to expand those efforts towards OCCI. At the same time, the early availability of these backend bindings in SAGA has enabled several students and researchers to perform experiments on FutureGrid systems which would have been harder to achieve otherwise, thus increasing the usefulness of FutureGrid to its users. Finally, the results of the SAGA implementation efforts are immediately usable for education and training efforts on FG, as described in 4.

Providing Interoperability for Distributed Cyber Infrastructures

Next to the implementation challenges for ALI oriented system stand the deployment challenges. In our experience, the efforts required to provide clean and stable deployment processes for a variety of environments can in fact well exceed the actual development efforts.

We find the FutureGrid approach, to act as an test and staging environment which explicitly targets a large production environment, XSEDE, as very appropriate: it provides us with the ability to develop, test and certify packaging and deployment procedures without interfering with production systems, and without exposing the evolution of that process to the end users on those systems.

In detail: SAGA is deployed as community software on TeraGrid/XSEDE. We are using the same model on FutureGrid – that provides SAGA installations to our (mostly experiment friendly) users on FutureGrid itself, and also perfectly mimics the procedures we use to deploy stable versions on XSEDE. Further, we have been using FutureGrid resources to perform certification and packaging processes toward other production infrastructures, most notably toward the SAGA enrollment in the European EGI software repositories (UMD).

4 Education: Distributed Scientific Computing

In Fall of 2010 a new class on Scientific Computing was co-introduced by Jha (led by Prof. Gabrielle Allen). The Distributed Scientific Computing component of this class was led and developed by Jha. The module (Fig. 6) covered the theory and practice of Distributed Scientific Computing, with an emphasis on production-grade distributed cyberinfrastructure.

There exist a broad range of computational infrastructure with varying support for application characteristics, usage modes and even access policies, all of which influence the usefulness of a given infrastructure for scientific applications. This module provided sufficient understanding of production grade distributed cyberinfrastructure to enable students to discern and determine which infrastructure would be appropriate and effective for their applications.

The module began by motivating the role and need for distributed computing by understanding six rather different distributed applications – and analyzing them for the following points: why distributed, how distributed and the challenges, success and/or issues in distributing them. Elements of data-intensive computing and the role of distributed data were also covered. After establishing the *essential* role of distributed computing in these applications, as well as understanding the critical role of *execution environments* for distributed computing, the students were exposed to a range of production distributed infrastructures and a brief overview of the design principles, objectives and application characteristics supported. Specific examples of production Grids – high-throughput as well as high-performance were analyzed.

To appreciate the reasons behind the rise of Cloud Computing, the module provided a basic overview of the important underlying trend in computing technology and data-intensive computing. This led to the broader domain of scientific computing as enabled on commercial infrastructure (Amazon EC2, Azure) as well as other non-commercial Cloud offerings

Practically, the students were given hands on experience with both virtualized resource and "bare-metal" resources thanks to the FutureGrid. Students mostly experimented with Eucalyptus on India and Sierra resources of the FutureGrid, as well as worked with a SAGA-enabled version of MapReduce.

	Lecture	Learning Objectives
E1	Introduction to the Practise of Distributed Computing - I	WLCG as a motivating example (order of magnitude estimates of number of jobs submitted, data transferred, CPU cycles consumed), Distributed Application Exemplars, Analyzing why and how distributed, and challenges & success in distribution. Introduction to SAGA and FutureGrid (FG).
E2	Introduction to the Practise of Distributed Computing - II	Examples of Production Grid Infrastructure - HPC vs HTC, Research vs Production, Commercial. Introduction to SAGA; Writing *your* first Dist. Application.
E3	Cloud Computing & Master-Worker Pattern	Cloud Computing. Convergence of multiple trends, Understanding Amazon Examples of M-W Pattern: SAGA-based MapReduce, Word Count Application and Mandelbrot Set.
E4	To Distributed or not to Distribute?	Case Studies, Observations on Distributed Applications, Development Objectives; Projects on FG.

Fig. 6: Module curricula for Distributed Scientific Computing

The Sapir-Whorf hypothesis implies that "language influences the (habitual) thought". The implication and analog in scientific computing is that the infrastructure used shapes the practise and formulation of research. Conversely, "for a given scientific application/research question, which cyberinfrastructure should I use?". Lecture E2 provided a brief overview and understanding of available infrastructures, and E4 covered the reasons and answers behind which infrastructure would be suitable for a broad range of applications.

5 Conclusion

About the Authors

Shantenu Jha is an Assistant Professor at Rutgers University and Assistant Research Professor at the Center for Computation & Technology at the Louisiana State University. He is also a member of the Graduate Faculty, School of Informatics, Edinburgh – Europe's top ranked CS department. He has more than 60 publications at the interface of Computer Science, Computational Science and Cyberinfrastructre Development. For select publications see: `http://www.cct.lsu.edu/~sjha/select_publications/` Jha is the PI of the SAGA (http://saga.cct.lsu.edu) Project and the UK-EPSRC & NSF co-funded 3DPAS (Distributed dynamical data-intensive programming abstractions and systems) research theme. He was the PI of the just concluded NSF-funded $2M Cybertools project. He is the lead author of a book to be published by Wiley in 2012 on "Abstractions for Distributed Applications and Systems: A Computational Science Perspective".

Ole Weidner is the technical project manager for the SAGA development efforts and a Ph.D. student in Informatics at the University of Edinburgh. Before joining CCT and the SAGA Team in 2006 he worked at the Max-Planck-Institute for Gravitational Physics in Potsdam as a software developer for the Grid Application Toolkit (JavaGAT).

Andre Merzky received his diploma in Particle Physics in 1998 at the Humboldt University in Berlin. He has worked since on Grid-related topics concerning data management and visualization, and is active in the Applications Area of the Open Grid Forum (OGF). Although located in far eastern Germany, he is funded by the Louisiana State University to continue his work on SAGA.

Andre Luckow is a consultant in the SAGA Group. He studied Computer Science at the Potsdam University where he obtained his doctorate degree in 2009. His main research interests are: distributed systems, fault tolerance, computational sciences and programming languages.

Mark Santcroos is based in Amsterdam, The Netherlands. Currently his main affiliation is the Academic Medical Center in Amsterdam, where his position is funded by BiG Grid, the Dutch NGI. His research interests are in data-intensive science and workflow systems on diverse and distributed computing infrastructures.

References

[1] Tom Goodale, Shantenu Jha, Hartmut Kaiser, Thilo Kielmann, Pascal Kleijer, Andre Merzky, John Shalf, and Christopher Smith. A Simple API for Grid Applications (SAGA). OGF Recommendation Document, GFD.90, Open Grid Forum, 2007.
`http://ogf.org/documents/GFD.90.pdf`.

[2] Shantenu Jha, Daniel S. Katz, Manish Parashar, Omer Rana, and Jon B. Weissman. Critical Perspectives on Large-Scale Distributed Applications and Production Grids (Best Paper Award). In *The 10th IEEE/ACM Conference on Grid Computing 2009*, pages 1–8, 2009.

[3] A. Luckow, L. Lacinski, and S. Jha. SAGA BigJob: An Extensible and Interoperable Pilot-Job Abstraction for Distributed Applications and Systems. In *The 10th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, pages 135–144, 2010.

[4] Andre Luckow, Mark Santcroos, Ole Weider, Andre Merzky, Sharath Maddineni, and Shantenu Jha. Towards a common model for pilot-jobs. In *Proceedings of The International ACM Symposium on High-Performance Parallel and Distributed Computing*, 2012.

[5] Andre Luckow, Mark Santcroos, Ole Weider, Andre Merzky, Pradeep Mantha, and Shantenu Jha. P*: A model for pilot abstractions. In *submitted to Supercomputing 2012*, 2012.

[6] Pradeep Mantha, Andre Luckow, and Shantenu Jha. Towards a common model for pilot-jobs. In *Proceedings of the third international workshop on MapReduce and its applications*, MapReduce '12, 2012.

[7] Pilot API. `http://github.com/saga-project/BigJob/tree/master/pilot/api`, 2012.

[8] Saurabh Sehgal, Miklos Erdelyi, Andre Merzky, and Shantenu Jha. Understanding Application-Level Interoperability: Scaling-Out MapReduce over High-Performance Grids and Clouds. *Future Generation Computer Systems*, 27(5):590 – 599, 2011.