FutureGrid 2012 Project Challenge: Project 45 Building Scalable, Dynamic and Distributed Applications Using SAGA

Pradeep Kumar Mantha *Sivakarthik Natesan* *Melissa Romanus*
Sai Saripalli *Ashley Zebrowski*
Faculty Advisor: Shantenu Jha

1 Introduction

There are multiple challenges in the effective design and implementations of scalable distributed applications and infrastructure: the spectrum of challenges range from managing the heterogeneity inherent in distributed systems on the one hand to the lack of well established programming models to support distributed applications. In addition there do not exist well defined set of base capabilities or unifying abstractions needed to reason about how, when and where to distribute applications. Against this backdrop, the range of distributed cyberinfrastructure (DCI) available to researchers is continually evolving. Thus, the process of designing and deploying large-scale DCI, as well as developing applications that can effectively utilize them, presents a critical and challenging agenda for domain researchers and CI developers alike.

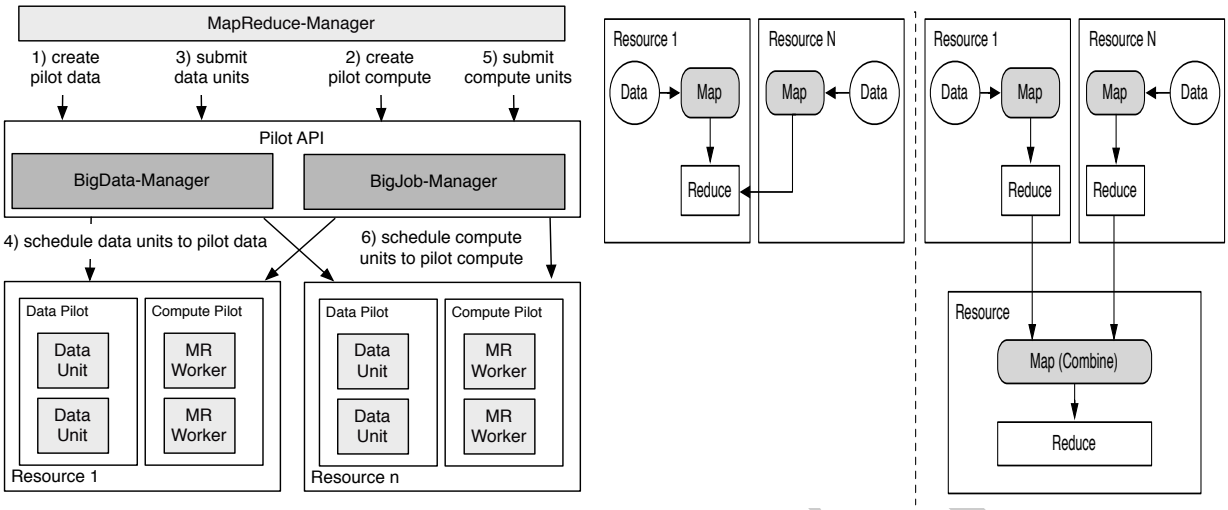
FutureGrid [8] provides students and researchers with new possibilities to engage in science relating to the state-of-the-art in cloud and grid computing. As student members of the Research in Distributed Cyberinfrastructure and Applications (RADICAL) group, we have taken full advantage of the opportunities that FutureGrid provides.

The students of the RADICAL group have been using SAGA on FutureGrid to address a wide spectrum of challenges: from scalable runtime systems for distributed data-intensive applications (Pilot-MapReduce) to novel dynamic execution modes for traditional HPC applications (Cactus-Spawner) as well as enhanced sampling algorithms (Replica-Exchange). In addition to flexible and scalable applications, we have used FutureGrid to enhance and extend the capabilities of SAGA. In this submission we outline how are some of the ways we are using SAGA on FutureGrid resources to build scalable production runtime systems and software whilst pushing the envelope by pursuing exciting new programming models and possibilities in application space.

2 Pilot-MapReduce

An increasing amount of data that scientific applications need to operate on is distributed. For example, the Earth Science Grid federates data of various climate simulations [6]. Meta-genomic workflows need to process and analyze data generated by various sequencing machines [9]; the localization onto a single resource is often not a possibility. Several options for running Hadoop on distributed data have been proposed [7]: (i) in a global Hadoop [5] setup one central JobTracker and HDFS NameNode is used for managing a distributed set of resources; (ii) in a hierarchical Hadoop setup multiple MapReduce clusters are used: a MapReduce cluster close to the data source for pre-processing data and a central cluster for aggregating the different de-central data sources.

Ref [7] shows that a hierarchical Hadoop configuration leads to a better performance than a global Hadoop cluster for high data aggregation applications like WordCount. A drawback of hierarchical Hadoop approach



(a) **Pilot-based MapReduce** (b) **Pilot-MapReduce Deployment Scenarios**

Fig. 1: Figure 1a shows the architecture of Pilot-MapReduce, Pilot-Jobs manage the map and reduce compute units and Pilot-data manage the intermediate data transfer between the map and reduce phase across multiple clusters. Figure 1b shows different Pilot-MapReduce Deployment Scenarios- In the distributed scenario (1b: left), the mapping tasks are run close to the data; reduced tasks are then run on a single resource. In the hierarchical scenario (1b:right) two full MapReduce runs are conducted.

is the increased complexity: Hadoop is not designed with respect to a federation of multiple MapReduce clusters. Setting up such a system typically requires a lot of manual effort.

To address these requirements, we design and implement Pilot-MapReduce (PMR) [11] - a flexible, infrastructure-independent runtime environment for MapReduce. PMR is based on Pilot abstractions for both compute (Pilot-Jobs) and data (Pilot-Data) [10]: it utilizes Pilot-Jobs to couple the map phase computation to the nearby source data, and Pilot-Data to move intermediate data using parallel data transfers to the reduce computation phase. Figure 1a depicts the architecture of PMR.

Pilot-MapReduce supports different distributed MapReduce topologies: (i) local, (ii) distributed and (iii) hierarchical. A local PMR performs all map and reduce computations on a single resource. Figure 1b shows options (ii) and (iii): A distributed PMR utilizes multiple resources often to run map tasks close to the data to avoid costly data transfers; the intermediate data is then moved to another resource for running the reduce tasks. [11]

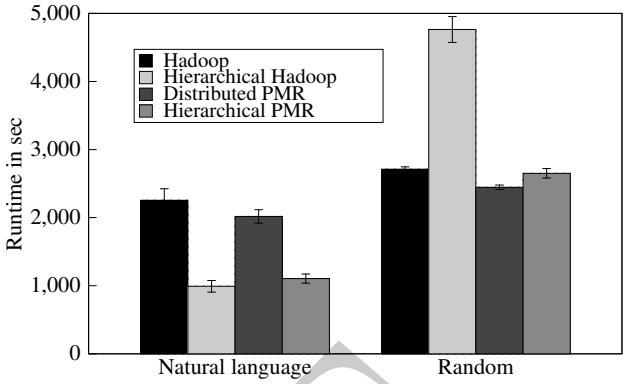
FutureGrid has been an important testbed for the development of Pilot abstractions and PMR. Our experimental evaluations on FutureGrid show that the *innovative* utilization of Pilot abstractions for compute and data treatment in MapReduce framework, across multiple clusters is promising, and can lower the execution time span of the entire MapReduce execution. To demonstrate the performance, extensibility and scalability of PMR, we utilized FutureGrid resources India, Sierra and Hotel. We benchmarked PMR against Hadoop using various MapReduce topologies and each experiment is repeated at least three times.

The *performance* and *scalability* of PMR with distributed data is compared to Hadoop MapReduce using canonical WordCount application on both natural language and random data using different MapReduce topologies. We utilize two machines, Sierra and Hotel. For all configurations, we use 8 nodes. Table 2a summarizes the characteristics of the WordCount application on natural language and random data.

The initial input data of 16 GB is equally distributed on these two machines. For the single resource Hadoop configuration, half of the input data needs to be moved from Sierra to Hotel prior to running the actual MapReduce job. Figure 2b shows the results. For natural language input, both Hadoop and PMR show comparable performance. A major determinant of performance for Hadoop (in the case of distributed

Data Stage	Natural Language	Random Data
Input	16 GB	16 GB
Inter-mediate	26 GB	30 GB
Output	20 MB	30 GB

(a)



(b)

Fig. 2: Table 2a shows the input, intermediate and output data volumes for WordCount applications on natural language and random data. Figure 2b shows WordCount on 16 GB Data Using Hadoop, Hierarchical Hadoop, Distributed PMR and Hierarchical PMR

data) is the necessity to move parts of the data (half of the input data) to the central Hadoop cluster. The performance of PMR is determined by the runtime of the map and reduce phase, which are slightly longer than for Hadoop mainly due to the resource heterogeneity and the resulting scheduling overhead: the slowest node determines the overall runtime of both the map and reduce phase.

Both hierarchical Hadoop and PMR perform better than the distributed PMR and single resource Hadoop configuration. The performance is mainly influenced by data movement costs. In the distributed PMR scenario, half of the *intermediate* data needs to be moved to the other resource; in the hierarchical case half of the *output* data requires movement. Since the output data in the hierarchical case is a magnitude smaller than the intermediate data in the distributed case (comparison. table 2a) – 20 MB in comparison. to 30 GB – the performance in the hierarchical case is significantly better.

For random data, the distributed PMR and single resource Hadoop perform better than the hierarchical PMR and hierarchical Hadoop configuration. As the output data is approximately equal to the intermediate data (30 GB), i. e. the advantage of a reduced transfer volume does not exit. For random data, the additional MapReduce run represents an overhead. In the Hadoop case, the moved data needs to be loaded into HDFS, which represents another overhead.

Pilot-MapReduce provides a flexible runtime environment for MapReduce applications on general-purpose distributed infrastructures, such as XSEDE [12] and FutureGrid. It brings the advantages of the Pilot abstraction to MapReduce, and enables utilization of federated and heterogeneous compute and data resources. In contrast to Hadoop, no previous cluster setup, which includes running several Hadoop/HDFS daemons, is required. Pilot-MapReduce provides a extensible runtime environment, which allows the flexible usage of sorting in the shuffle, more fine-grained control of data localities and transfer, as well as support for different MapReduce topologies. Using these capabilities, applications with different characteristics, e.g. compute/IO and data aggregation ratios, can be efficiently supported.

3 Replica Exchange

Replica-Exchange (RE) methods represent a class of algorithms that involve a multiple instances of almost similar copies of a system (i.e. replica), which used to understand physical phenomena ranging from protein folding dynamics to binding affinity calculations. These replicas are loosely-coupled with each other, i.e., replicas exchange small amounts of *state* information infrequently. RE algorithms can be formulated as either synchronous or asynchronous. In principle the asynchronous RE algorithm should have better performance in the presence of heterogeneous and distributed resources. However, this has not been established before;

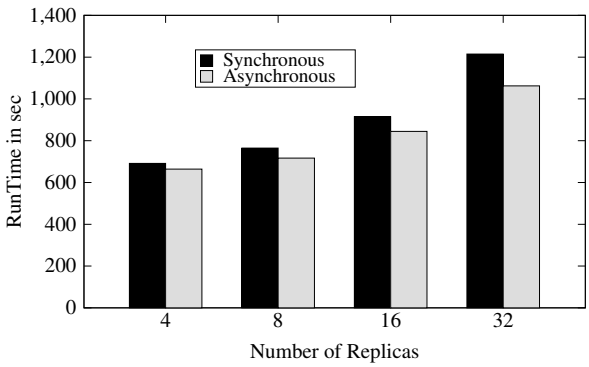


Fig. 3: Performance of Centralized Replica-Exchange Mechanism: Synchronous vs Asynchronous

the aim of our RE work was to determine if this indeed was the case and to analyze the performance of the different algorithmic formulations.

Experiments and Results: We have used FutureGrid clusters namely: Sierra, India, Alamo and Hotel for RE experiments involving up to 128 cores on each of these FutureGrid clusters. We evaluate and compare the scaling performance of the two algorithms as the number of in conjunction with our framework. To understand the scaling behavior of the different RE formulations, we analyze total time to completion (T) for different replica counts from 4, 8, 16, 32. We have kept the ratio of number of attempted exchanges to the number of replicas as constant. The number of times each replica is restarted to complete all exchanges remains constant. Hence the comparison between different cases will reveal differences in coordination cost. The increase is however not uniform between the two implementations: it is largest for synchronous RE than Asynchronous RE. We analyzed the total run time on FutureGrid machines and values for all the different replica counts can be seen in the following graph

In Fig. 3: As we scale the number of replicas over multiple machines, keeping number of replicas to number of exchange ratio constant, we observe that Asynchronous RE outperforms Synchronous RE. This is due to the fact that the synchronization cost (waiting times for all replicas to reach a state where each is ready for an exchange attempt, i.e., global-exchange) is higher in synchronous RE; in contrast, there are no global exchanges in asynchronous RE. Results from RE experiments on multiple FutureGrid machine therefore demonstrate relative scalability of the different algorithms; the SAGA-based Pilot-Job – BigJob, is used to manage the dynamic execution of these replicas, thus further establishing the power of abstractions in supporting interoperability.

4 Cactus Spawner

The Cactus Spawner project envisions application frameworks with simulations that can be broken down to their constituent components and run across multiple distributed systems. A chief consideration is that of "intelligent computing", of knowing when and where to run simulation components separately from the main simulation. Work is being done to enable this by modeling simulation components and predicting the time to run locally vs. the time to transport them and execute them remotely. To model real-world problems, the Cactus framework is used, and actual black hole simulations are executed and spawned from. Contributions to the field involve algorithms in modeling, spawning, and performance evaluation on the I/O systems of FutureGrid hardware. Of special note is the role SAGA plays in the Spawner project; SAGA allows multiple middleware systems to be specified without the need for extensive code rewriting to support each middleware.

FutureGrid resources were key to the initial development of the Spawner technology. HPC resources *Alamo* and *Hotel* were used to develop the methods of separating functions from the main simulation. Details of the Spawner mechanism developed on FutureGrid can be seen in figure 4. The large scale needed

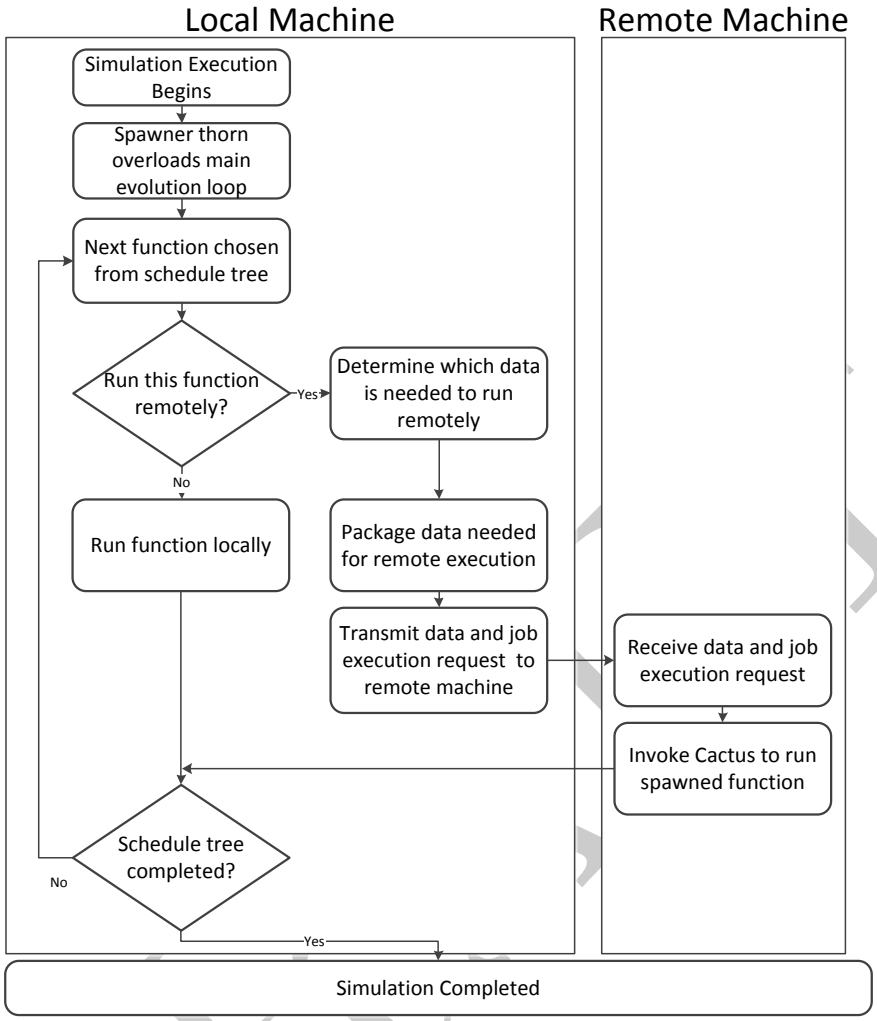


Fig. 4: Spawner decision-making logic

to run the spawning experiments precluded the use of FutureGrid hardware for the main simulation runs. For this reason, simulation results were gathered on XSEDE machines. The base research behind the Spawner technology was still conducted on FutureGrid, including the initial coding and testing.

An increase in scalability may be realized by executing nondependent functions in parallel across multiple machines rather than in serial on one. An improvement in interoperability may be realized by making use of heterogeneous systems, enabling simulation functions to run on the machines which suit them best. This is naturally improved through the use of SAGA, which extends the range of systems which may be utilized.

The eventual goal of the Spawner project is to, (i) provide a basis for an increase in scalability and interoperability for future simulations, and (ii) to serve as the basis for building "intelligent" frameworks that support multi-level and multi-dimensional (such as compute and data) scheduling. With SAGA and the selective execution technology behind Spawner, scalability across and interoperability between systems, as well as extensible capabilities to different applications can be enhanced.

5 SAGA-Bliss Development

SAGA-Bliss [**?**] is an implementation of the OGF GFD.90 SAGA standard, built with an emphasis on usability and ease of deployment. In short, the focus of the SAGA-Bliss implementation is to have a highly-usable API for distributed computing with a low barrier of entry for application development. The SAGA-Bliss code base is still under heavy initial development, and FutureGrid was found to be an ideal location for development and testing.

Two key features for SAGA-Bliss were developed while making use of FutureGrid resources; the SSH and Eucalyptus cloud plugins. The SSH plugin enables the use of the SSH protocol to submit jobs to remote resources, utilizing SAGA abstractions such as contexts in order to specify authentication information. The Eucalyptus plugin allows users to access Eucalyptus compute clouds, starting up, shutting down, and querying virtual machines. Adding these additional plugins to SAGA-Bliss contributes greatly to the interoperability of SAGA-Bliss, with each new plugin providing a means of communicating with a new resource.

Initial results on FutureGrid's *India* and *Sierra* indicate that the SSH plugin works and is useful for cross-machine job submission, and indeed the SSH plugin passes all SAGA-Bliss job plugin test cases. The Eucalyptus plugin is successful starting, stopping, and querying virtual machines. To test the functionality of the plugins together, a scenario of cloud resources running Cactus simulations was developed. From within SAGA-Bliss, first a virtual machine is booted, then a Cactus job is sent to the virtual machine and successfully executed, the remote machine terminating once execution has completed. This proves that the SAGA-Bliss approach maps well to handling cloud compute resources, and that scientific simulations can be executed successfully from this base. Continued development of SAGA-Bliss will increase the degree of interoperability that SAGA-Bliss offers between distributed resources, with additional plugins offering additional methods of communicating with resources in much the same way the SSH and Eucalyptus plugins offer access to SSH servers and Eucalyptus compute clouds.

6 Production Infrastructure: Deployment and Testing

SAGA and BigJob are deployed [1] on all major FutureGrid machines (India, Sierra, Hotel and Alamo). FutureGrid provides Community Software Areas (CSA) to communities, which basically is a system level area where software packages can be shared among a community of users. While it is possible to install and use SAGA and BigJob in each user's directory, to ensure minimum repetition of effort and to ensure users of SAGA get the best possible support, we employ the CSA model. It is not the case that SAGA is difficult to install, but the fact remains that the adaptors to many middlewares have widely varying configuration and deployment requirements. Thus having a central location on each machine where SAGA is deployed as a community code ensures effective and correct installation. Similar considerations hold for BigJob and SAGA-Bliss – it was therefore only logical to deploy SAGA and BigJob in a CSA space on FutureGrid machines.

The source code for SAGA, BigJob and SAGA-Bliss is stored in a git repository [2], and the deployment scripts pull sources from specific branches in the repository. A set of semi-automatic deployment scripts is used to manually trigger CSA software updates. An update can encompass an individual machine or all machines, an individual SAGA component or the entire SAGA/BigJob/SAGA-Bliss stack. The set of supported components includes the SAGA core libraries, different API packages, the supported middleware adaptors for each machine, the python bindings, and the BigJob package and its dependencies, and SAGA-Bliss.

The CSA installations are accompanied by automatically generated documentation (READMEs) and configuration scripts, which simplify the environment settings for the end user. Site specific configuration details are also documented on github wiki pages. For BigJob, a github wiki is used to store user guides for each of the individual FutureGrid machines.

A BigJob CSA release is the result of a testing and deployment pipeline. Any newly developed features, code modifications and bug fixes are created in branches and are later merged onto the master. Once

merged, it is deployed as experimental feature. After each update, a set of manual tests is performed to ensure the validity of the deployment candidate. After testing is complete, the python code is (manually) pushed to the Python Package Index (*pypi*). This code is then deployed into the CSA space using pypi. Careful consideration is taken to ensure that the updates to CSA space will not impact any current users of BigJob on the target resources. Another round of testing is then completed to verify that the CSA installations are working and no changes to the users' environments are required. The deployment documentation (README), a generated 'module' file and basic test results are all committed to the central SAGA code repository [3], and (partially) used to document the current state of deployment on the SAGA deployment wiki. Any corresponding documentation on the BigJob github wiki is updated to reflect the changes.

For SAGA and for SAGA-Bliss, an additional set of test is run once the CSA deployment completes, to verify the viability of the installations. For SAGA, those tests are relatively simple, and check if the expected set of adaptors are available and load correctly. For SAGA-Bliss, the set of tests is more elaborate, and also cover correct environment settings, and n:m testing of remote system access: for example, a test run for the SAGA-Bliss installation on India would attempt to submit test jobs via ssh and pbs+ssh to Alamo, Sierra and Hotel. As those tests use exactly the environment settings which are documented for the end user, we gain some confidence that the installation is in fact viable. Test results are timestamped and again pushed back into the SAGA git repository.

Additionally, SAGA-Bliss is using another buildbot [4] driven test regime to ensure the stability of the SAGA-Bliss code base in respect to FutureGrid and XSEDE as target environments: on every git code update (and also at regular intervals), a set of tests is run on all FutureGrid backends, to verify that the SAGA-Bliss master branch is in usable state, and to notify the developers of potential problems as early as possible. That mechanism has proven extremely valuable for the release process of SAGA-Bliss.

7 SAGA and BigJob on FutureGrid Cloud Resources

Though SAGA was originally developed for Grids and clusters, its importance must also be considered in the context of Cloud computing. SAGA for Clouds provides a promising route for cloud-cloud and cloud-grid interoperability, in addition to providing a useful and consistent set of abstractions for compute and data management across grids and clouds.

SAGA on clouds has been used to provide "distributed functionality" for data-intensive applications. Grids and clouds differ in many semantical aspects, such as service-based vs. direct resource access, what scalability entails, and differences in platforms. SAGA requires functionality that allows the application to be immune to the fact that it is running on a grid vs. a cloud. This functionality is achieved through a context-aware adaptor that is dynamically loaded. Interoperability at the application-level must be provided through the use of such adaptors. In order to facilitate the use of SAGA on clouds, we have developed an Eucalyptus adaptor to spawn the virtual machines. The ssh adaptor was used to submit jobs to the actual machines.

A natural extension of this work is to use BigJob – the SAGA-based Pilot Job Runtime system that supports dynamic execution, with clouds. BigJob has been shown in the past to work with both the Windows Azure Cloud operating system and Nimbus platform. On FutureGrid, cloud capabilities are offered through a choice of IaaS platforms – Eucalyptus, Nimbus, or OpenStack. Testing of SAGA and BigJob on cloud resources was performed on India with both Eucalyptus and OpenStack. Though SAGA had been tested with both Eucalyptus and Nimbus before, it had yet to be tested on the new OpenStack resources on India. India offers both OpenStack Nova (compute resources) and Swift (storage). Preliminary work on integrating SAGA with these resources focuses only on Nova and job submission, but in the future, the focus will shift to cloud-based storage to support both scalability and data exchanges.

8 Conclusion

In this project report, we have discussed how right abstractions for runtime systems can result in scalable, extensible and interoperable distributed applications and systems. SAGA is an OGF standard and attempts to

provide generic programming abstractions and interoperable application frameworks to simplify and increase the utilization of DCIs. FutureGrid provides wide variety of traditional grid clusters of different architectures and clouds like OpenStack, Nimbus, Eucalyptus. These FutureGrid infrastructures play an important role for SAGA group in providing and validating interoperable frameworks, on a variety of infrastructure types. These frameworks are used to develop important software that support programming models and novel and optimized algorithms.

For example, Pilot-MapReduce is designed and implemented to provide a *interoperable*, *scalable* MapReduce framework on distributed cyberinfrastructures. PMR has been entirely developed and validated using FutureGrid. Pilot-MapReduce provides flexibility to scale MapReduce applications to multiple infrastructures thus coupling FutureGrid resources with other infrastructure resources like XSEDE, OSG and EGI. The *insightful performance comparison* between PMR and Hadoop MR configurations, shows that PMR is a promising distributed MapReduce framework and can lower the execution time span of entire MapReduce application. Furthermore, we developed and implemented an *innovative* and completely *new algorithm* to perfrom distributed asynchronous decentralized Replica-Exchange and validated it on FutureGrid, LONI and XSEDE resources.

SAGA-Bliss is a light weight purely python implementation of SAGA, built with an emphasis on usability and ease of deployment. Initial developments of SAGA-Bliss on FutureGrid involves development of job submission plugins to both Grids and Clouds of FutureGrid. The plugins were validated using Cactus simulations on both FutureGrid Eucalytpus cloud and Grid resources thus demonstrating Grid and cloud *interoperability*. Developments are still in progress to support OpenStack and Nimbus clouds to demonstrate cloud-cloud and cloud-grid interoperability at large scale utilizing all possible FutureGrid resources. To enable the development of the *new software* like various SAGA-Bliss job plugins and usage of applications based on SAGA and SAGA-Bliss, SAGA and SAGA-Bliss are made available to all users through CSA installation. These SAGA based CSA installations are validated via a automated test-suite, which can be triggered via Bliss buildbot and manually.

Publications were submitted to MapReduce, ECMLS (held in conjunction with HPDC) and a SC12 workshops and described the importance of FutureGrid and key role as an infrastructure for our research. We believe the results as captured in this report, demonstrates the use of Futuregrid for innovation in research, a high quality of papers, development of new software, for algorithmics and insightful performance measurements.

9 About Authors

Pradeep Mantha earned a Bachelors in Technology from JNTU University, India. He is curently working on a Master's in Computer Science at Louisiana State University. His research interests are in computational science, high performance & distributed computing. His research involves enhancing cyber infrastructure tools to solve computational and data intensive scientific applications.

Sai Saripalli is currently doing his Masters of Computer Science in Louisiana State University. His research interests are distributed and cloud computing.

Sivakarthik Natesan is currently pursuing Masters of Computer Science in Louisiana State University. His research interests are high performance distributed and data intensive computing.

Melissa Romanus is a Master's student studying computer engineering at Rutgers University. She received her undergraduate degree from Tufts University in 2006. Her interests are in distributed computing, data-driven science, and cloud-grid interoperability.

Ashley Zebrowski is a PhD student at Rutgers University, where she is an awardee of a Presidential Scholarship. Her interests are in High-Performance and Distributed Computing, Algorithmics and Computational Science.

Shantenu Jha is an Assistant Professor of Computer Engineering at Rutgers University and Assistant Research Professor at the Center for Computation & Technology at the Louisiana State University. He is lucky to be the Faculty advisor of incredibly talented, motivated and eager Graduate students, and lives off their generosity in allowing his name to be on publications arising from their hard efforts.

References

[1] https://github.com/saga-project/saga-deployments.

[2] http://www.github.org/saga-project/.

[3] https://github.com/saga-project/saga-deployments/tree/master/test.

[4] http://faust.cct.lsu.edu:8080/waterfall.

[5] Apache Hadoop. `http://hadoop.apache.org/`, 2012.

[6] D. Bernholdt, S. Bharathi, and D. Brown et al. The earth system grid. *Proc. of the IEEE*, 93(3):485–495, 2005.

[7] Michael Cardosa, Chenyu Wang, Anshuman Nangia, Abhishek Chandra, and Jon Weissman. Exploring mapreduce efficiency with highly-distributed data. In *Proceedings of 2nd international workshop on MapReduce and its applications*, pages 27–34, New York, NY, USA, 2011. ACM.

[8] FutureGrid: An Experimental, High-Performance Grid Test-bed. `https://portal.futuregrid.org/`, 2012.

[9] Shantenu Jha, J. D. Blower, Neil Chue Hong, Simon Dobson, Daniel S. Katz, Andre Luckow, Omer Rana, Yogesh Simmhan, and Jon Weissman. 3dpas: Distributed dynamic data-intensive programming abstractions and systems. 2011.

[10] 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.

[11] Pradeep Mantha, Andre Luckow, and Shantenu Jha. Pilot-mapreduce: An extensible and flexible mapreduce implementation for distributed data. In *Proceedings of the third international workshop on MapReduce and its applications*, MapReduce '12, 2012.

[12] XSEDE: Extreme Science and Engineering Discovery Environment. `https://www.xsede.org/`, 2012.