Federated Computing for the Masses – Aggregating Resources to Tackle Large-scale Engineering Problems

Javier Diaz-Montes
Rutgers University

Yu Xie
Iowa State University

Ivan Rodero
Rutgers University

Jaroslaw Zola
Rutgers University

Baskar Ganapathysubramanian
Iowa State University, baskarg@iastate.edu

Follow this and additional works at: http://lib.dr.iastate.edu/me_pubs

Part of the Acoustics, Dynamics, and Controls Commons, and the Systems and Communications Commons

The complete bibliographic information for this item can be found at http://lib.dr.iastate.edu/me_pubs/225. For information on how to cite this item, please visit http://lib.dr.iastate.edu/howtocite.html.
Authors
Javier Diaz-Montes, Yu Xie, Ivan Rodero, Jaroslaw Zola, Baskar Ganapathysubramanian, and Manish Parashar

This article is available at Iowa State University Digital Repository: http://lib.dr.iastate.edu/me_pubs/225
Federated Computing for the Masses – Aggregating Resources to Tackle Large-scale Engineering Problems

Javier Diaz-Montes*, Yu Xie**, Ivan Rodero*, Jaroslaw Zola*, Baskar Ganapathysubramanian**, and Manish Parashar*

*Rutgers Discovery Informatics Institute, Rutgers University
**Department of Mechanical Engineering, Iowa State University

Abstract

The complexity of many problems in science and engineering requires computational capacity exceeding what average user can expect from a single computational center. While many of these problems can be viewed as a set of independent tasks, their collective complexity easily requires millions core-hours on any state-of-the-art HPC resource, and throughput that cannot be sustained by a single multi-user queuing system. In this paper we explore the use of aggregated HPC resources to solve large-scale engineering problems. We show it is possible to build a computational federation that is easy to use by end-users, and is elastic, resilient and scalable. We argue that the fusion of federated computing and real-life engineering problems can be brought to average user if relevant middleware is provided. We report on the use of federation of 10 distributed heterogeneous HPC resources to perform a large-scale interrogation of the parameter space in the microscale fluid flow problem.

Keywords: Federated computing, cloud computing, software-defined infrastructure, large-scale engineering problems, fluid flow

1 Introduction

The ever-growing complexity of scientific and engineering problems continues to pose new requirements and challenges for computing and data management. The analysis of high-dimensional parameter spaces, uncertainty quantification by stochastic sampling, or statistical significance assessment through resampling, are just few examples of a broad class of problems that are becoming increasingly important in a wide range of application domains. These “ensemble” (also termed as many task computing or MTC) applications consist of a set of heterogeneous computationally intensive, and independent or loosely coupled tasks, and can easily consume millions of core-hours on any state-of-the-art HPC resource. While many of these problems are conveniently parallel, their collective complexity exceeds computational time and throughput that average user can obtain from a single computational center. For instance, the fluid flow problem considered in this article comprises more than ten thousand MPI tasks, and would require approximately 1.5 million core-hours to solve on the Stampede cluster at TACC – one of the most powerful machines within XSEDE [14]. Although XSEDE allocations of that size are not uncommon, the heavy utilization of Stampede, and its typical queue waiting times make it virtually impossible to execute that number
of tasks within an acceptable time limit. The problem becomes even more complex if we take into account that individual tasks are heterogeneous, and add in the possibility of failures that are not uncommon in large-scale multi-user systems.

The above constraints are not unique to one particular problem or a system. Rather, they represent common obstacles that limit the scale of problems that can be considered by an ordinary researcher on a single, even very powerful, system. What is important is that this trend continues and one can only expect that more and more users will require computational throughput that cannot be delivered just by one resource. In order to overcome these limitations two important questions have to be addressed. First, how to empower a researcher with computational capability that is compatible to what currently is reserved for the “elite” problems. Second, how to deliver this capability in a user-centered way. In this article, we argue that both these questions can be answered by implementing a software-defined federation model in which a user, without any special privileges, can seamlessly aggregate multiple, globally distributed and heterogeneous HPC resources exploiting their intrinsic capabilities. In this vision, a user is presented with programmable mechanisms to define resources as well as policies and constraints that autonomously guard how resources are used, and how to react to changes in the federation.

Federated computing has been explored in various contexts and has been demonstrated as an attractive and viable model for effectively harnessing the power offered by distributed resources [1, 3, 5, 7, 8]. For example, volunteer computing systems (e.g., BOINC) enable end-user resources, provided by a crowd of volunteers, to be aggregated to obtain non-trivial computing capabilities towards an application. While this model is easy to configure and use from a user perspective, it can support only a limited class of applications, i.e. those with large numbers of small and independent tasks. At the other end of the spectrum, Grids (e.g. EGEE [11], Grid’5000 [12], OSG [13]) have targeted more compute/data-intensive applications by federating capacity and/or capabilities into secure and dependable virtual organizations. Grids often have user-perceived complexity, and configuring them involves complex software-hardware interaction requiring significant experience from the end-users [7]. More recently, cloud federations are being explored as means to extend as-a-service models to virtualized data-centers federations. Given increasing importance of ensemble applications, and their computational requirements, it is becoming important to revisit user-centered federated computing from the perspective of this class of applications and their requirements.

In this article, we explore the use of software-defined federated computing to solve large-scale engineering problems in a user-centered way. Our focus is on empowering average user with aggregated computational capabilities typically reserved for selected high-profile problems. To achieve this, we propose to aggregate heterogeneous HPC resources in the spirit of how volunteer computing assembles desktop computers. Specifically, we describe a model of computational federation that i) allows users to be in control of the federation process by specifying resources accessible to them, the constraints associated to them, and how they want to make use of them as part of their federations, ii) is extremely easy to deploy and offers an intuitive API to meet expectations and needs of average user; iii) encapsulates cloud-like capabilities, e.g. on-demand resource provisioning, elasticity and resilience, to provide sustainable computational throughput; iv) provides strong fault-tolerance guarantees through constant monitoring of tasks and resources; v) bridges multiple, highly heterogeneous resources, e.g. servers, clusters, supercomputers and clouds, to effectively exploit their intrinsic capabilities; and vi) leverages security and authentication from underlying infrastructure. To demonstrate potential of the resulting federated infrastructure to address the
computational requirements of real-world large-scale computational engineering problems, we im-
plemented a prototype of the federation and used it to analyze a high-dimensional parameter space
in the fluid flow problem. The presented case-study involves a federation of 10 different and dis-
tributed HPC centers, consumes over 2.5 million core-hours, and provides the most comprehensive
data to-date on the effect of pillars on flow in microchannels.

2 Defining a Federation Model for the Masses

Our goal is to develop a federation model that would be able to support large scientific workloads,
but at the same time would be user-centered. To build such a model it is imperative to under-
stand two key elements. First, which specific properties of large-scale scientific and engineering
applications must be taken into consideration to enable efficient execution in a large federated
environment? Second, what kind of expectations must be addressed in order to achieve a user-
centered design? Because it would be unrealistic to assume that all types of scientific applications
can benefit from federated computing, we focus on a particular class of computational workloads
in which large search-spaces are investigated in a coordinated manner. Here, classic examples are
Monte Carlo methods, stochastic sampling strategies (e.g. sparse grid collocation), or soft comput-
ing approaches (e.g. simulated annealing). These techniques constitute a significant fraction of all
scientific codes in use today, and hence are of great practical importance to average user.

2.1 Large-scale Scientific and Engineering Applications in the Target

A typical approach to investigate large search-spaces combines two elements: a master module that
encapsulates a problem logic, e.g., to decide how the search-space should be navigated through,
and a science-driver that implements the actual computational core. Usually, both elements are
contained within separate software components, and problem logic can be implemented indirectly
in the execution environment (e.g. as a script interacting with a queuing system). Individual
instances of a science-driver are either independent or involve asynchronous communication. Nat-
urally, complexity of both modules may vary drastically. However, in the vast majority of cases
it is the science-driver that is represented by a complex parallel code, and requires HPC resources
to execute. For instance, in the case study that we describe in the next section, the problem logic
amounts to a simple enumeration of selected points in the search space, while the science-driver is a
complex fluid flow simulation. Although a science-driver is computationally challenging on its own,
the actual complexity comes from the fact that the usual investigation involves large number of
tasks (e.g. more than 12,000 in our study, millions in any Monte Carlo analysis). Oftentimes a single
resource is insufficient to execute the resulting workload either because of insufficient throughput
or limited computational capability. Additionally, tasks might be heterogeneous and have diverse
hardware requirements, or can be optimized for specific architectures. Moreover, except of sim-
ple scenarios, tasks are generated dynamically, based on partial or complete results delivered by
previously completed tasks.

2.2 Shaping the Federation from the Scientist Perspective

We focus now on what kind of user expectations must be addressed in order to achieve a user-
centered design. Here, we have to keep in mind that our federation model must serve a regular
user with a need for large computational capacity. Such a user most likely has access to several
heterogeneous resources using a standard environment, for example, a shell account. Consequently, the key feature that must be offered by a federation is the ability to aggregate heterogeneous resources while operating completely in a user-space. After all, it is unrealistic to expect that a user will have an administrative privileges on any HPC resource. Another important factor is how federated resources are exposed to a user. The federation should hide low-level details, such as geographic location or hardware architecture, while offering a familiar programming interface, for example, supporting common parallel programming idioms like master/worker or MapReduce, which could be used directly by a user. At the same time, a user must be able to deploy existing applications, i.e. science-drivers and sometimes a problem logic module, within the federation and without modifications.

If we look at the above characteristic it becomes apparent that there are several key features that our federation model must account for. The federation must be elastic and scalable – the ability to scale up/down and out becomes essential to handle varying over time number of tasks. What is important is that elasticity also makes the infrastructure resilient and hence improves its ability to sustain computational throughput. The federation must be able to adapt to the diverse task requirements, and make optimal use of distinct features contributed by the heterogeneous federated resources. Consequently, capability, which we define as the ability of a federation to take advantage of particular hardware characteristics, must be the first-class citizen in our model. This requirement is synergistic with the concept of autonomic computing.

3 A User-centered Approach to Federation

To deliver a federation model with properties highlighted in the previous section we focused on usability, elasticity and resilience as primary objectives. The presented model is aimed to allow the creation of software-defined federated infrastructures where resources are exposed using elastic on-demand cloud abstractions. In particular, we envision “living” federations that can dynamically evolve in terms of size and capabilities following user-defined constraints and instructions. The underlying infrastructure is presented as a single elastic pool of resources regardless of their physical location. The design is based on four layers, where the lowest layer is responsible for the interaction with physical resources, and the highest one is the actual user application. The appropriate provisioning of resources in accordance with user provided policies is realized by the cross-layer autonomic manager. The schematic representation of the design is presented in Figure 1.

This design allows us to separate the different functionality required by the federation. At the bottom we have a federation overlay, which creates a uniform view on top of physical resources and the foundation that supports the higher level services of the federation. It allows users to add and remove heterogeneous resources dynamically, and handles network and resource failures. This layer also provides a routing engine to address resources using their attributes rather than specific addresses, and supports flexible content-aware routing, and complex querying using partial keywords, wildcards, or ranges. Next, the service layer provides a range of services to support autonomies at the programming and application level. It includes a coordination service that handles the execution of applications, a discovery service to find resources based on their properties, and an associative object store service to manage tasks and data. These services are encapsulated and offered to the users through the programming layer, which provides the basic framework for application development and management. This layer supports several common distributed programming
Figure 1: Multi-layer design of the proposed federation model. Here, the autonomic manager is a cross-layer component that based on user data and policies provisions appropriate resources.

paradigms, including the master/worker, workflow, and MapReduce. These programming abstractions ease the development of applications by decoupling the application from the particularities of the infrastructure. Moreover, they affect the way the application is executed in the resources. Finally, the application layer represents the final application developed by a user on top of the programming layer. In many cases a user might be interested in benefiting from the federation to execute third-party, perhaps closed-source, software. In such cases the target software cannot or should not be modified, for example due to efficiency considerations. To accommodate for this, the programming layer can still be used in the standard way, however, the resulting application becomes a mere container that acts as a facade for the target software. This tremendously simplifies migration from traditional environments to our federation model. We should keep in mind however, that in this scenario the target application must be deployed on the federated resources beforehand.

The key ingredient of the federation is the autonomic manager. The manager enables the autonomic management and multi-objective optimization (including performance, energy, cost, and reliability criteria) of application execution through cross-layer application/infrastructure adaptations. This component offers QoS by adapting the provisioned resources to the application’s behavior as well as system configuration, which can change at run time, using the notion of elasticity at the application level. As a result, the federated infrastructure increases the opportunities to provision appropriate resources for given application based on user objectives or policies, and different resource classes can be mixed to achieve the user objectives. The manager can scale federation up/down/out based on the dynamic workload and provided user policies. For example, a user objective can be to accelerate the application execution within a given budget constraints, to complete the application within assumed deadline, or to use resources best matching to the application type (e.g., computation vs. data intensive). Because application requirements and resource status may change, for example, due to workload surges, system failures or emergency system maintenance, the manager provisions resources adaptively to accommodate for these changes. Note, that the adaption ensures implicitly resilience of the federation.

The security and authentication are leveraged solutions provided by each site (e.g., based on
This decision is motivated by the difficulties that the grid computing community found when trying to introduce a new authentication/authorization model, based on x.509 certificates, in the existing production infrastructures. Consequently, the federation allows users to select their preferred authentication/authorization mechanisms among those directly supported by each site.

4 Case Study

In order to demonstrate applicability and scalability of our federation model in the real-life scenario, we focused on the problem of constructing the phase diagram of fluid flow in microscale devices. The problem is highly representative for a broad category of parameter space interrogation techniques, which are essential for understanding how process variables affect behavior of the modeled system, to quantify uncertainty of the model when input data is incomplete or noisy, or to establish a ground on which inverse problems can be investigated. While these techniques are very diverse, typically they involve a large collection of computationally intensive tasks, with little or no synchronization between the tasks.

4.1 Application Description

Our focus on the fluid flow problem is motivated by its great practical importance. The ability to control fluid streams at microscale has significant applications in many domains, including biological processing [9], guiding chemical reactions [4], and creating structured materials [6]. Two of the authors, henceforth referred to as the end-user, are part of a team that recently discovered that placing pillars of different dimensions, and at different offsets, allows “sculpting” the fluid flow in microchannels [2]. The design and placement of sequences of pillars allows a phenomenal degree of flexibility to program the flow for various bio-medical and manufacturing applications. However, to achieve such a control it is necessary to understand how flow is affected by different input parameters.

The end-user has developed a parallel, finite element and MPI-based Navier-Stokes equation solver, which can be used to simulate flows in a microchannel with an embedded pillar obstacle. Here, the microchannel with the pillar is a building block that implements a fluid transformation. For a given combination of microchannel height, pillar location and diameter, and Reynolds number (4 variables), the solver captures both qualitative and quantitative characteristics of flow (see Figure 2). In order to reveal how the input parameters interplay, and how they impact flow, the end-user seeks to construct a phase diagram of possible flow behaviors. In addition, the end-user would like to create a library of single pillar transformations to enable analysis of sequences of pillars. This amounts to interrogating the resulting 4D parameter space, in which a single point is equivalent to a parallel Navier-Stokes simulation with a specific configuration.

The problem is challenging for several reasons. The search space consists of tens of thousands of points, and an individual simulation may take hundreds of core-hours, even when executed on a state-of-the-art HPC cluster. For example, the specific instance we consider requires 12,400 simulations. The individual tasks, although independent, are highly heterogeneous and their cost of execution is very difficult to estimate a priori, owing to varying resolution and mesh density required for different configurations. In our case, the cost may range from 100 core-hours to 100,000 core-hours per task executed on the IBM Blue Gene/P. Consequently, scheduling and coordination of the execution cannot be performed manually, and a single system cannot support it. Finally,
because the non-linear solver is iterative, it may fail to converge for some combinations of input parameters, in which case fault-tolerance mechanisms should be engaged. The above properties make the problem impossible for the end-user to solve using the standard computational resources (e.g. computational allocation from XSEDE). At the same time, they exemplify main advantages of our proposed federation model.

4.2 Experimental Setup

In order to run our computational problem on a federation of resources, we integrated the MPI-based solver with the federation framework using the master/worker paradigm. In this scenario, the simulation software serves as a computational engine, while federation framework is responsible for orchestrating the entire execution. We implemented a prototype of the described federation using CometCloud [10]. Here, CometCloud provides a basic functionality on top of which a federation can be achieved. For example, it offers autonomic capabilities, fault tolerance mechanisms, and transparent access to cloud, grid, and HPC infrastructures. As a result, sites can join and leave the federation at any moment without interrupting the execution. We note that although master/worker paradigm best fits our problem, the proposed federation model and its CometCloud implementation also support MapReduce and workflows.

We identified a total of 12,400 simulations (tasks) as essential to interrogate the parameter space at the precision level satisfactory to the end-user. The estimated collective cost of these tasks is 1.5 million core-hours if executed on the Stampede cluster. While this number is already challenging, we note that approximately 300,000 tasks would be required to provide a fine-grained view of the parameter space. As we already mentioned, tasks are very heterogeneous in terms of hardware requirements and computational complexity. This is because of varying mesh density and size, as well as convergence rate of the solver. For instance, some tasks require minimum 512 GB of total RAM, while many can execute in 64 GB. To accommodate for this variability we classified tasks into three groups (small, medium, large), based on their estimated minimal hardware requirements. Although this classification is necessarily error-prone, due to non-trivial dependencies between mesh size, and memory and time complexity, it serves as a good proxy based on which computational sites can decide which tasks to pull. At the same time, misclassified tasks can be handled by fault-tolerance mechanisms of CometCloud.

To execute the experiment we federated 10 different resources, provided by six institutions from three countries. The characteristics of the selected machines are summarized in Tables 1 and 2. As can be seen, utilized resources span different hardware architectures and queuing systems,
ranging from the high-end supercomputers to small-scale servers. Depending on the hardware characteristics different machines accepted tasks from different classes (see Table 2). This was achieved by providing a simple configuration file to respective CometCloud worker. Our initial rough estimates indicated that the first seven machines (Excalibur, Snake, Stampede, Lonestar, Hotel, India, Sierra) would be sufficient to carry out the experiment, and conclude it within two weeks. However, during the experiment, as we explain later, we decided to integrate additional resources (Carver, Hermes, Libra). Because all machines were used within limits set by the hosting institutions no special arrangements were made with their system administrators, and both the-end users’ software and CometCloud components were deployed using a basic SSH account.

4.3 Experimental Results

The experiment lasted 16 days during which 10 different HPC resources were federated, and total of 12,845 tasks were executed. Together, all tasks consumed 2,897,390 core-hours, and generated 398 GB of the output data. The progress of the experiment is summarized in Figure 3.

The initial configuration of the federation included only five machines (Excalibur, Snake, Stam-
Figure 3: Summary of the experiment. Top: Utilization of different computational resources. Line thickness is proportional to the number of tasks being executed at given point of time. Gaps correspond to idle time, e.g. due to machine maintenance. Bottom: The total number of running tasks at given point of time.

Figure 4: Throughput and queue waiting time. Top: Dissection of throughput measured as the number of tasks completed per hour. Different colors represent component throughput of different machines. Middle: Throughput contribution by different task classes. Bottom: Queue waiting time on selected resources. Please view in color.
pede, Lonestar, Hotel) out of seven planned. Two other machines, India and Sierra, joined with a delay caused by maintenance issues. After the first day of execution it became apparent that more computational resources were needed to finish the experiment within assumed deadline. This is because some machines were experiencing problems, and more importantly, our XSEDE allocation on Stampede was being exhausted rapidly. At that point, the first significant feature of our solution came into play – thanks to the extreme flexibility of the CometCloud platform temporal failures of individual resources did not interrupt the overall progress, and adding new resources was possible within few minutes from the moment the access to a new resource was acquired, and the simulation software was deployed. Indeed, on the second day Hermes from Spain was added to the execution pool, and soon after NERSC’s Carver, and Libra from Singapore were federated. Consequently, the federation was able to sustain computational performance. Figure 3 shows that most of the time anywhere between 5 and 25 simulations were running, despite multiple idle periods scattered across the majority of the machines. These idle periods were caused by common factors, such as for example, hardware failures and long waiting times in system queues. All failures were handled by the CometCloud fault-tolerance mechanism. During the experiment 249 tasks had to be regenerated due to hardware errors, and 167 due to inability of the solver to converge. We note, that 29 additional tasks were run as a result of a speculative execution. All this demonstrates great robustness of the framework – depending on the availability of resources, and the rate of the execution, federation can be scaled up or down accordingly.

Figure 4 outlines how the computational throughput, measured as the number of tasks completed per hour, was shaped by different computational resources. Here, several interesting observations can be made. First, no single resource dominated the execution. Although Stampede, the most powerful machine among all federated, provided a brief performance burst during the first two days, it was unable to deliver a sustained throughput. In fact, tasks on this machine were submitted to the “development” queue that limits the number of processors used by a job, but offers relatively high turnover rate. Yet, even this queue got saturated after the first day of execution, which caused a sudden drop in the throughput. This pattern can be observed on other systems as well (e.g., see Lonestar and Carver), and it confirms our earlier observation that no single system can offer a sufficient throughput. Another observation is related to how the throughput was distributed in time. The peak was achieved close to the end of the experiment, even though after twelfth day Excalibur was running at half its initial capacity (see Figure 3). This can be explained by the fact that the majority of tasks executing towards the end were small tasks. Consequently, all available resources were able to participate in execution, and short runtimes increased the overall throughput.

The last important element of the experiment was data management. In our case, the input data consisted of two components: a finite element mesh database tightly integrated with the simulation software, and hence deployed together with the software, and a 4-tuple describing simulation parameters. As a result, no special mechanisms were required to handle the input. The output data consisted of simulation results and several small auxiliary files. The size of the output varied between simulations ranging from 3 MB to 30 MB when compressed. The data was compressed in situ and on-the-fly during the experiment, and then transferred using the RSYNC protocol to the central repository for a subsequent analysis.

The presented results clearly demonstrate feasibility and capability of our proposed federation model. In our experiment a single user, with basic SSH access to several globally distributed and heterogeneous resources, was able to solve a large-scale computational engineering problem, within just two weeks. Importantly, this result was achieved in a few simple steps executed completely in
a user-space. By providing a simple master/worker code the user gained access to a unified and fault-tolerant platform able to sustain computational throughput.

4.4 Science Outcomes

The above experiment provided the most comprehensive data on the effect of pillars on microfluid channel flow. Although we are still in the process of analyzing this massive output, we already gained several interesting insights regarding fundamental features of the flow. Figure 5 shows how different flow modes are distributed in the parameter space. Here, each mode corresponds to one or two vortices generated, as proposed in [2]. In the introduction to this section we hinted that by arranging pillars into a specific sequence it is possible to perform basic flow transformations. Thanks to the library of flow configurations that we generated in this experiment, we can now investigate the inverse problem and, for example, ask questions about the optimal pillar arrangement to achieve a desired flow output. The implications of such capability are far-reaching, with potential applications in medical diagnostics and smart materials engineering.

Figure 5: The phase diagram showing how different flow modes are distributed in the parameter space. Here, pillar offset is 0, $D$ is a pillar diameter, $h$ is a channel height, $w$ is channel width, and $Re$ is Reynolds number. Please view in color.

5 Conclusions

Providing an easy access to large-scale computational resources is one of the most important challenges facing the entire HPC community. In this article we presented a software-defined federation model aimed to empower average user with computational capabilities typically reserved for high-profile computational problems. The proposed model offers a unified view of heterogeneous HPC resources, and exposes them using cloud-like capabilities. At the same time the model remains user-centered, and can be used by any user without special privileges on the federated resources. To demonstrate applicability of our approach we solved the actual problem of constructing phase diagram of possible flow behaviors in the microscale devices. This experiment not only confirms great flexibility and potential of a user-centered computational federation, but also provides the most comprehensive data on the effect of pillars on microfluid channel flow.
6 Acknowledgments

This work is supported in part by the NSF under grants ACI 1339036, IIP-0758566, DMS-0835436, CAREER-1149365, PHY-0941576, and by IBM via OCR and Faculty awards. This project used resources provided by: XSEDE supported by NSF OCI-1053575, FutureGrid supported in part by NSF OCI-0910812, and NERSC Center supported by DOE DE-AC02-05CH11231. The authors would like to thank the SciCom group at the Universidad de Castilla-la Mancha, Spain (UCLM) for providing access to Hermes, and Distributed Computing research group at the Institute of High Performance Computing, Singapore (IHPC) for providing access to Libra. The authors would like to acknowledge the CINECA, Italy, LRZ, Germany, CESGA, Spain, and NICS for willing to share their computational resources. The authors would like to thank Dr. O. Wodo for discussion and help with development of the simulation software, Dr. D. DiCarlo for discussions about the problem definition, and M. Abdelbaky for helpful comments on early version of this paper. The authors express gratitude to all administrators of systems used in this experiment, especially to P. Bisbal from RDI\textsuperscript{2} and K. Tanaka from FutureGrid, for their efforts to minimize downtime of computational resources, and a general support.

References


Javier Diaz-Montes is currently Research Associate at Rutgers University and a member of the Rutgers Discovery Informatics Institute (RDI2) and the US National Science Foundation (NSF) Cloud and Autonomic Computing Center. He received his PhD degree in Computer Science from the Universidad de Castilla-La Mancha (UCLM), Spain (“Doctor Europeus”, Feb. 2010). Before joining Rutgers, he was Postdoctoral Fellow of the Pervasive Technology Institute at Indiana University. His research interests are in the area of parallel and distributed computing and include autonomic computing, grid computing, cloud computing, virtualization and scheduling. He is a member of IEEE and ACM.

Yu Xie received the B.S. degree in Mechanical Engineering from Peking University, Beijing, China, in 2009. He is currently working toward the Ph.D. degree in the Department of Mechanical Engineering at Iowa State University. His research interests include high performance computing, uncertainty quantification of complex systems, and finite element method for multiphase flow simulation.

Ivan Rodero is an assistant research professor at Rutgers University and a member of the Discovery Informatics Institute (RDI2) and the US National Science Foundation (NSF) Cloud and Autonomic Computing Center. His research interests fall in the broad area of parallel and distributed computing and include high-performance computing, energy efficiency, autonomic computing, grid computing, cloud computing, and data analytics at extreme scales. Rodero has a PhD in computer science and engineering from the Technical University of Catalonia. He is a member of IEEE, ACM and the American Association for the Advancement of Science (AAAS). Contact him at irodero@cac.rutgers.edu.

Jaroslaw Zola is an Associate Research Professor at Rutgers University. Prior to joining Rutgers he was a faculty at Iowa State University. Dr. Zola received M.Sc. degree in computer science from Czestochowa University of Technology, Poland, and Ph.D. degree from Grenoble Institute of Technology, France, in 2001 and 2005, respectively. His research activities are focused on data driven large-scale computing in life sciences and engineering. He is a senior member of the Institute for Electrical and Electronics Engineers (IEEE). He is also member of ACM, ISCB and AAAS.
Baskar Ganapathysubramanian is an Assistant Professor of Mechanical Engineering at Iowa State University. His research interests are in stochastic analysis, multiscale modeling, and design of materials and processes using computational techniques. Ganapathysubramanian completed his PhD and MS from Cornell University and holds a BS degree from the Indian Institute of Technology-Madras.

Manish Parashar is Professor of Electrical and Computer Engineering at Rutgers University. He is the founding Director of the Rutgers Discovery Informatics Institute (RDI2) and of the NSF Cloud and Autonomic Computing Center (CAC), and is Associate Director of the Rutgers Center for Information Assurance (RU-CIA). Manish received a BE degree from Bombay University, India and MS and Ph.D. degrees from Syracuse University. His research interests are in the broad areas of Parallel and Distributed Computing and Computational and Data-Enabled Science and Engineering. A key focus of his research is on addressing the complexity or large-scale systems and applications through programming abstractions and systems. Manish serves on the editorial boards and organizing committees of a large number of journals and international conferences and workshops, and has deployed several software systems that are widely used. He has also received numerous awards and is Fellow of AAAS, Fellow of IEEE/IEEE Computer Society and Senior Member of ACM. For more information please visit http://parashar.rutgers.edu/.