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 and Jaroslaw Zola | Rutgers University
Baskar Ganapathysubramanian | Iowa State University
Manish Parashar | Rutgers University

An exploration of the use of aggregated high-performance computing resources to solve large-scale engineering problems shows that it's possible to build a computational federation that's easy for users to implement, and is elastic, resilient, and scalable. The fusion of federated computing and real-life engineering problems is brought to the average user by providing relevant middleware.

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 a few examples of a broad class of problems that are becoming increasingly important in a wide range of application domains. These ensemble (also called many-task computing) 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 high-performance computing (HPC) resource. While many of these problems are conveniently parallel, their collective complexity exceeds computational time and throughput that average users can obtain from a single computational center. For instance, the fluid flow problem considered in this article comprises more than 10,000 message passing interface (MPI) tasks, and would require approximately—1.5 million core-hours to solve on the Stampede cluster at the Texas Advanced Computing Center—one of the most powerful machines within the Extreme Science and Engineering Discovery Environment (see www.xsede.org). Although XSEDE allocations of that size aren't uncommon, the heavy use 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 aren't uncommon in large-scale multiuser systems.

These constraints aren't unique to one particular problem or a system. Rather, they represent common obstacles that limit the scale of problems that an ordinary researcher can consider on a single, but powerful, system. What's important is that this trend continues and we can only expect that more and more users will require computational throughput that can't be delivered just by one resource. To

 1521-9615/14/$31.00 © 2014 IEEE Copublished by the IEEE CS and the AIP

Other Approaches to Federated Computing

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.[1-5] For example, volunteer computing systems (such as the Berkeley Open Infrastructure for Network Computing) enable the aggregation of user resources, provided by a crowd of volunteers, to obtain nontrivial computing capabilities for an application. While this model is easy to configure and use from a user perspective, it can support only a limited class of applications; that is, those with large numbers of small and independent tasks. At the other end of the spectrum, grids—for example, Enabling Grids for E-sciencE (now supported by the European Grid Infrastructure; see www.egi.eu), Grid'5000 (www.grid5000.fr), and Open Science Grid (www.opensciencegrid.org)—have targeted more computing/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.[4] More recently, cloud federations are being explored as a means to extend as-a-service models to virtualized data-center federations. Given the increasing importance of ensemble applications and their computational requirements, it's becoming important to revisit user-centered federated computing from the perspective of this class of applications and their requirements.

References

1. G. Allen and D. Katz, *Computational Science, Infrastructure And Interdisciplinary Research on University Campuses: Experiences and Lessons from the Center for Computation & Technology*, tech. report CCT-TR-2010-1, Center for Computation & Technology, Louisiana State Univ., 2010.
2. F. Berman, G. Fox, and A. Hey, *Grid Computing: Making the Global Infrastructure a Reality*, John Wiley & Sons, 2003.
3. G. Garzoglio et al., "Supporting Shared Resource Usage for a Diverse User Community: The OSG Experience and Lessons Learned," *J. Physics: Conf. Series*, vol. 396, 2012; doi:10.1088/1742-6596/396/3/032046.
4. M. Parashar and C. Lee, "Special Issue on Grid Computing," *Proc. IEEE*, vol. 93, no. 3, 2005.
5. P. Riteau et al., "Large-Scale Cloud Computing Research: Skycomputing on FutureGrid and Grid'5000," *ERCIM News*, 2010; http://ercim-news.ercim.eu/en83/special/large-scale-cloud-computing-research-sky-computing-on-futuregrid-and-grid5000.

overcome these limitations, we need to address two important questions: how to empower a researcher with computational capability compatible to that currently reserved for the elite problems; and how to deliver this capability in a user-centered manner. Here, 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.

Thus, in this article we explore the use of software-defined federated computing to solve large-scale engineering problems in a user-centered way (for other approaches, see the related sidebar). Our focus is on empowering the 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

- lets users be in control of the federation process by specifying resources accessible to them, the constraints associated to those resources, and how they want to make use of the resources as part of their federations;
- is extremely easy to deploy and offers an intuitive API to meet the expectations and needs of the average user;
- encapsulates cloud-like capabilities, for example, on-demand resource provisioning, elasticity, and resilience, to provide sustainable computational throughput;
- provides strong fault-tolerance guarantees through constant monitoring of tasks and resources;
- bridges multiple, highly heterogeneous resources, for example, servers, clusters, supercomputers, and clouds, to effectively exploit their intrinsic capabilities; and
- leverages security and authentication from the underlying infrastructure.

To demonstrate the potential of the resulting federated infrastructure in addressing the computational requirements of real-world, large-scale computational engineering problems, we implemented 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 distributed HPC centers, consumes more than 2.5 million core-hours, and provides the most comprehensive data to-date on the effect of pillars on flow in microchannels.

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's imperative to understand 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 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 (for example, sparse grid collocation), or soft computing approaches (such as simulated annealing). These techniques constitute a significant fraction of all scientific codes in use today, and hence are of great practical importance to the average user.

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 the problem logic, for example, to decide how the search-space should be navigated; 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 (for example, as a script interacting with a queuing system). Individual instances of a science-driver are either independent or involve asynchronous communication. Naturally, complexity of both modules might vary drastically. However, in the vast majority of cases it's 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 a large number of tasks (for example, more than 12,000 in our study, and millions in any Monte Carlo analysis). Often 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 in simple scenarios, tasks are generated dynamically, based on partial or complete results delivered by previously completed tasks.

Shaping the Federation from the Scientist's Perspective

We focus now on what type of user expectations must be addressed 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, such as 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's unrealistic to expect that a user will have 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 such as master/worker or MapReduce, which the user could access directly. At the same time, a user must be able to deploy existing applications, such as science-drivers and sometimes a problem logic module, within the federation and without modifications.

If we look at these characteristics, it becomes apparent that there are several key features for which our federation model must account. The federation must be elastic and scalable—the ability to scale up/down and out becomes essential to handling a varying number of tasks over time. What's 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

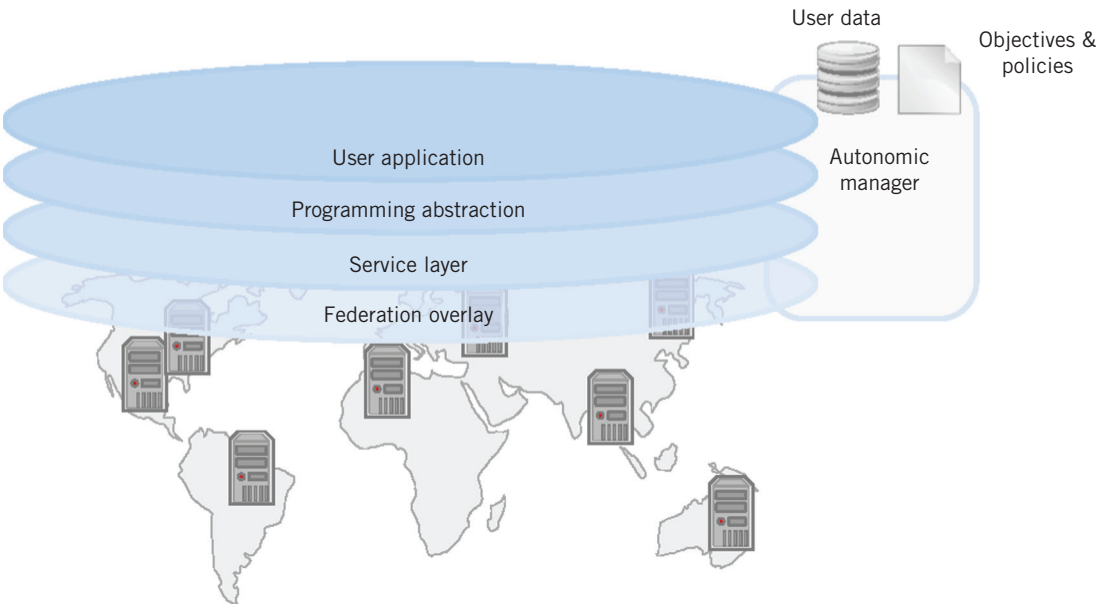

Figure 1. Multilayer design of the proposed federation model. Here, the autonomic manager is a cross-layer component that, based on user data and policies, provisions appropriate resources.

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.

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 at allowing 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 lets us separate the different functionalities 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 lets users 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 autonomics at the programming and application level. It includes a coordination service that handles application execution, 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 paradigms, including master/worker workflow and MapReduce. These programming abstractions ease the development of applications by decoupling the application from the infrastructure particularities. 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 employ the federation to execute third-party, perhaps closed-source, software. In such cases the target software can't or shouldn't 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 multiobjective optimization (including performance, energy, cost, and reliability criteria) of application execution through cross-layer application/infrastructure adaptations. This component offers quality of service (QoS) by adapting the provisioned resources to the application's behavior as well as system configuration, which can change at runtime, using the notion of elasticity at the application level. As a result, the federated infrastructure increases the opportunities to provision appropriate resources for a 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 given budget constraints, to complete the application within an assumed deadline, or to use resources best matching to the application type (for example, computation versus data-intensive). Because application requirements and resource status might 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 federation resilience.

The security and authentication are leveraged solutions provided by each site (for example, based on Secure Socket Shell, or SSH). 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 lets users select their preferred authentication/authorization mechanisms among those directly supported by each site.

Case Study

To demonstrate the applicability and scalability of our federation model in a 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 model uncertainty when input data is incomplete or noisy, or to establish a ground on which inverse problems can be investigated. While these techniques are diverse, typically they involve a large collection of computationally intensive tasks, with little or no synchronization between the tasks.

Application Description

Our focus on the fluid flow problem is motivated by its great practical importance. The ability to control fluid streams at the microscale has significant applications in many domains, including biological processing,[1] guiding chemical reactions,[2] and creating structured materials.[3] Two of the authors (henceforth, they're 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.[4] The design and placement of pillar sequences enables a phenomenal degree of flexibility to program the flow for various biomedical and manufacturing applications. However, to achieve such a control it's 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, pillar diameter, and Reynolds number (four variables), the solver captures both qualitative and quantitative characteristics of flow (see Figure 2). 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 might 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 execution cost is difficult to estimate a priori, owing to varying resolution and mesh density required for different configurations. In our case, the cost can range from 100 to 100,000 core-hours per task executed on the IBM Blue Gene/P. Consequently, scheduling and coordination of the execution can't be performed manually, and a single system can't support it. Finally, because the nonlinear solver is iterative, it might fail to converge for some combinations of input parameters, in which case fault tolerance mechanisms should be engaged. These properties make the problem impossible for the end user to solve using the standard computational resources (for example, computational allocation from XSEDE). At the same time, they exemplify the main advantages of our proposed federation model.

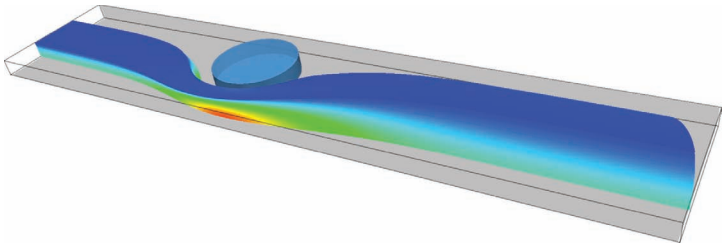

Figure 2. Example flow in a microchannel with a pillar. Four variables characterize the simulation: channel height, pillar location, pillar diameter, and Reynolds number.

Experimental Setup

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 the federation framework is responsible for orchestrating the entire execution. We implemented a prototype of the described federation using CometCloud (see www.cometcloud.org). 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 the 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 heterogeneous in terms of hardware requirements and computational complexity. This is because of varying mesh density and size, as well as the convergence rate of the solver. For instance, some tasks require a minimum of 512 Gbytes of total RAM, while many can execute with 64 Gbytes. To accommodate for this variability, we classified tasks into three groups (small, medium, and large) based on their estimated minimal hardware requirements. Although this classification is necessarily error-prone due to nontrivial 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 the fault tolerance mechanisms of CometCloud.

To execute the experiment we federated 10 different resources, provided by six institutions from three countries. Tables 1 and 2 summarize the characteristics of the selected machines. As can be seen, used resources span different hardware architectures and queuing systems, ranging from 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 the respective CometCloud worker. Our initial rough estimates indicated that the first seven machines (Excalibur, Snake, Stampede, Lonestar, Hotel, India, and Sierra) would be sufficient to carry out the experiment, and conclude it within two weeks. However, during the experiment, as we

Table 1. Computational resources used to execute the experiment.

Name	Provider	Type	Cores*	Memory**	Network	Scheduler
Excalibur	Rutgers Discovery Informatics Institute (RDI2)	IBM Blue Gene/P (BG/P)	8,192	512 Mbytes	BG/P	LoadLeveler
Snake	RDI2	Linux symmetric multiprocessing (SMP)	64	2 Gbytes	N/A	N/A
Stampede	Extreme Science and Engineering Discovery Environment (XSEDE)	iDataPlex	1,024	4 Gbytes	InfiniBand (IB)	SLURM
Lonestar	XSEDE	iDataPlex	480	2 Gbytes	IB	Sun Grid Engine (SGE)
Hotel	FutureGrid	iDataPlex	256	4 Gbytes	IB	Torque
India	FutureGrid	iDataPlex	256	3 Gbytes	IB	Torque
Sierra	FutureGrid	iDataPlex	256	4 Gbytes	IB	Torque
Carver	Dept. of Energy/ National Energy Research Scientific Computing Center (NERSC)	iDataPlex	512	4 Gbytes	IB	Torque
Hermes	Universidad de Castilla-La Mancha, Spain	Beowulf	256	4 Gbytes	10 Gigabit Ethernet (GbE)	SGE
Libra	Institute of High Performance Computing, Singapore	Beowulf	128	8 Gbytes	1 GbE	N/A

* Peak number of cores available to the experiment.
** Memory per core.

Table 2. Capability of each resource.

Name	Cores per task	Accepted tasks
Excalibur	1,024	Small/medium/large
Snake	64	Small/medium
Stampede	128	Small/medium/large
Lonestar	120	Small/medium/large
Hotel	128	Small/medium/large
India	128	Small/medium/large
Sierra	128	Small/medium/large
Carver	256	Small/medium
Hermes	128	Small/medium/large
Libra	128	Small/medium

explain later, we decided to integrate additional resources (Carver, Hermes, and 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.

Experimental Results

The experiment lasted 16 days during which we federated 10 different HPC resources and executed a total of 12,845 tasks. Together, all tasks consumed 2,897,390 core-hours and generated 398 Gbytes of the output data. Figure 3 summarizes the experiment's progress.

The initial federation configuration included only five machines (Excalibur, Snake, Stampede, Lonestar, and 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 the 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

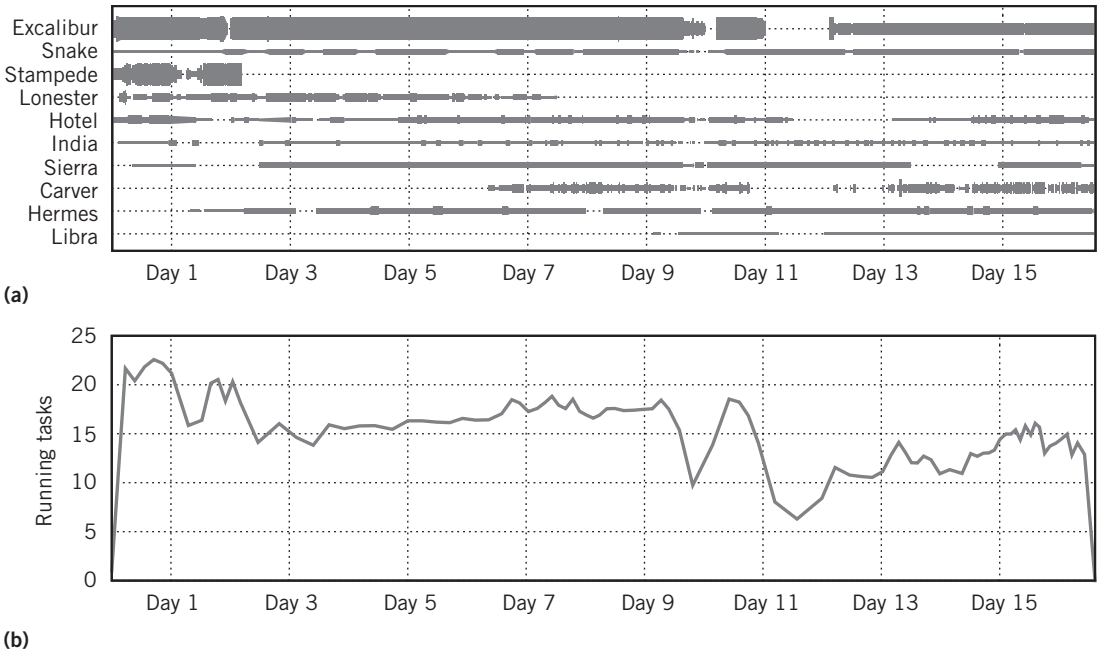

Figure 3. Summary of the experiment. (a) Use of different computational resources. Line thickness is proportional to the number of tasks being executed at given point of time. Gaps correspond to idle time, for example, due to machine maintenance. (b) The total number of running tasks at a given point of time.

came into play—thanks to the extreme flexibility of the CometCloud platform, temporal failures of individual resources didn't interrupt the overall progress, and adding new resources was possible within a few minutes from the moment the access to a new resource was acquired and the simulation software deployed. Indeed, on the second day Hermes from Spain was added to the execution pool, and soon after the National Energy Research Scientific Computing Center's (NERSC's) Carver, and Singapore's Libra were federated. Consequently, the federation was able to sustain computational performance. Figure 3 shows that most of the time there were between five and 25 simulations, despite multiple idle periods scattered across the majority of the machines. These idle periods were caused by common factors, such as hardware failures and long waiting times in system queues. The CometCloud fault tolerance mechanism handled all failures. 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 framework robustness—depending on resource availability and execution rate, the 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, we can make several interesting observations. 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 (for example, 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 the 12th 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,

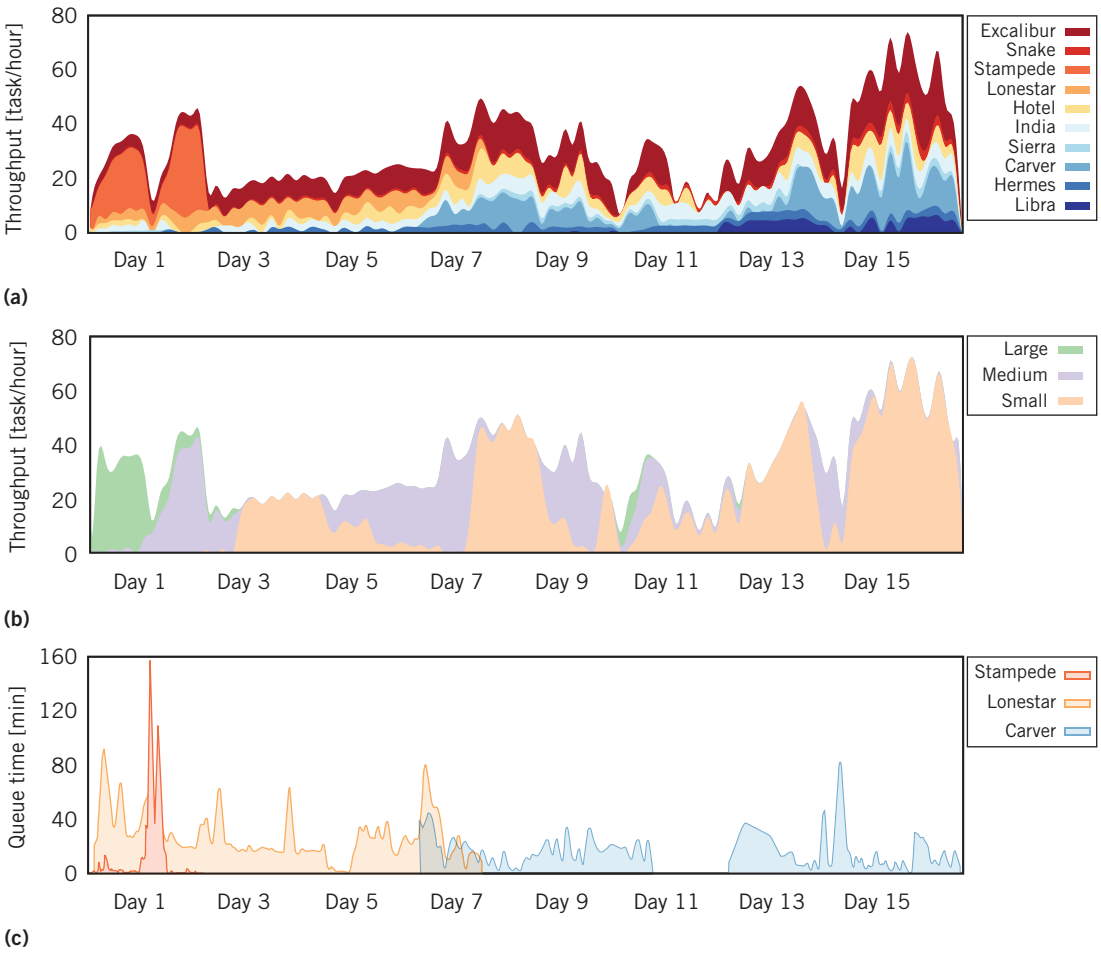

Figure 4. Throughput and queue waiting time. (a) Dissection of throughput measured as the number of tasks completed per hour. Different colors represent component throughput of different machines. (b) Throughput contribution by different task classes. (c) Queue waiting time on selected resources.

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 output size varied between simulations ranging from 3 to 30 Mbytes 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 the 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.

This experiment provided what is currently the most comprehensive data available on the effect of pillars on microfluid channel flow. Although we're still in the process of analyzing this massive output, we already gained several interesting

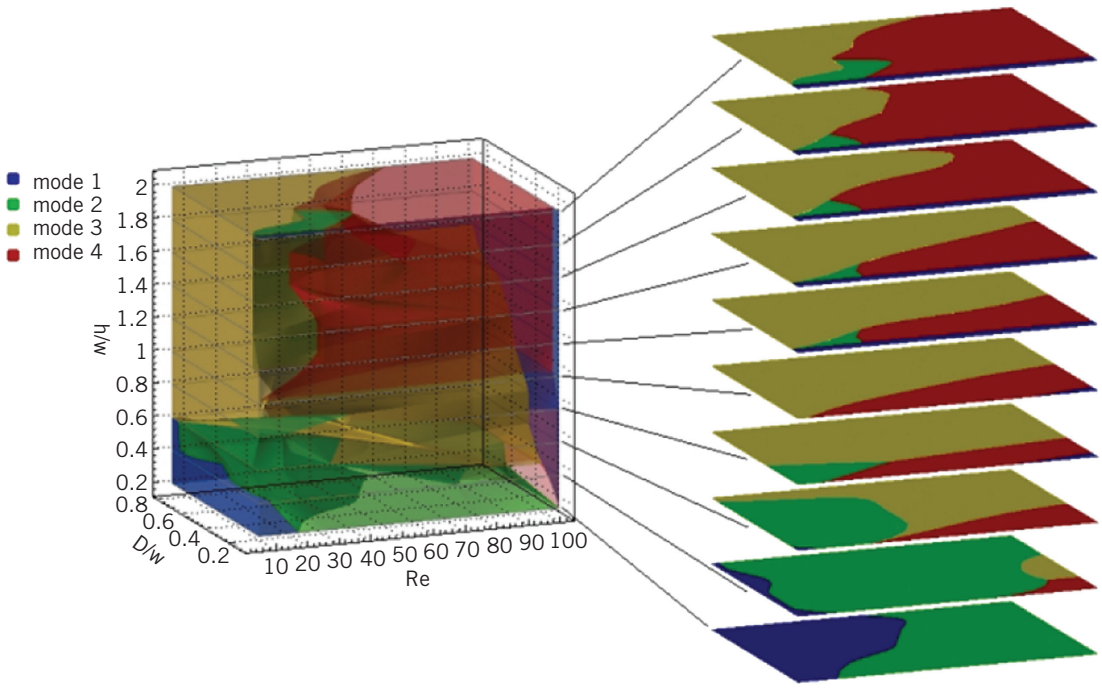

Figure 5. The phase diagram showing how different flow modes are distributed in the parameter space. Here, pillar offset is *O*, *D* is a pillar diameter, *h* is a channel height, *w* is channel width, and *Re* is the Reynolds number.

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 other work.[4] In the introduction to this section, we hinted that by arranging pillars into a specific sequence that it's 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 capabilities are far reaching, with potential applications in medical diagnostics and smart materials engineering. ■

Acknowledgments

This work is supported in part by the US National Science Foundation (NSF) under grants ACI 1339036, IIP-0758566, DMS-0835436, CAREER-1149365, PHY-0941576, CBET-1307743, CBET-1306866, and by IBM via Open Collaboration Research 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 supported by DoE DE-AC02-05CH11231. We thank the SciCom group at the Universidad de Castilla-la Mancha, Spain, for providing access to Hermes; and the Distributed Computing research group at the Institute of High-Performance Computing, Singapore, for providing access to Libra. We acknowledge the Consorzio Interuniversitario del Nord est Italiano Per il Calcolo Automatico (CINECA), Italy; LRZ, Germany; Centro de Supercomputación de Galicia (CESGA), Spain; and National Institute for Computational Sciences (NICS) for sharing their computational resources. We thank Olga Wodo for discussion and help with development of the simulation software, Dino DiCarlo for discussions about the problem definition, and Moustafa Abdelbaky for helpful comments on an early version of this article. We're grateful to all the administrators of systems used in this experiment, especially to Prentice Bisbal from RDI2 and Koji Tanaka from FutureGrid, for their efforts to minimize downtime of computational resources and for general support.

References

1. J. Wang et al., "Vortex-Assisted DNA Delivery," *Lab on a Chip*, vol. 10, 2010, pp. 2057–2061.
2. Y. Gambin et al., Visualizing a One-Way Protein Encounter Complex by Ultrafast Single-Molecule Mixing, *Nature Methods*, vol. 8, 2011, pp. 239–241.
3. H. Lee et al., "Colour-Barcoded Magnetic Microparticles for Multiplexed Bioassays," *Nature Materials*, vol. 9, 2010, pp. 745–749.

4. H. Amini et al., "Engineering Fluid Flow Using Sequenced Microstructures, *Nature Comm.*, 2013; doi:10.1038/ncomms2841.

Javier Diaz-Montes is an assistant research professor at Rutgers University and a member of the Rutgers Discovery Informatics Institute (RDI2) and the US NSF Cloud and Autonomic Computing Center. His research interests include parallel and distributed computing, autonomic computing, grid computing, cloud computing, virtualization, and scheduling. Diaz-Montes has a PhD in computer science from the Universidad de Castilla-La Mancha, Spain. He's a member of IEEE and the ACM. Contact him at javier.diazmontes@gmail.com.

Yu Xie is working toward a PhD in the Department of Mechanical Engineering at Iowa State University. His research interests include HPC, uncertainty quantification of complex systems, and finite element method for multiphase flow simulation. Xie has a BS in mechanical engineering from Peking University, Beijing. Contact him at yuxie@iastate.edu.

Ivan Rodero is an assistant research professor at Rutgers University and a member of the RDI2 and the NSF Cloud and Autonomic Computing Center. His research interests include parallel and distributed computing, 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's a member of IEEE, the 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 and a member of the RDI2. His research interests include data-driven, large-scale computing in life sciences and engineering. Zola has a PhD in computer science from Grenoble Institute of Technology, France. He's a senior member of IEEE, and a member of ACM, the International Society for Computational Biology (ISCB), and AAAS. Contact him at jaroslaw.zola@rutgers.edu.

Baskar Ganapathysubramanian is an associate professor of mechanical engineering at Iowa State University. His research interests include stochastic analysis, multiscale modeling, and design of materials and processes using computational techniques. Ganapathysubramanian has a PhD in mechanical and aerospace engineering from Cornell University. Contact him at baskarg@iastate.edu.

Manish Parashar is a professor in the Department of Computer Science at Rutgers University, he's the director of the Rutgers Discovery Informatics Institute (RDI2) and the US National Science Foundation (NSF) Cloud and Autonomic Computing Center (CAC) at Rutgers, and he's the associate director of the Rutgers Center for Information Assurance. His research interests focus on applied parallel and distributed computing and computational and data-intensive science and engineering. Parashar has a PhD in computer engineering from Syracuse University. He received the IBM Faculty award twice, as well as a US NSF Career award, and he is an American Association for the Advancement of Science (AAAS) and IEEE Fellow. Contact him at parashar@rutgers.edu.