Exploring the Use of Elastic Resource Federations for Enabling Large-scale Scientific Workflows

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

An important class of scientific and engineering workflows, e.g. those used for uncertainty quantification, design optimization and parametric studies, naturally map onto the Many-Task Computing (MTC) paradigm. However, what distinguishes these workloads is a unique combination of dynamically changing resource requirements and very large computational and throughput demands. Such workflows can benefit from an elastic execution infrastructure that is based on the dynamic federation of resources. The overarching goal of this paper is to explore the nature of such an elastic, dynamically federated platform, and to experimentally demonstrate that it can effectively support the targeted class of scientific and engineering workflows. As a driving application for our study we use the problem of constructing a phase diagram in microfluidics, which is representative for a broader class of parameter space interrogation techniques. To satisfy its computational demands of 2.5 million core-hours within reasonable time limits, we construct a dynamic federation of ten HPC resources from six different computing centers. This experiment delivers the most comprehensive data on fluid flow in a microchannel with an obstacle. Moreover, it offers important insights that enable us to identify key requirements and architectural components that a platform based on federated resources must provide in order to efficiently handle considered scientific MTC workloads.

1. INTRODUCTION

The rapid and sustained progress in computational modeling, and the resulting computational and data complexity, require new approaches to large-scale 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" applications usually 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. At the same time, these applications can be generally described as MTC [19], and can benefit from a federation of computational resources. However, they differ from traditional MTC applications as they combine large computational and throughput demand with dynamically changing resource requirements. Consequently, such workloads are hard to efficiently support on classic federation models in which a user is presented with a fixed set of resources. All this necessitates the exploration of new federation models that could be dynamically shaped to meet the specific properties of scientific MTC applications.

In this paper, we investigate the use of a dynamic federation to support large-scale scientific and engineering workflows. To perform the study, we solve the actual problem of constructing phase diagram of possible flow behaviors in a microchannel. The problem is highly representative for a broad spectrum of methods in which large search-spaces are analyzed in a coordinated manner. Because the problem involves executing over 12,000 highly heterogeneous and compute intensive MPI tasks, we develop a federation which can be elastically adapted to the changing requirements of the application and the availability of the resources over time. The federation spans ten different HPC resources from six computational centers, and delivers 2.5 million core-hours required to conclude the tasks. This experiment not only delivers the most comprehensive data on fluid flow in a microchannel with an obstacle, but offers important insights that enable us to identify key requirements and architectural components that a federation must provide in order to support large-scale scientific MTC workloads.

The reminder of this paper is organized as follows. In Section 2 we introduce the fluid flow problem, our representative scientific application with dynamic resource requirements. In Section 3 we describe a model of elastic federation on which we execute the test case. In Section 4 we present our test environment and experimental results followed by discussion in Sections 5 and 6. Finally, in Section 7 we review related work, and then conclude the paper in Section 8.

2. USE CASE

In order to perform our study we selected the question of understanding fluid flow in microdevices. Our choice is motivated by the great practical importance of this problem. The ability to control fluid streams at microscale has significant applications in many domains, including biological processing [26], guiding chemical reactions [9], and creating structured materials [15]. 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 provides 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 1). 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.

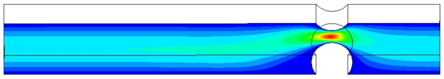

Figure 1: Example flow in microchannel with pillar.

The problem is challenging for several reasons. Although it clearly can be expressed as MTC, varying and non-trivial computational requirements make it very dynamic. The search space consists of tens of thousands of points, and an individual simulation may take hundreds of core-hours, even when executed using parallel computing on a state-of-the-art HPC cluster. The individual tasks 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 is insufficient to 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 practically impossible for the end-user to solve using standard computational resources (e.g. computational allocation from XSEDE).

It is important to note that the described 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. Collectively, different variants of parameter space interrogation techniques constitute a significant fraction of all computational workloads executed on HPC resources these days [12].

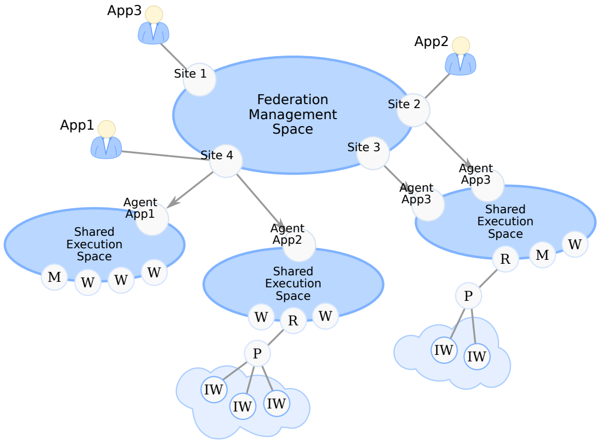

Figure 2: Architecture of CometCloud-based federation model. Here, (M) denotes a master, (W) is a worker, (IW) an isolated worker, (P) a proxy, and (R) is a request handler. Arrows represent deployment of a computational site, while lines show communication.

3. FEDERATION ARCHITECTURE

To build a federation that could dynamically evolve in terms of size and capabilities, and could serve as the execution and test environment for our selected application, we decided to use the CometCloud framework [28]. 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 the transparent access to cloud, grid, and HPC infrastructures. Moreover, it defines a flexible API that supports several common programming paradigms such as, e.g. Master/Worker, Map/Reduce and Bag-of-Tasks.

The general overview of our CometCloud-based federation is provided in Figure 2. The core element and the driving mechanism used to coordinate different aspects of the federation are the Comet coordination spaces [16]. Specifically, we define two types of spaces. First, we have a single federated management space for creating the actual federation and orchestrating different resources. This space is responsible for interchanging any operational messages to discover resources, announcing changes in a site, routing users' requests to appropriate sites, and initiating negotiations to create ad-hoc execution spaces. Then, we have multiple shared execution spaces that are created on demand to satisfy computational needs of the users. Execution spaces can be created in the context of a single site to provision local resources, and can cloudburst to public clouds or external HPC systems. Moreover, they can be used to create a private sub-federation across several sites. This case can be useful when several sites have some common interest and they decide to jointly target certain type of tasks as a spe-

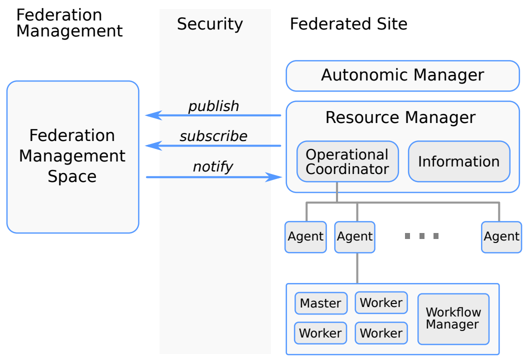

Figure 3: Main elements implementing a federated site. Each agent is responsible for executing tasks using one of the programming models.

cialized community.

Each shared execution space is controlled by an agent that creates the space, and coordinates the resources that execute a particular set of tasks. Agents can act as a master of the execution, or delegate this duty to a dedicated master (M) when some specific functionality is required. Additionally, agents deploy workers to perform the actual execution of tasks. These workers can be in a trusted network, be part of the shared execution space and store data, or they can be part of external resources such as public clouds, and therefore in a non-trusted network. The first type of workers is called secure (W), and can pull tasks directly from the space. The second type is called isolated (IW), and cannot directly interact with the shared space. Instead, isolated workers depend on a proxy (P), and a request handler (R), to obtain tasks from the space. This distinction is important since it allows us to define specific boundaries in the way data is accessed. This in turn can be used to optimize the data storage and exploit data locality. Moreover, this mechanism can be used to define security policies and decide who can access which data.

Users can access the federation and benefit from its capabilities from any participating site. Figure 3 presents the architectural details of a federated site. Here, we distinguish two main components, namely resource manager and autonomic manager. The first one manages local resources, and elastically deploys agents to meet the computational requirements of the users. It also includes a monitoring system that collects status information of all federated resources. This information system can be used to announce the capabilities of the federation, to drive the execution of a user application, etc. On the other hand, the autonomic manager provides users with on-site autonomic capabilities. This component makes sure that user's application is executed within terms of the specified policies, and adapts the provisioned resources accordingly.

Federation sites interact with the rest of the federation though the federation management space in a publish/subscribe fashion. Each site publishes information about the status of its resources, the services they offer, or computational needs of its users. Additionally, each site creates subscriptions to be notified when there is some event of interest, such as for example that a user requests one of the offered services. Alternatively, the federation site is also able to work using a push/pull model.

In our approach a federation is created dynamically in a collaborative way, where CometCloud enables sites to talk to each other to identify themselves, negotiate the terms of adhesion, discover available resources, and advertise their own resources and capabilities. This requires minimal configuration at each site to specify the available resources, queueing system or type of cloud, and credentials. The federated management space is dynamically created at runtime, and sites can join and leave at any point. We note that as a part of the adhesion negotiation, sites may have to verify their identities using security mechanisms such as X.509 certificates, public/private key authentication, or others. Finally, the coordination between the execution spaces and the federation sites is transparently managed by the infrastructure based on the programming model chosen by the user.

4. EXPERIMENTAL SETUP

In order to perform the experiment, the end-user federated 10 different resources, provided by six institutions from three countries. We wish to stress that all elements of the experiment, e.g. creation of the federation, deployment of the CometCloud middleware and the simulation software, as well as the execution of computations, were performed completely by operating within limits imposed by the shell account of the end-user. Only `SSH` access was utilized, and no special privileges were required. This approach was also employed to guarantee the security of the federation, i.e. we leveraged only the existing `SSH` infrastructure. Table 1 provides a summary of resources used during the experiment. As can be seen, aggregated resources span different hardware architectures and queuing systems, ranging from the high-end supercomputers to small-scale servers.

We integrated the MPI-based solver, described in Section 2, with the CometCloud-based federation infrastructure using the Master/Worker paradigm. The master component takes care of generating tasks, collecting results, verifying that all tasks executed properly, and keeping log of the execution. To prevent the master from becoming a bottleneck, CometCloud allows the creation of a hierarchy of masters to distribute the request load. Here, each task is described by a simulation configuration (specific values of the input variables), and minimal hardware requirements. All tasks are automatically placed in the CometCloud-managed distributed task space for execution. In the proposed approach, workers' sole responsibility is to execute tasks pulled from the task space. To achieve this, each worker interacts with the respective queuing system and the native MPI library via a set of dedicated drivers implemented as simple shell scripts.

Fault-tolerance mechanisms are in place to handle failed tasks. First, the master recognizes the source of the error and then either directly resubmits the failed task as is (in case of a hardware error or a resource leaving the federation), or regenerates it after first increasing the minimal hardware requirements and/or modifying solver parameters (in case of an application error and/or insufficient resources).

To interrogate the parameter space at the precision level satisfactory to the end-user we identified 12,400 simulations (tasks) as essential. 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 al-

Table 1: Computational resources used to execute the experiment and their capability.

Name	Provider	Type	Cores†	Memory‡	Network	Scheduler	Cores/task	Accepted tasks
Excalibur	RDI^2	IBM BG/P	8,192	512 MB	BG/P	LoadLeveler	1024	S/M/L
Snake	RDI^2	Linux SMP	64	2 GB	N/A	N/A	64	S/M
Stampede	XSEDE	iDataPlex	1,024	4 GB	IB	SLURM	128	S/M/L
Lonestar	XSEDE	iDataPlex	480	2 GB	IB	SGE	120	S/M/L
Hotel	FutureGrid	iDataPlex	256	4 GB	IB	Torque	128	S/M/L
India	FutureGrid	iDataPlex	256	3 GB	IB	Torque	128	S/M/L
Sierra	FutureGrid	iDataPlex	256	4 GB	IB	Torque	128	S/M/L
Carver	DOE/NERSC	iDataPlex	512	4 GB	IB	Torque	256	S/M
Hermes	UCLM, Spain	Beowulf	256	4 GB	10 GbE	SGE	128	S/M/L
Libra	IHPC, Singapore	Beowulf	128	8 GB	1 GbE	N/A	128	S/M

Note: † – peak number of cores available to the experiment. ‡ – memory per core. S – small, M – medium, and L – large.

ready 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. Depending on the hardware characteristics different machines accepted tasks from different classes (see Table 1). This was achieved by providing a simple configuration file to the respective workers and helped to organize the tasks so as to achieve a better load balance (big tasks at the start and small tasks at the end). Although the task 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.

5. 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 4.

The initial configuration of the federation included only five machines (*Excalibur, Snake, Stampede, 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 4 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 requirements of the application and the availability of resources, federation can be scaled up or down accordingly.

Figure 5 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 4). 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. 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.

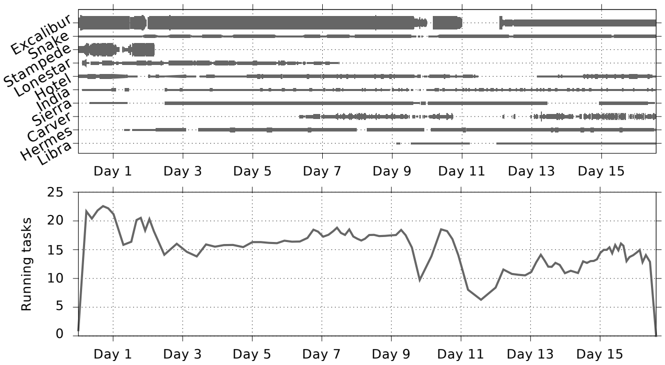

Figure 4: Summary of the experiment. Top: Utilization of different computational resources. Line thickness is proportional to the number of tasks being executed at given point of time. Gaps correspond to idle time, e.g. due to machine maintenance. Bottom: The total number of running tasks at given point of time.

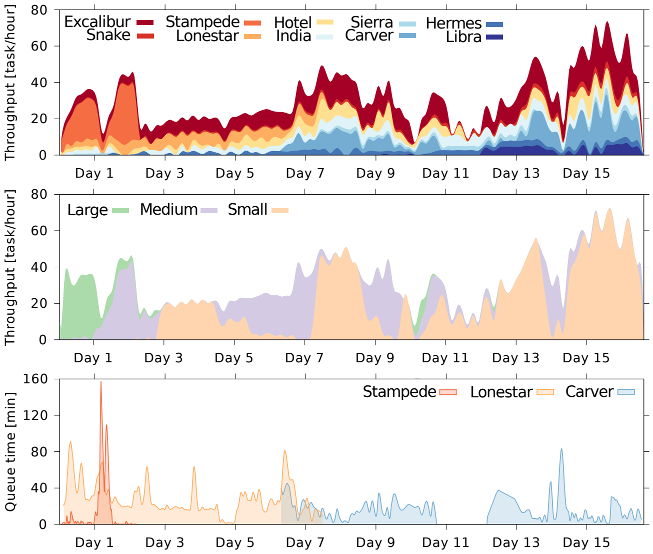

Figure 5: Throughput and queue waiting time. Top: Dissection of throughput measured as the number of tasks completed per hour. Different colors represent component throughput of different machines. Middle: Throughput contribution by different task classes. Bottom: Queue waiting time on selected resources. Please view in color.

6. LESSONS LEARNED

The results presented in the previous section clearly demonstrate feasibility and capability of an elastic, dynamically federated platform. 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.

Currently, the majority of investigators with large computational demands have access to multiple HPC systems. In order to take the advantage of the collective power these systems offer, we need a way to make these systems work together. The classic idea of a federation is to deploy an infrastructure that cannot be altered by a user. Our results show that a user-oriented approach, where a user can adjust the federation on-demand and on-the-fly is a better fit for the dynamic heterogeneous computational problems. In our approach a user is empowered with a simple mechanism to quickly federate resources as they become available, and without a third-party intervention. As a result, the federation is much more robust in responding to the changing computational demand (e.g. in our test we federated additional resources after the second day of the experiment to meet the assumed deadline). A natural extension of this concept, is to offer resources as a service, hiding their actual location and architecture. In this way, a user could interact with a federation via familiar and intuitive programming abstraction, which ultimately would improve usability. Currently, our solution provides a unified programming abstraction, e.g. in the experiment we used the Master/Worker model, and we continue to work on the Resource-as-a-Service capability.

One important element that contributed to the success of the experiment, was the ability of the federation to scale across institutional and geographic boundaries. Oftentimes, a single resource is not sufficient to execute a given scientific workload (e.g. because it is of limited scale, or it mismatches application requirements). Moreover, in the majority of cases predicting computational and storage requirements is difficult or impossible. Therefore, scaling up/down or out as needed becomes essential for dynamic workloads. Our results show that elasticity makes the infrastructure resilient to changes in the federation. Consequently, the federation is able to better sustain computational throughput.

The last remark regards the heterogeneity of the federation. Having highly heterogeneous resources as a part of the federation, it is crucial to take advantage of their particular characteristics and optimize resources allocation. This is synergistic with the concept of autonomic computing. In our experiment different resources were assigned different types of tasks depending on their computational power. This enabled us to use powerful supercomputers for large tasks, and smaller systems for the remaining tasks. This significantly contributed to the overall throughput of the experiment.

Based on the above observations, we believe that to tackle large scale problems with very dynamic computational demands a federation must ensure: i) extreme ease of deployment, ii) on-demand resource provisioning, elasticity and resilience, iii) sustainability of computational throughput, iv) strong fault-tolerance guarantees for tasks and resources, v) efficient use of the existing infrastructure capabilities, and vi) data security.

7. RELATED WORK

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, 10, 20, 21]. For example, volunteer computing systems (e.g. BOINC) enable end-user resources provided by a crowd of volunteers to be aggregated to provide 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, Grid'5000, OSG) have targeted more compute/data intensive applications by federating capacity and/or capabilities into secure and dependable virtual organizations. However, grids often have user-perceived complexity, and configuring them involves complex software-hardware infrastructure requiring significant experience from the end-users.

Projects like XSEDE, PRACE, and Grid'5000 developed successful means of sharing large-scale HPC resources. However, these projects are focused on delivering loosely coupled computational and storage capability, and do not implement any specific solution to enable seamless access to the federated resources. Efforts such as MyCluster [25] allow the aggregation of resources from different clusters. Among grid federation efforts we can find InterGrid [7] and LA-Grid meta-scheduling [4] that promote interlinking of different grid systems through peering agreements [22], and GridWay [23] and Koala [17] with the use of delegated matchmaking and grid meta-brokering [14]. Although grid technology significantly contributed to scientific projects, for example ALMA or LHC, it has intrinsic issues such as usability and relatively static pool of resources. In that sense, cloud computing leaps forward by offering resources on-demand, and creating the illusion of a unique unlimited pool of resources. Current cloud platforms can provide effective solution for certain classes of applications, as reported by several projects [6, 8, 13, 18]. There are also efforts exploring how to compose providers as a federated cloud environment [24], how to combine clouds with integrated computing infrastructures [5, 20], and testbeds such as FutureGrid and Open Cirrus that support research on cloud federation and resource management. Nevertheless, many HPC applications, for example tightly coupled MPI codes, cannot effectively take advantage of cloud infrastructures, due to the high network latency, which translates in the low performance and reliability [11]. Moreover, even for suitable applications, the cost and relative performance of cloud becomes a concern [6, 27].

8. CONCLUSIONS

In this paper, we focused on a class of MTC problems with dynamic and non-trivial computational requirements. Using a representative application of constructing phase diagram of possible flow behaviors in microscale devices, we performed a large-scale experiment to understand which features must be provided by a federation in order to efficiently support large scientific workloads. We developed a federation that enabled us to dynamically aggregate HPC resources, to solve the selected problem within acceptable time limits. Our results demonstrate great potential of elastic federation to tackle large-scale problems, that otherwise would require dedicated leadership-class systems.

9. ACKNOWLEDGMENTS

This work is supported in part by the NSF under grants ACI 1339036, ACI 1310283, IIP-0758566, ACI-1339036, DMS-0835436, CAREER-1149365 and PHY-0941576, and by IBM via OCR and Faculty awards. We used resources provided by: XSEDE NSF OCI-1053575, FutureGrid NSF OCI-0910812, and NERSC Center DOE DE-AC02-05CH11231. Authors would like to thank the research groups at UCLM, IHPC for providing access to their resources; Dr. O. Wodo, Dr. D. DiCarlo, M. Abdelbaky for their discussion and helpful comments; and P. Bisbal and K. Tanaka for their support.

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. Technical Report CCT-TR-2010-1, Center for Computation & Technology, Louisiana State University, 2010.

[2] H. Amini, E. Sollier, M. Masaeli, et al. Engineering fluid flow using sequenced microstructures. *Nature Communications*, 2013.

[3] F. Berman, G. Fox, and A. Hey. *Grid Computing: Making the Global Infrastructure a Reality.* John Wiley & Sons, 2003.

[4] N. Bobroff, L. Fong, S. Kalayci, et al. Enabling interoperability among meta-schedulers. In *Proc. CCGrid*, pages 306–315, 2008.

[5] A. Celesti, F. Tusa, M. Villari, et al. How to enhance cloud architectures to enable cross-federation. In *Proc. IEEE Int. Conf. on Cloud Computing (Cloud)*, pages 337–345, 2010.

[6] E. Deelman, G. Singh, M. Livny, et al. The cost of doing science on the cloud: the Montage example. In *Proc. SC*, 2008.

[7] M. Dias de Assuncao, R. Buyya, and S. Venugopal. InterGrid: a case for internetworking islands of grids. *Concurrency Computat. Pract. Exper.*, 20(8):997–1024, 2008.

[8] G. Fox and D. Gannon. Cloud programming paradigms for technical computing applications. In *Cloud Futures Wksp.*, 2012.

[9] Y. Gambin, V. VanDelinder, F. A., et al. Visualizing a one-way protein encounter complex by ultrafast single-molecule mixing. *Nature Methods*, 8:239–241, 2011.

[10] G. Garzoglio, T. Levshina, M. Rynge, et al. Supporting shared resource usage for a diverse user community: the OSG experience and lessons learned. *J. of Physics: Conf. Series*, 396, 2012.

[11] A. Iosup, S. Ostermann, M. Yigitbasi, et al. Performance analysis of cloud computing services for many-tasks scientific computing. *IEEE TPDS*, 22(6):931–945, 2011.

[12] D. S. Katz, T. G. Armstrong, Z. Zhang, M. Wilde, , and J. M. Wozniak. Many-task computing and blue waters. Technical Report CI-TR-13-0911, Computation Institute, University of Chicago & Argonne National Laboratory, 2012.

[13] K. Keahey and T. Freeman. Science clouds: Early experiences in cloud computing for scientific applications. In *Proc. Cloud Computing and Its Applications (CCA)*, 2008.

[14] A. Kertesz, I. Rodero, and F. Guim. BPDL: A data model for grid resource broker capabilities. Technical Report TR-0074, CoreGRID Inst. on Resource Management and Scheduling, 2007.

[15] H. Lee, J. Kim, H. Kim, et al. Colour-barcoded magnetic microparticles for multiplexed bioassays. *Nature Materials*, 9:745–749, 2010.

[16] Z. Li and M. Parashar. A computational infrastructure for grid-based asynchronous parallel applications. In *Proc. HPDC*, pages 229–230, 2007.

[17] H. Mohamed and D. Epema. KOALA: a co-allocating grid scheduler. *Concurrency Computat. Pract. Exper.*, 20:1851–1876, 2008.

[18] M. Parashar, M. AbdelBaky, I. Rodero, and A. Devarakonda. Cloud paradigms and practices for computational and data-enabled science and engineering. *Computing in Science & Engineering*, 15:10–18, 2013.

[19] I. Raicu, I. Foster, and Y. Zhao. Many-task computing for grids and supercomputers. In *Proc. Workshop on Many-Task Computing on Grids and Supercomputers*, pages 1–11, 2008.

[20] P. Riteau, M. Tsugawa, A. Matsunaga, et al. Large-scale cloud computing research: Sky computing on FutureGrid and Grid'5000. In *ERCIM News*, 2010.

[21] I. Rodero, F. Guim, J. Corbalan, and A. Goyeneche. The grid backfilling: a multi-site scheduling architecture with data mining prediction techniques. In *Grid Middleware and Services*, pages 137–152. Springer US, 2008.

[22] I. Rodero, D. Villegas, N. Bobroff, Y. Liu, L. Fong, and S. M. Sadjadi. Enabling interoperability among grid meta-schedulers. *J. Grid Comput.*, 11(2):311–336, 2013.

[23] T. Vazquez, E. Huedo, R. Montero, et al. Evaluation of a utility computing model based on the federation of grid infrastructures. In *Proc. Euro-Par Conf.*, pages 372–381, 2007.

[24] D. Villegas, N. Bobroff, I. Rodero, et al. Cloud federation in a layered service model. *J. of Computer and System Sciences*, 78(5):1330–1344, 2012.

[25] E. Walker. Continuous adaptation for high performance throughput computing across distributed clusters. In *Proc. of IEEE Cluster 2008*, 2008.

[26] J. Wang, Y. Zhan, V. Ugaz, et al. Vortex-assisted DNA delivery. *Lab on a Chip*, 10:2057–2061, 2010.

[27] K. Yelick, S. Coghlan, B. Draney, et al. The Magellan report on cloud computing for science. Technical report, U.S. DoE Office of Advanced Scientific Computing Research (ASCR), 2011.

[28] CometCloud Project. http://www.cometcloud.org/.