Exploring HPC-based Scientific Software as a Service using CometCloud

Moustafa AbdelBaky[1], Javier Diaz-Montes[1], Michael Johnston[2], Vipin Sachdeva[2], Richard L. Anderson[3], Kirk E. Jordan[2], and Manish Parashar[1]
[1]Rutgers Discovery Informatics Institute (RDI[2]), Rutgers University, Piscataway, NJ, USA
[2]IBM TJ Watson Research Center, Cambridge, MA, USA
[3] Science and Technology Facilities Council, Hartree Centre, SciTech Daresbury, Warrington, UK
moustafa.a@rutgers.edu, javidiaz@rdi2.rutgers.edu, michaelj@ie.ibm.com, vsachde@us.ibm.com, richard.anderson@stfc.ac.uk, kjordan@us.ibm.com, parashar@rutgers.edu

Abstract— **The use of *in-silico* simulations in experimental science can greatly increase laboratory efficiency and provide additional insights into interactions not easily described by traditional methods. Such simulations require significant amounts of computational resources, accessible only via supercomputers of large-scale high-performance clusters. Due to the complexity of the computational experiments, as well as the usage of the underlying resources, experimental scientists heavily rely on computational scientists with HPC expertise to perform these simulations. This additional bottleneck prevents the widespread adoption of real time in-silico simulation as a driver for laboratory experimentation. In this paper, we aim to overcome this bottleneck by presenting the architecture of an end-to-end framework to enable HPC Software as a Service. This framework is designed to make it easy for scientific applications to run on top of dynamically federated HPC resources. The framework enables HPC resource sharing while maximizing throughput and utilization. We focus specifically on a use case where an experimental scientist uses a mobile portal to control dissipative particle dynamics experiments that are executed on a remote supercomputer (IBM Blue Gene/Q).**

Keywords-software as a service; science as a service; scientific computing; dissipative particle dymanics

I. INTRODUCTION

Historically, scientific discovery via laboratory experimentation was a time-consuming and expensive process. However, recent advancements in computer simulation have proven that many of these laboratory experiments can be replicated *in-silico* using analogous computational algorithms. Some advantages to the use of computational methods is that a wider variety of combinations can be explored than can be created in a laboratory, environmental factors and errors can be completely contained, and the time taken for processes or reactions to occur can be adjusted by the computer user. Thus, modern day scientific discovery can benefit from a combination of both methods, wherein in-silico experiments serve as drivers for real-world laboratory experiments, saving time and reducing product waste.

For example, drug discovery often requires testing a large number of protein-ligand binding interactions to identify new potential drug candidates. These interactions can occur over different time intervals, and in-lab experiments can take months to show results. On the other hand, computational methods can be used to narrow down this pool of molecular candidates by reporting which simulation conditions yield the desired binding behavior, thus reducing the number of in-lab experiments that the scientist must perform, and ideally, the overall time to discover new binding interactions.

A number of conditions must be met in order for in-silico experiments to be usable in hypothesis testing and decision support roles. Firstly, they must be substantially cheaper and/or quicker (in time-to-solution) than their real-world counterparts. Second, the variance and bias of the experiments must be such that experimentalists can trust the in-silico results and use them for guidance. Finally they must be easily accessible and usable by experimentalists.

Experience shows that usability is a major barrier that prevents experimental scientists from using computational-based techniques, because of complexities related to the design and execution of in-silico experiments. In order to design an in-silico version of a desired laboratory experiment, simulation-specific expertise is required to package computational components in such a way that mimics the conditions of the laboratory. For example, a typical workflow for computational simulation involves many stages: (i) the simulation requires input parameters that must be tuned and adjusted to a particular experiment, (ii) after the simulation has completed, post-processing tools must be used to organize the simulation output into meaningful results, and (iii) various statistical methods may be employed to aggregate results from multiple simulations. In addition, simulation of such complex systems generally necessitates the use of high-performance computing (HPC) resources, which requires the scientist to have another level of computational expertise.

As a result of these complexities, an experimental scientist typically outsources the task of running an in-silico experiment to a computational collaborator. However, in many cases, this method incurs its own bottlenecks, in that the experimental scientist must explain the laboratory experiment to the computational scientist and wait for the computational scientist to design and run the in-silico experiment. When the

COLLABORATECOM 2014, October 22-25, Miami, United States

DOI 10.4108/icst.collaboratecom.2014.257833

experimental scientist receives the post-processed results of the simulation, he or she may realize that the desired solution was not reached or that the experiment was misinterpreted by the computational chemist designing the experiment, which would require the simulation to be fixed and re-run. Additionally, the computational chemists often design each experiment in an ad-hoc manner that varies from experiment to experiment, leaving little room for reusability or automation. For these reasons, the present process is unsuccessful in producing the desired timesaving and increased efficiency needed for its widespread adoption. Hence, new models and techniques need to be explored to address this issue.

One such technique that has been proposed to address such issues is to explore the use of computational resources that do not require new knowledge or expertise. As a result, many users in the scientific community have looked into using clouds for scientific applications [5] in order to alleviate some of these HPC-related issues [7]. While initial experiments have been successful for some classes of scientific applications (e.g., loosely coupled or embarrassingly parallel) [6], performance remains an issue that prevents widespread of cloud computing in the HPC domain [8]. For this reason, in our previous work [9], we argue that instead of moving scientific applications to typical cloud platforms (i.e., virtualized), we should focus on creating cloud abstractions that expose existing bare-metal HPC resources using cloud capabilities (e.g., on-demand provisioning, elasticity). In this paper, we extend these ideas to provide pervasive access to high-end resources in a collaborative manner, and propose a complete Software as a Service (SaaS) [10] abstraction that allows scientists to easily perform in-silico experiments on geographically distributed HPC resources. Our approach has three main advantages: (i) users are presented with a domain specific interface where they can focus exclusively on the science aspect of the experiments; (ii) bare-metal resources are transparently allocated on demand to efficiently execute the required scientific experiments and minimize performance degradation; and (iii) these resources are abstracted to the computational scientist at a high level to facilitate the deployment of the actual application. These characteristics are demonstrated in this paper using a driving scenario where experimental chemists use mobile devices to perform Dissipative Particle Dynamics (DPD) [1] experiments on high-end HPC resources.

The remainder of this paper is organized as follows: Section II discusses related work in providing HPC Software as a Service. Section III presents a background on the CometCloud framework, while Section IV provides the details of the system components and architecture for providing HPC software as a service. Section V presents an overview of the DPD Simulation application and its requirements as well as the implementation of the use case. An evaluation and analysis of the proposed framework is then presented in Section VI using the DPD Simulation application running on an IBM Blue Gene/Q. The paper concludes in Section VII outlining future work.

II. Related Work

The majority of research for using cloud computing in science and engineering applications has been focused at the Infrastructure as a Service (IaaS) layer [8][15][18]. At higher levels there have been some successful attempts on providing scientific workflows as a service [11][12], the goal of this work is to provide tools for building and deploying scientific workflows on cloud infrastructure. For example, Neptune [13] provides a domain specific language for deploying HPC software on cloud platforms. Additionally, some researchers have looked into providing scientific data as a service [14], where scientific data from previous experiments are stored on cloud services and can be easily shared and accessed through simple portals. At the software layer, Opal2 [16] was proposed as a toolkit that wraps scientific applications as web services. Once an application is deployed, the toolkit provides mechanisms for launching and monitoring jobs on grid, clusters, or cloud resources. Finally, ASAAS [17] was proposed as a SaaS implementation for HPC applications. The main services of ASAAS include automatic encapsulation and deployment service, web services portal for the usage, on-demand license service and accounting.

Our proposed work is different from existing research in two ways. First our work uses a high-end supercomputer (IBM Blue Gene/Q) that is provisioned and accessed in a similar fashion to cloud resources as opposed to running on virtualized cloud resources, which does not provide the high performance necessary for tightly coupled applications [8]. Second, the existing work is focused exposing HPC applications by providing portals for launching and monitoring jobs. While our proposed work also provides mechanisms for launching and monitoring jobs, our focus is to deliver actual application functionalities to non-computational scientists to bridge the gap between their scientific domain knowledge and the use of advanced computing techniques. Using a simple mobile device (e.g., a tablet), these scientists gain access to a service that allows them to design experiments, modify their parameters, execute them, and visualize their results without any knowledge about the underlying computational tools and infrastructure. At the back-end, computational scientists design and deploy computational tools (e.g., Dissipative Particle Dynamics (DPD)) that simulate real-world phenomena (e.g., wet-lab experiments). The provided framework translates experimentalists' inputs into computational inputs and deploys the required experiments on supercomputers, maximizing throughput and utilization. A further aspect of this approach is that it allows the computational experts to ensure simulations are carried out within the regime for which they are reliable. To the best of our knowledge there is no research that is focused on providing a complete HPC software as a service similar to the one presented in this paper.

III. Background: CometCloud

CometCloud is an autonomic framework for enabling real-world applications on software-defined federated cyber-infrastructure, including hybrid infrastructures integrating public & private Clouds, data-centers and Grids. The overarching goal of CometCloud is to realize a software-defined federation with cloud abstractions that offer resources in an elastic and on-demand manner. It also provides

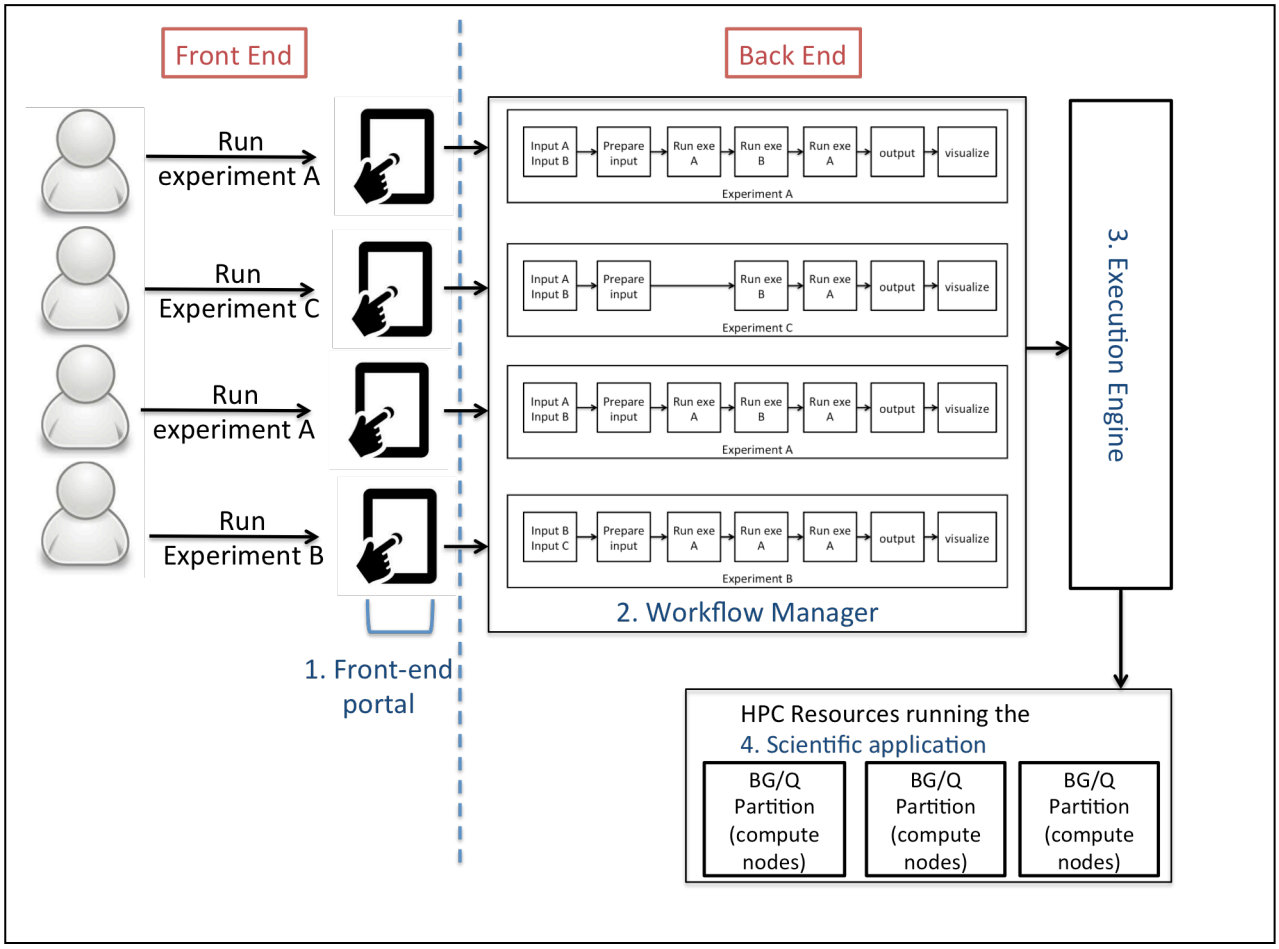

Figure 1: Overall Framework Architecture and System Components

abstractions and mechanisms to support a range of programming paradigms and application requirements on top of the federation [19].

Conceptually, CometCloud is composed of a programming layer, service layer, and infrastructure layer. The infrastructure layer uses the Chord self-organizing overlay [20] and the Squid [21] information discovery to create a scalable content-based coordination space for wide-area and a content-based routing substrate, respectively. The routing engine supports flexible content-based routing and complex querying using partial keywords, wildcards, or ranges. It also guarantees that all peer nodes with data elements that match a query/message will be located. The service layer provides a range of services to support autonomics at the programming and application level. This layer supports a Linda-like [22] tuple space coordination model, and provides a virtual shared-space abstraction as well as associative access primitives. Dynamically constructed transient spaces are also supported to allow applications to explicitly exploit context locality to improve system performance. Asynchronous (publish/subscribe) messaging and event services are also provided by this layer. The programming layer provides the basic functionality for application development and management. It supports a range of paradigms including the master/worker/BOT. Masters generate tasks and workers consume them. Masters and workers can communicate via virtual shared space or using a direct connection. Scheduling and monitoring of tasks are supported by the application framework. The task consistency service handles lost/failed tasks. CometCloud is not restricted to applications that have been developed in a particular programming language (e.g. Java), and has been demonstrated to work as a wrapper for a number of other languages (e.g. C, Matlab, Fortran, etc.) [18].

IV. System Components & Architecture

The main goal of this paper is to present the design of an end-to-end framework to enable HPC Software as a Service. Providing an end-to-end simulation as a service presents many challenges, some of which are administrative (e.g., access, accounting, billing, etc.), while others are more technical (e.g., scheduling, security, etc.). The focus of our work is on the technical challenges and how the proposed framework addresses these challenges. The technical challenges can be divided into two groups: front-end and back-end. Some of the challenges at the front-end are: what is the right mechanism for delivering the application, what is the right interface for the end-user scientist, how do we expose different experiments designed by the computational scientists, and how to securely connect the front and back-ends. At the back-end, some of the challenges are: how to run/monitor the different simulations on the given system, how to handle system level failure, how to

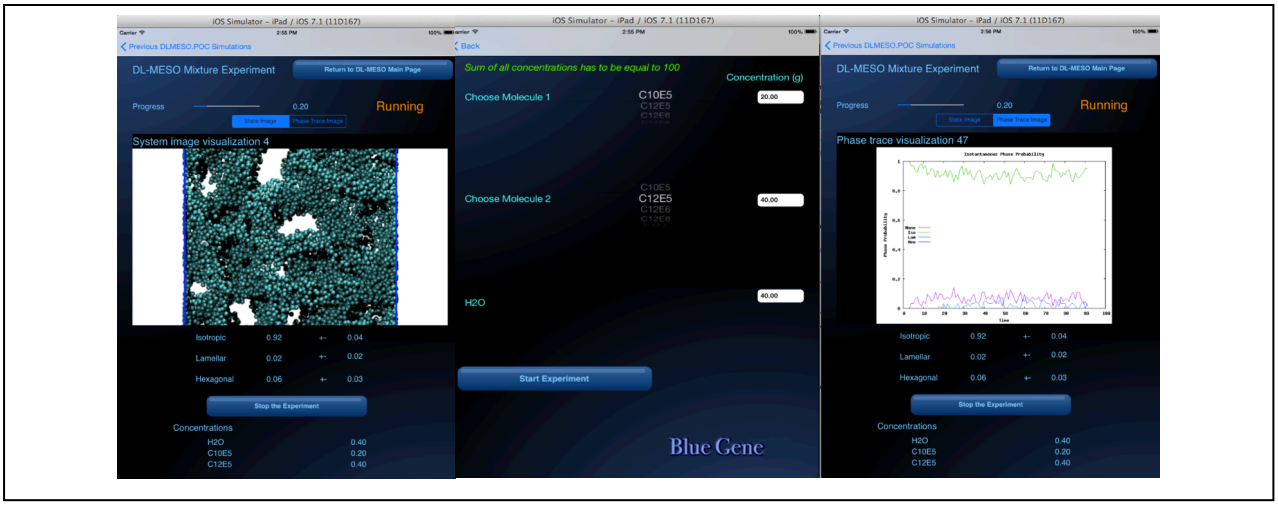

Figure 2: Front-end Portal Running the DPD Application

visualize results and send them back to the front-end, how to support multiple simulations/experiments from (a) single/multiple user(s), and how do we support scaling and provide high utilizations. Additionally, given that the Blue Gene/Q system used for our experiments is a production system running a queue-based scheduling software (IBM LoadLeveler[1]), what are the best ways to interact with the scheduler in order to minimize wait time and maximize throughput?

In order to address these challenges, we propose the framework depicted in Figure 1. This framework is composed of the following system components:

1. A front-end portal: the interface to the end-user, which communicates securely with the back-end.
2. Workflow manager: the manager responsible for the design of different scientific experiments and workflows.
3. Execution engine: the engine responsible for running the resulting experiments or workflows on the available HPC resources.
4. Scientific simulation application: the application running on the compute nodes of the HPC system.

In order to provide a better understanding of the applicability of our framework and the function of each one of the system components, we have used a driving use case in the context of dissipative particle dynamics.

V. Use Case: Dissipative Particle Dynamics (DPD)

DPD [1][2] is a particle-based simulation technique, which focuses on the meso-scale behavior of molecular systems. Meso-scale can be defined as the time and length scales that lie between the atomic-world (10^{-10} m) and the macro-world (10^{-5} m) in which we exist. These time and size scales are expensive and difficult to access using atomistic methods like Molecular Dynamics but with DPD they can be simulated at much less computational cost [3]. This allows the microstructure and hydrodynamics of various soft-matter systems (not completely liquid or solid) to be investigated. Formulation chemistry deals with the behavior and characteristics of mixtures of molecules that do not react with each other, but whose interactions yield a range of different behaviors. It is important in the manufacturing of a wide-range of household products, for example detergents and shampoos. DPD has proven effective at describing the structure and behavior of a variety of systems in formulation chemistry and a number of experiments formulation chemists carry out in the lab can be replicated in-silico using this method. However, using these computational tools and the underlying resources is not trivial for experimentalists. Hence, in this section we show how we can use our approach to ease this undertaking.

A. *Front-end Portal*

In our approach to deliver this application to the end-user, we focus on two key design points: accessibility and usability. By definition, a SaaS solution encapsulates the provisioning of software over the Internet [10]. Similarly, our front-end portal shown in Figure 2 uses Internet connectivity to communicate with the back-end, however, instead of providing a web-based application we opted to build a standalone application that can offer more customizable features. To provide ease of access, the application was designed for tablets. Ease of access enables the experimental chemists to run the simulations without having to go back and forth between a computer lab and the wet-lab. Because of the nature of wet-labs, a tablet proves to be a more accessible and portable choice than a standalone desktop or laptop computer, as it can be easily carried around to multiple workbenches and can be protected from the environment. To address usability, computational chemists and experimental chemists are providing input and feedback on this work.

In addition to these design points, a team of computational scientists designed and created end-to-end experiments that simulate important wet-lab experiments using the DL_MESO code [4], so that the experimental chemists could simply adjust the meaningful wet-lab parameters in order to visualize simulation results with minimal overhead. The experimental chemists were not exposed to the technical aspects of the computational experiment design. Similarly, the HPC resources

[1] http://www-03.ibm.com/software/products/en/tivoliworkloadschedulerloadleveler

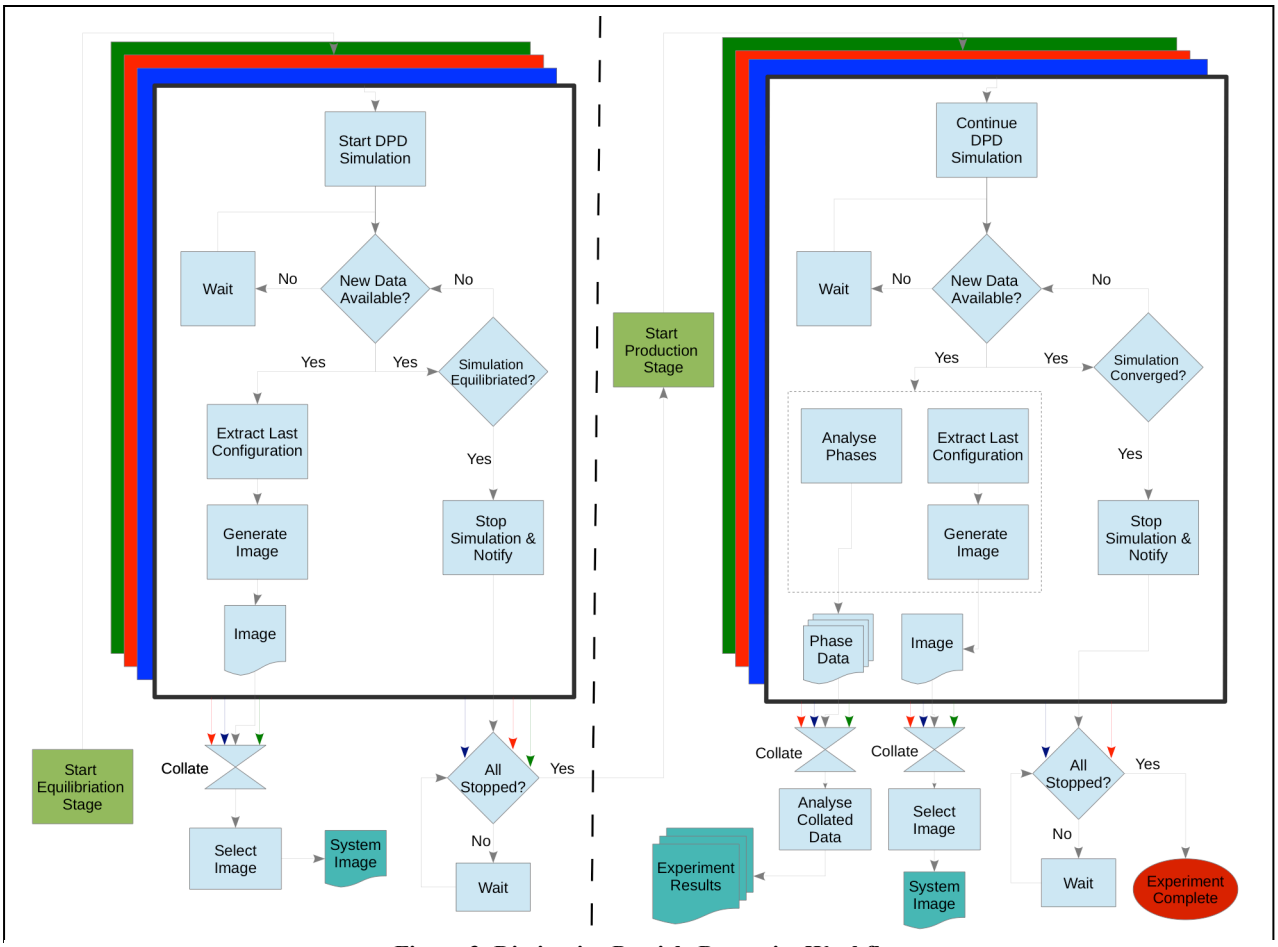

Figure 3: Dissipative Particle Dynamics Workflow

are completely masked from the experimental chemists to simplify the process and help the chemists focus on the end results.

B. *Workflow Manager*

The workflow manager is responsible for the development and deployment of the different experiments (workflows). In this use case we use a DPD Workflow Manager. The manager creates the workflow shown in **Error! Reference source not found.** and tracks its progress, state e.g. running, waiting on resource, and output of the various jobs. Specifically we consider the prediction of the structure of a mixture (also known as phase, which can yield information about system properties, e.g., rheology) of interacting molecules selected by the chemist. The inputs provided by the chemist are two molecules (selected from a library of molecules relevant to the experimentalists field of study). The amount is specified in grams where the sum of the weight of the two molecules is <= 100g. Water is added to make the total weight equal to 100g. These are measurements that the experimental chemist would usually use in the wet-lab.

The output from the experiment is a prediction of the phase the mixture will form along with confidence. The workflow consists of three stages: preparation, equilibration and production.

Preparation stage: in the preparation stage the inputs for four different DPD simulations of the mixture are generated from the inputs. The output of a DPD simulation consists of a sequence of frames, or snapshots, of the system configuration. Physical characteristics of the mixture are calculated by performing statistical averaging over all the frames produced by the simulation. Four simulations are launched in order to obtain better statistics.

Equilibration stage: the equilibration stage is required because the DPD simulations are started in random configurations. The simulations must evolve from these initial configurations to those that are more representative of the systems behavior. Technically the probability of seeing a configuration with a given energy must converge to a given distribution (the equilibrium distribution). In the equilibration phase the four DPD simulations are started. In addition a number of further jobs are launched:

- Equilibration Monitors: An equilibration job is launched periodically on the output of each simulation to determine if equilibration has been reached. When all four simulations have equilibrated the stage is finished.
- Structure Extraction: A component is periodically launched to extract the latest structure in one of the simulations

- Image generation: A component is launched to convert the output of the structure extraction into a PNG image that can be displayed to the chemist

Production stage: in the production stage the four DPD simulations are continued. Again a number of further jobs are launched to monitor and process the output of the simulations. These include the Structure Extraction and Image Generation steps above. In addition, the following jobs are employed in the workflow:

- Phase Determination: For each simulation a phase-determination program is periodically launched to determine the structure of phases present in each frame output since last launch (and after equilibration)
- Output Aggregation: A job is periodically launched consuming the output of the Phase Determination jobs. This job aggregates the output from the four simulations
- Phase Processing: A job is periodically launched consuming the aggregated output and calculating the phase present in the simulation and the confidence value. These values are displayed to the user.

C. *Execution Engine*

CometCloud is responsible for executing different simulations on the Blue Gene/Q system. The DPD Workflow Manager generates different experiments and interacts with CometCloud to execute these experiments. CometCloud then interacts with LoadLeveler (the scheduler running on the Blue Gene/Q) to submit and run these jobs. The addition of the CometCloud layer between the DPD simulation workflow manager and LoadLeveler adds extra overhead, however, there are a number of added benefits to using the CometCloud framework, which are discussed below.

CometCloud exposes the hardware as a black box component and guarantees the execution of the workflow specified by the DPD Workflow Manager. CometCloud is also responsible for monitoring the submitted runs (jobs), handling system level failures, and resubmitting failed jobs. In addition, CometCloud handles the interaction with LoadLeveler and can mitigate common queue problems, such as long queue times as well as fixed partition sizes. For example, the smallest partition size a user can request on this specific installation of Blue Gene/Q is 128 nodes (2048 cores). This is due to the limited number of available I/O nodes on the system (each partition requires at least one I/O node). Therefore, running jobs smaller than 2048 cores results in wasted CPUs that are just idling. With our design, we were capable of packing multiple smaller jobs per partition to increase utilization, all of which is transparent to the application and user. In summary, CometCloud abstracts the hardware from the computational scientists to enable them to focus on designing experiments and developing meaningful science while providing them with an execution engine that will deploy the application in real-time on a federation of available resources while maximizing throughput and utilization.

The specific implementation of CometCloud is presented in Figure 4. In order to avoid any performance degradation on the compute nodes, CometCloud runs fully on the Blue Gene/Q front-end node. CometCloud is composed of the following components:

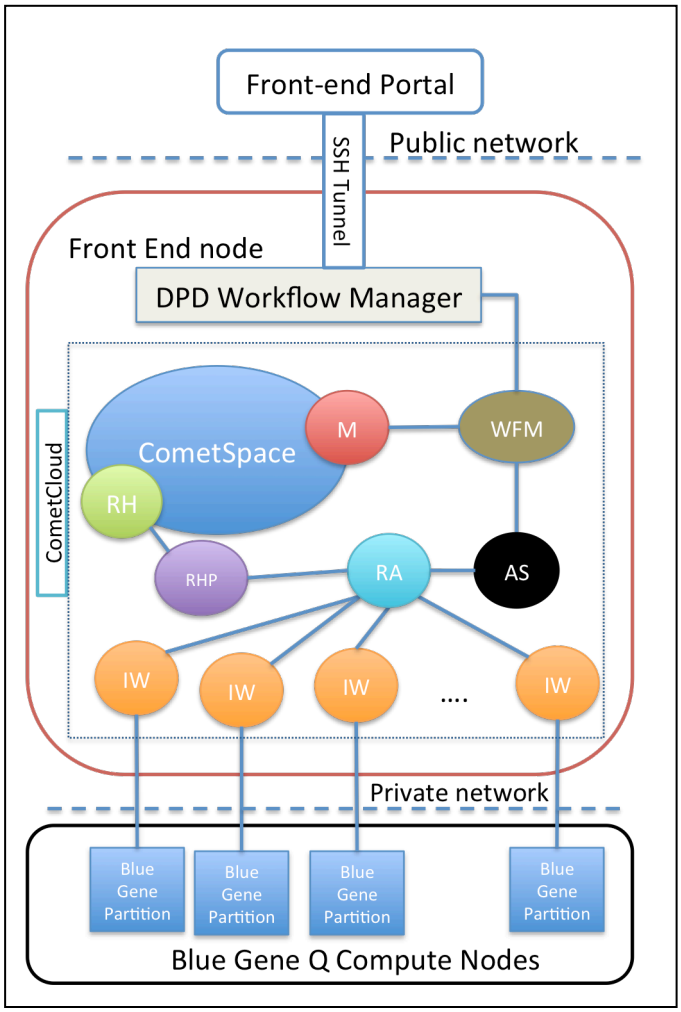

Figure 4: CometCloud-based Implementation

Comet Workflow Manager (WFM): The WFM is responsible for deploying the end-to-end workflow for a single simulation (passed from the DPD Workflow Manager). Each stage in the workflow is a set of jobs that can be executed. The WFM handles the dependency between stages. Stages can be executed in parallel if there is no dependency, and sequentially otherwise. Jobs within one stage have no dependency on one another.

CometSpace (CS): The CS is a shared space that contains all the jobs that are ready for execution. The CS can be distributed across multiple machines, however for this experiment, the CS is limited to the Blue Gene/Q front-end node. This was done to avoid any performance overhead on the compute nodes that run the DL_MESO application.

Master (M): The WFM communicates with the master when a workflow stage is ready to be executed. The master generates the jobs for the given stage and inserts them into the CometSpace. The master then notifies the WFM after the jobs are inserted in the space.

Isolated Workers (IW): The workers are responsible for executing the jobs on the allocated Blue Gene/Q partitions. Workers can submit the fetched jobs to LoadLeveler or execute directly using the *runjob* command. Workers communicate with their corresponding Resource Agent (to retrieve jobs).

Request Handler (RH): The RH fetches jobs from the CS

Request Handler Proxy (RHP): The RHP receives requests from the Resource Agent and submits them to the RH.

Resource Agent (RA): The RA is responsible for allocating/terminating Comet Workers as well as managing hardware resources (in our case partitions on the Blue Gene/Q). There exists one RA for each system; in this paper there is only one system and therefore one RA. Additionally, the agent is responsible for fetching jobs from the space through the RHP. This was done so that different RAs can fetch different types of jobs based on the type of the underlying system.

Autonomic Scheduler (AS): The AS is used to schedule jobs on available resources. The WFM notifies the AS when jobs are ready for execution and the number/size of these jobs. The RA notifies the AS of all available (active and inactive) resources for the given system. The AS then schedules the jobs according to this information and notifies the AS to adjust its running local resources if necessary (i.e. allocate more partitions, terminate some partitions, start new Comet Workers, terminate running workers).

D. Scientific Simulation Application

DL_MESO [4] is a meso-scale simulation package providing two distinct simulation methods to users: DPD and lattice Boltzmann equation methods, with the DPD program being of importance in this work. It is a hybrid parallel program capable of distributed memory parallelism via MPI and shared memory parallelism via OpenMP, and can scale to thousands of processors on high-performance supercomputers. The package also provides a number of post-processing tools. Of particular relevance here is the isosurface tool, which provides a prediction of the phase of a meso-scale DPD system. DL_MESO is developed by the Science & Technology Facilities Council at Daresbury Laboratory for the United Kingdom Collaborative Computational Project (CCP5).

VI. Evaluation & Analysis

We have performed a set of experiments to evaluate our approach. These experiments are aimed to show that our approach is able to enable HPC software as a service, where resources are dynamically allocated to optimize their usage. All of the following experiments ran on an IBM Blue Gene/Q at the Hartree Centre, Science & Technology Facilities Council in Daresbury, UK. For these experiments we have used an application instance that requires around five minutes of computation in 512 cores of the previously mentioned machine.

A. Evaluating CometCloud Overhead

In this experiment we evaluate the overhead of using the CometCloud framework as opposed to submitting jobs directly to LoadLeveler from the DPD Workflow Manager. There are three types of overheads when using CometCloud.

- Startup overhead: is the time needed to launch the CometCloud framework. This includes the time to start all components of CometCloud, which includes the Workflow Manager (WFM), the Autonomic Scheduler (AS), the CometSpace (CS), the Master (M), the Request Handler (RH), the Request Handler Proxy (RHP), and the Resource Agent (RA). Note - Workers (W) are started and terminated automatically when new jobs are submitted from the DPD Workflow Manager.
- Job submission overhead: is the time needed to submit a job from the DPD Workflow Manager to LoadLeveler. This includes the time to generate a task for the submitted job with the correct job parameters, inserting the task into the CometSpace (CS), starting a worker (W), fetching the task from the space, and submitting the job to LoadLeveler.
- Job status overhead: is the time needed to return the status of a complete job back to the DPD Workflow Manager. This time includes the Master (M) verifying the completion of the task and the termination of the Worker (W) thereafter. In case of a job failure, the Master reinserts the job back into the CometSpace (CS).

The startup overhead results are shown in Table 1

Startup overhead	WFM & AS	CS, M, RH, & RPH	RA
Time (sec)	0.359	1.801	0.9

Table 1: CometCloud-based implementation startup overhead

The job submission overhead results are shown in Figure 5, in the left column we submitted 8 tasks sequentially, thus using only 1 worker to execute 8 tasks (1 task at a time). In the right column, we submitted 8 tasks in parallel, thus using 8 workers to execute all tasks (8 tasks at the same time). Using 1 worker, the average job submission overhead is 1.877 seconds (minimum 1.458 seconds and maximum 2.718 seconds). Using 8 workers, the average job submission overhead is 1.621 seconds (minimum 1.385 seconds and maximum 2.249 seconds). This shows that the overall job submission overhead (1.786 seconds) does not vary significantly when increasing the number of workers.

The job status overhead results are also shown in Figure 5, As before, the left column shows the results of executing 8 tasks sequentially, while the right column shows the results of executing 8 tasks in parallel. We can observe that the average job status overhead in the sequential case is 1.803 seconds versus the 1.571 seconds that takes in the parallel case.

Figure 5: Job submission & job status overheads for the CometCloud-based implementation

B. *Evaluating Fault Tolerance*

In this experiment we simulate a system failure by externally canceling jobs from LoadLeveler. CometCloud handles system failures and reinserts the failed tasks to the CometSpace (CS) to be resubmitted to LoadLeveler. Without CometCloud, manual user intervention is necessary to resubmit failed jobs. Figure 6 shows the time to completion for a single job when no failure is present, when a failure occurs but no fault tolerance is in place, and when a failure occurs with fault tolerance. The average time to resubmit a job to LoadLeveler (including reinserting the task into the CometSpace and a Worker fetching the task from the space) is 0.27 seconds. Waiting time in LoadLeveler queue is not shown in Figure 5. Figure 7 shows the total number of completed jobs after 100 job submissions and 7 simulated failures.

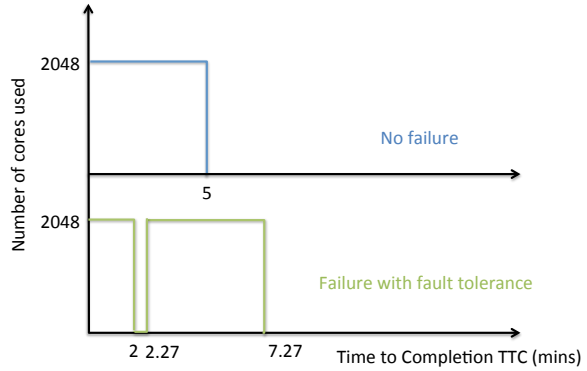

Figure 6: Application time to completion with fault tolerance

Figure 7: Number of complete jobs with fault tolerance

C. *Maximizing Utilization*

In this experiment we take advantage of CometCloud capabilities to maximize resource utilization. As we mentioned earlier, the minimum size of a Blue Gene/Q partition at Daresbury is 128 nodes (2048 cores). Submitting jobs that are of smaller size results in a waste of compute cycles. The simulations used in the DPD workflow stop scaling at 32 nodes (512 cores), which leads to a mismatch in resources.

In this experiment we examine the effect of packing 4 jobs (each of size 32 nodes (512 cores) into a single partition of size 128 nodes in two scenarios separately. The impact of packing on DL_MESO performance is measured by the average time per DPD step achieved over 40,000 steps.

In the first scenario, we compare executing 4 (512 core) jobs using 4 partitions simultaneously (no packing) versus using 1 partition (packing). Figure 8 shows that packing 4 jobs per partition causes a slow down in DL_MESO performance of 1.38% on average. This is attributed to the fact that the partition has only 1 I/O node and therefore I/O operations slow down the overall execution of the application. However the resource utilization in the case of no packing is 4 times less than with packing, see Figure 9.

In the second scenario, we compare packing vs. no packing when only 1 partition of size 128 nodes (2048 cores) is available to execute all 4 jobs. In this case, the jobs with no packing will be serialized and therefore packing jobs would result in almost four-fold (3.83) speedup, see Figure 10. The resource utilization in this scenario is the same as in the first scenario. The number of allocated resources with no packing will be 8196 cores (2048 cores allocated and released four times) but only 2048 cores will be used (512 cores used four times). When packing is used 2048 cores will be allocated and fully utilized once.

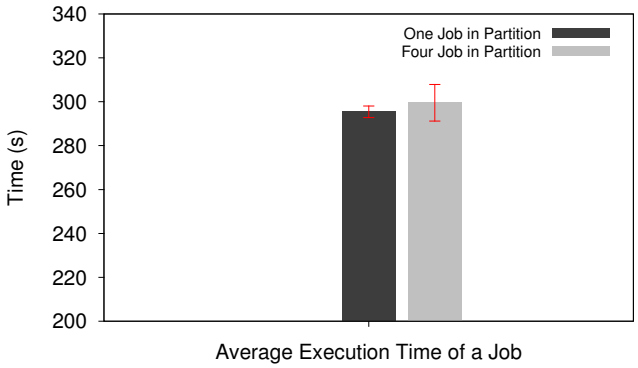

Figure 8: DL_MESO execution time per step in scenario 1

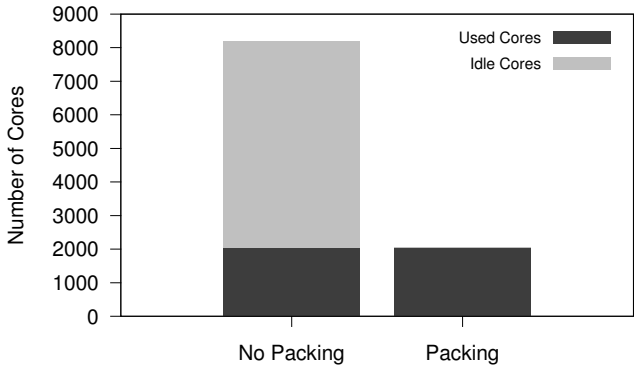

Figure 9: Resource utilization in Scenarios 1 and 2

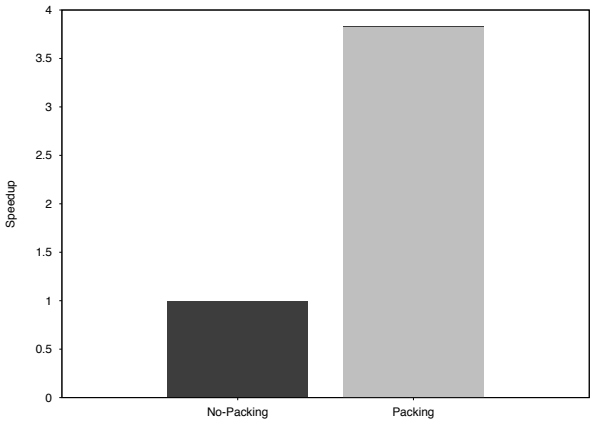

Figure 10: DL_MESO execution time per step in Scenario 2

D. Minimizing Queue Time

In this experiment we take advantage of CometCloud capabilities to minimize queue wait times. Without using CometCloud running 4 jobs involves submitting 4 jobs to LoadLeveler, which results in 4 wait_times (1 per job) in the queue and 4 run_times (also 1 per job). The wait_time varies significantly based on the overall utilization of the system whereas the run_time in this experiment is relatively fixed. CometCloud can pack multiple jobs per submission (as opposed to the previous experiment of multiple jobs per partition) and minimize the wait_time to one per multiple jobs, Note - run_time remains as 1 per job. This can be used to exploit the scheduling of LoadLeveler since the scheduling is based on fixed time allocations. For example, in the Daresbury installation, all jobs running for less than 20 minutes are considered of one class (e.g. small_class), whereas all jobs that are larger than 20 minutes but smaller than 1 hour are considered of a second class (e.g. medium_class), finally jobs that require more than 1 hour are considered of a third class (e.g. large_class). This results in that a similar wait_time for any job of the small_class whether it will run for 5 minutes or 20 minutes. CometCloud takes in consideration the policies set by LoadLeveler when packing multiple jobs into a single submission to try to minimize wait_time. For example, CometCloud can pack 4 (5 minutes) jobs into a single submission that will run in 20 minutes as opposed to 4 job submissions of 5 minutes each. This results in a reduction of wait_time by 75% percent. Note – this is only useful for heavily utilized systems.

In Summary – CometCloud can abstract high-end resources from the computational scientists while providing fault tolerance, high resource utilization, and minimal queue wait times. As we previously mentioned the CometCloud framework runs fully on the front-end node to avoid any performance degradation to the actual DPD application running on the compute nodes. The only performance degradation to the DL_MESO application occurs when we pack multiple smaller jobs into a partition. However, increasing the job throughput and the utilization of the partition compensates this performance degradation in this case.

VII. CONCLUSION & FUTURE WORK

In this paper, we presented the design and implementation of a framework that abstracts high-end resources from computational scientists in order to facilitate the development and deployment of DPD experiments. In addition, we provided a complete software stack coupled with high-end HPC resources that are provisioned to the end-user, the experimental chemist, using portable devices. In our approach the usability is a first class citizen, hence users are presented with a domain specific interface while the underlying environment details are completely hidden from the experimentalists. We show that the resulting framework enables HPC resource sharing with a small management overhead on the front-end node of the supercomputer, but without any performance degradation to the actual DPD application running on the compute-nodes.

Finally, the framework tries to maximize throughput and utilization of high-end resources with small performance degradation. Providing cloud abstractions for DPD simulations allows these experiments to be run directly by the experimentalists. In this model the computational experts create a workflow for the computational experiment that encapsulates both their state-of-the-art knowledge of the computational science and the HPC software involved. The experimentalists can then treat this easily accessible workflow as a black box that returns only the chemically relevant information to them. The increased accessibility of these simulations can help drive wet-lab experimentation in real-time. The service model enables higher utilization of high-end resources and facilitates the collaborative sharing of resources

Future work includes supporting federation of resources and elasticity. This includes running the workflow on multiple systems with different capabilities to optimize the execution. For example, visualization of the output would be run on an IBM iDataplex as opposed to running on the Blue Gene/Q system.

ACKNOWLEDGMENT

This work is supported in part by the NSF under OCI 1339036, OCI 1310283, OCI 1441376, IIP 0758566, and by IBM via OCR and Faculty awards. The authors would like to thank the Hartree Centre Science & Technology Facilities Council in Daresbury, UK for the access and allocations on the IBM Blue Gene/Q. M. AbdelBaky would like to thank IBM T. J. Watson for their support through the EPIC internship program.

REFERENCES

[1] P. J. Hoogerbrugge and J. Koelman, "Simulating Microscopic Hydrodynamic Phenomena with Dissipative Particle Dynamics," Europhysics Letters, 19, 155-160, (1992)

[2] P. B. Warren, "Dissipative particle dynamics," Current opinion in colloid & interface science 3.6 (1998): 620-624.

[3] R. D. Groot and P. B. Warren, "Dissipative particle dynamics: bridging the gap between atomistic and mesoscopic simulation," Journal of Chemical Physics, 107(11), 4423.

[4] M. A. Seaton, R. L. Anderson, S. Metz, and W. Smith, "DL_MESO: highly scalable mesoscale simulations," Molecular Simulation 39, no. 10 (2013): 796-821.

[5] C. Evangelinos and C. Hill, "Cloud computing for parallel scientific hpc applications: Feasibility of running coupled atmosphere-ocean climate models on amazon's EC2," In Cloud Computing and its Applications (CCA-08), Chicago, IL, USA, Oct 2008

[6] E. Deelman, G. Singh, M. Livny, B. Berriman, and J. Good, "The cost of doing science on the Cloud: the montage example," In SC '08: Proceedings of the 2008 ACM/IEEE conference on Supercomputing, pages 1–12, Piscataway, NJ, USA, 2008. IEEE Press

[7] K. Keahey, R. Figueiredo, J. Fortes, T. Freeman, and M. Tsugawa, "Science clouds: Early experiences in cloud computing for scientific applications," Cloud computing and applications 2008 (2008): 825-830.

[8] G. Fox and D. Gannon, "Cloud Programming Paradigms for Technical Computing Applications," technical report.

[9] M. AbdelBaky, M. Parashar, K. E. Jordan, H. Kim, H. Jamjoom, Z. Y. Shae, G. Pencheva et al. "Enabling high-performance computing as a service." Computer 45, no. 10 (2012): 72-80.

[10] P. Buxmann, T. Hess, and S. Lehmann, "Software as a Service," Wirtschaftsinformatik 50.6 (2008): 500-503.

[11] C. Hoffa, G. Mehta, T. Freeman, E. Deelman, K. Keahey, B. Berriman, , and J. Good, "On the use of cloud computing for scientific workflows," In eScience, 2008. eScience'08. IEEE Fourth International Conference on (pp. 640-645). IEEE.

[12] G. Juve and E. Deelman, "Scientific workflows and clouds," Crossroads 16, no. 3 (2010): 14-18.

[13] C. Bunch, N. Chohan, C. Krintz, and K. Shams, "Neptune: a domain specific language for deploying hpc software on cloud platforms," In Proceedings of the 2nd international workshop on Scientific cloud computing, pp. 59-68. ACM, 2011.

[14] B. Allen, J. Bresnahan, L. Childers, I. Foster, G. Kandaswamy, R. Kettimuthu, J. Kordas et al., "Software as a service for data scientists," Communications of the ACM 55, no. 2 (2012): 81-88.

[15] L. Wang, J. Tao, M. Kunze, A. C. Castellanos, D. Kramer, and W. Karl, "Scientific Cloud Computing: Early Definition and Experience," In HPCC, vol. 8, pp. 825-830. 2008.

[16] S. Krishnan, L. Clementi, J. Ren, P. Papadopoulos, and W. Li, "Design and evaluation of opal2: A toolkit for scientific software as a service," In Services-I, 2009 World Conference on, pp. 709-716. IEEE, 2009.

[17] Z. Hou, X. Zhou, J. Gu, Y. Wang, and T. Zhao, "ASAAS: Application software as a service for high performance cloud computing," In High Performance Computing and Communications (HPCC), 2010 12th IEEE International Conference on, pp. 156-163. IEEE, 2010.

[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, no. 4 (2013): 10-18.

[19] H. Kim, and M. Parashar, "CometCloud: An autonomic cloud engine," Cloud Computing: Principles and Paradigms (2011): 275-297.

[20] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications," 2001, pp. 149–160.

[21] C. Schmidt and M. Parashar, "Squid: Enabling search in dht-based systems," *J. Parallel Distrib. Comput.*, vol. 68, no. 7, pp. 962–975, 2008.

[22] N. Carriero and D. Gelernter, "Linda in context," *Commun. ACM*, vol. 32, no. 4, pp. 444–458, 1989.