Converting A High Performance Application to an Elastic Cloud Application

Dinesh Rajan, Anthony Canino, Jesus A Izaguirre, and Douglas Thain
Department of Computer Science and Engineering
University of Notre Dame
Notre Dame, Indiana 46556
Email: dpandiar@nd.edu, acanino@nd.edu, izaguirr@nd.edu, dthain@nd.edu

***Abstract*—Over the past decade, high performance applications have embraced parallel programming and computing models. While parallel computing offers advantages such as good utilization of dedicated hardware resources, it also has several drawbacks such as poor fault-tolerance, scalability, and ability to harness available resources during run-time. The advent of cloud computing presents a viable and promising alternative to parallel computing because of its advantages in offering a distributed computing model. In this work, we establish directives that serve as guidelines for the design and implementation or identification of a suitable cloud computing framework to build or convert a high performance application to run in the cloud. We show that following these directives leads to an elastic implementation that has better scalability, run-time resource adaptability, fault tolerance, and portability across cloud computing platforms, while requiring minimal effort and intervention from the user. We illustrate this by converting an MPI implementation of replica exchange, a parallel tempering molecular dynamics application, to an elastic cloud application using the Work Queue framework that adheres to these directive. We observe better scalability and resource adaptability of this elastic application on multiple platforms, including a homogeneous cluster environment (SGE) and heterogeneous cloud computing environments such as Microsoft Azure and Amazon EC2.**

I. Introduction

Scientists and developers have built several scientific and high performance applications to study, simulate, investigate, and evaluate scientific phenomena. With the advent of super- and parallel computing, these high performance applications have evolved to take advantage of dedicated hardware to achieve scalability within the constraints of available resources at startup time, and lower time to completion. They employ parallel programming techniques such as Message Passing Interface (MPI) to achieve these improvements over traditional sequential implementations [1].

At the same time, new computing trends and paradigms have continued to emerge. Recently, cloud computing has emerged as a very popular computing paradigm offering on-demand resource allocation and usage, lower costs, distributed storage and backup, and a usage based pricing model [2], [3]. With the advent of the distributed execution and run-time environments offered in cloud computing, we argue that the programming frameworks for scientific and high performance applications need to shift away from parallel execution models to distributed computing models to fully leverage the benefits offered in cloud environments.

We first study the issues and disadvantages of using parallel computing frameworks. We found that these frameworks have poor fault tolerance and they leave the responsibilities of providing failure recovery to the application developer. As a result, scalability is limited and constrained by the availability of dedicated and carefully controlled resources. In addition, we find that parallel execution environments are inefficient and ineffective in utilizing all available resources and any resources that become available during run-time. Based on these disadvantages, we determine that parallel computing environments cannot serve as a platform for building and running *elastic* applications that can dynamically adapt to resource availability and harness available resources, scale, progress through failures and errors, and be portable across platforms. In this paper, we explore and show cloud computing as a viable platform for building and running elastic applications.

We present guidelines for the design and development of cloud computing frameworks [4], [5], [6] that abstract underlying execution environments and distributed resources, and provide an interface for building and deploying applications in the cloud. We formulate these guidelines from the insights gained in studying high performance applications implemented using parallel computing interfaces and techniques.

We also present a case study in converting a high performance application, a replica exchange molecular dynamics application [7], [8] implemented using MPI, to an elastic cloud application using a distributed computing framework that adheres to the presented guidelines. We compare the MPI-based high performance application with its corresponding elastic cloud application and show the challenges in running high performance computations, such as scalability, fault-tolerance, failure recovery, are elegantly addressed with the elastic application. We then deploy and run our elastic application on different cloud computing platforms illustrating its portability across multiple platforms. We also use our experimental runs on the different platforms to demonstrate the benefits of adherence to the guidelines.

The contributions of this paper are summarized as follows: It proposes guidelines for the design, implementation, or identification of a cloud computing framework used to convert and run existing high performance applications or builds new elastic applications in the cloud. The paper then proceeds to describe a case study in the conversion of an existing high performance application to an elastic implementation through a framework that adheres to these guidelines.

The rest of the paper is organized in the following way: We study the problems with parallel computing techniques, such as MPI, as a framework in building high performance applications and use the insights gained to formulate directives for the design and use of a cloud computing framework in Section II. In Section III, we describe the architecture of a cloud computing framework that adheres to these directives. In Section IV, we describe replica exchange molecular dynamics, a high performance computation used in the study of protein folding and typically implemented using MPI. We also study its underlying computational logic and explain its conversion from an MPI-based implementation to an implementation in the cloud. In Section V, we present the evaluations and results of running this cloud implementation of the high performance computation. We describe related work in Section VI. We conclude in Section VII and discuss directions and avenues for future work.

II. Motivation

Applications and software built with parallel programming interfaces and techniques offer certain advantages, such as better utilization of dedicated resources and lower time-to-completion, over traditional sequential models. Most of these applications are highly computation intensive and demand excessive memory and resource availability due to the complexities involved in studying scientific phenomena. Such applications typically employ parallel programming interfaces, such as MPI, to achieve parallelism in their execution and take advantage of dedicated hardware to speed-up execution and time-to-completion. Nevertheless, the excessive and high demands for resources present significant obstacles to the successful execution and completion of these programs. To overcome the limitations of excessive resource requirements, the simulations are typically run with expensive high performance and supercomputing hardware components. While this approach can satisfy resource requirements to a certain degree, it offers poor scalability since resources are limited by hardware costs and availability.

Further, such parallel computing based applications lack the ability to harness any currently available resources to proceed with execution and this often leads to poor productivity where valuable time is spent waiting for all requested resources to be available. Also, while fault tolerance can be achieved with checkpointing or other error recovery techniques implemented at the application level, they inherently lack a dynamic fault-tolerance and error-recovery mechanism that will allow for executions to recover from multiple failures, proceed execution or migrate seamlessly to another site in the event of unrecoverable failures. The behavior and performance of such applications vary with hardware, platform, network characteristics and quirks [9], [10] and often need to be tuned to suit the platform and hardware used in execution. These factors further limit scalability and lead to poor portability across platforms and, high development and deployment costs.

To overcome these shortcomings of parallel computing and take advantage of the distributed programming and execution models offered by cloud computing, these application need to be converted to an elastic application in the cloud. However, designing and rewriting these applications to run in the cloud computing environments can consume extensive effort, time, and cost, especially if these applications contain large amounts of code. To lower the level of effort and cost required, developers can take advantage of cloud computing frameworks and engineer their applications to run in cloud environments through their API and library interfaces. To help build, design, identify, or use an existing cloud computing framework for creating or modifying high performance applications for the cloud, we establish a set of directives to follow, which are presented below:

Directive 1: *Scalability.* The cloud computing framework must allow applications to scale in size, complexity, and resource usage without being constrained by the need for dedicated hardware resources.

Directive 2: *Resource Adaptability.* The framework must dynamically harness resources as they become available and allow applications to utilize these resources to progress in their execution. It must also adapt to loss or failure of allocated resources and still allow the application to continue execution. This must be done seamlessly and transparently to the application.

Directive 3: *Fault tolerance.* The framework must provide robust fault-tolerance and error recovery at different levels to the application. It must allow the application to continue execution even in the presence of hardware failures, communication failures, site failures, and execution errors and failures. The framework must dynamically rerun failed tasks or seamlessly migrate them to other sites in the event of such failures.

Directive 4: *Portability.* The framework must be able to deploy and run the application on different cloud computing platforms with minimal effort and intervention from the user.

Directive 5: *Platform independence.* The framework must be independent of platforms, operating environments, and hardware and must be able to deploy and run applications with minimal effort and intervention from the user. It must also hide any platform, operating system, and hardware characteristics from the application.

Directive 6: *Application independence.* The framework must not be tied to any particular application architecture, type, or implementation. In essence, the framework should be able to execute any application as long as the required dependencies and executing environments are available or provided.

Directive 7: *Ease of effort.* The framework must allow users to migrate their applications with minimal effort to the cloud. It must offer easy to use API, libraries, interfaces and facilitate quicker development and deployment of their applications to the cloud.

With the establishment of the directives for the design and/or use of a cloud computing framework, we proceed to describe one such framework, Work Queue [4], developed at the University of Notre Dame.

III. Cloud Computing Framework

In this work, we utilize and employ a cloud computing framework called Work Queue [4] to deploy and run elastic implementations of high performance applications. The Work Queue framework is based on the master-worker paradigm, where multiple worker processes can receive and execute workloads sent by the master. The master coordinates the execution of a given application by assigning and scheduling work units to each of the workers. Figure 1 illustrates the master-worker architecture of Work Queue. The arrows describe the communications between the master and worker. The communications occur at the following times: (a) transfer of input including application executables, binaries, input files etc., from master to workers, (b) communication of the task execution commands and their arguments by master to its workers, and (c) transfer of output including output files and logs from workers to master.

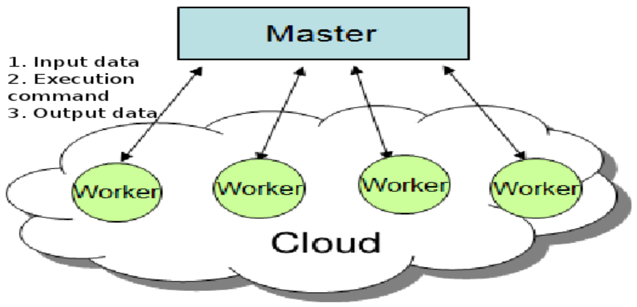

Fig. 1. Architecture of Work Queue.

The master in the framework can be considered as a wrapper script around the application. This master script is implemented by the user using the Work Queue API. The input files to be sent to the worker site, the executables to be run, the execution commands and arguments, the output files that need to be retrieved from the workers are specified in the master script using the appropriate API calls. The individual units of execution of the application, referred from here on as tasks, are also specified in the master and are dispatched to the workers for execution. The master coordinates the executions of the tasks and aggregates the results and specified output files from their execution.

The workers are deployed as executables on the cloud platform and they are invoked and run as jobs on these platforms through their respective job submission interfaces. The workers can be compiled, installed, and run on any POSIX compliant environment. This implies that the worker can virtually be deployed and run on any operating environment including Microsoft Windows based environments (using cygwin).

The master script implemented by the user is often relatively simple. This is because the master script only contains the input file specifications, the executables required for task execution, the output file specifications, and the task execution command and arguments. The assignment and scheduling of tasks to workers, transfer of input and output files, fault tolerance and recovery in the event of any errors and failures at workers are handled without application intervention. These properties illustrate Work Queue's adherence to Directive 7.

We proceed to describe a case study in converting an high performance application into an elastic cloud application. The high performance application considered is replica exchange molecular dynamics, which accelerates the simulation of protein motion due to atomic interactions and is useful in studying protein folding.

IV. Case Study: Elastic Replica Exchange

Proteins are molecular machines that need to move in order to function. Nascent proteins must fold into a distinctive shape that enables them to interact with other molecules and carry several functions in the cell. Protein folding is a grand challenge and has been simulated using molecular dynamics (MD), which numerically integrates Newton's equations of motion for all the atoms in a protein. The potential energy of the system dictates the probability of remaining in a given geometric configuration, while the temperature provides energy to jump over barriers in the potential energy surface. Due to the high dimensionality of the problem, MD simulations often get trapped in local minima of the potential energy surface. One way of overcoming energy barriers is to raise the temperature of the system; however, the paths obtained from high temperature simulations do not correspond to the paths at the lower temperature. A technique used to improve the sampling of the potential energy surface is parallel tempering, also called replica exchange molecular dynamics [7], [8], which has replicas at many temperatures. Many configurations are visited by the high temperature replicas and then annealed to lower temperature replicas by a Monte Carlo procedure that achieves the correct statistical distribution. Particularly, simulations are run by creating multiple replicas of a protein molecule and executing each over several Monte Carlo steps or iterations at different temperatures. At the end of every iteration, an exchange is attempted between neighboring replicas, where if certain criteria are met, the replicas are swapped with regards to their temperature and the simulation is continued. The simulations of replicas in each iteration are completely independent and can be performed parallel to each other. The communication between replicas only happens at the end of each iteration when an exchange between replicas is attempted.

There are several simulation software packages, such as ProtoMol, Gromacs, NAMD, that simulate the dynamics of protein molecules and are used to study and perform replica exchanges [11], [12], [13]. In this work, we employ ProtoMol

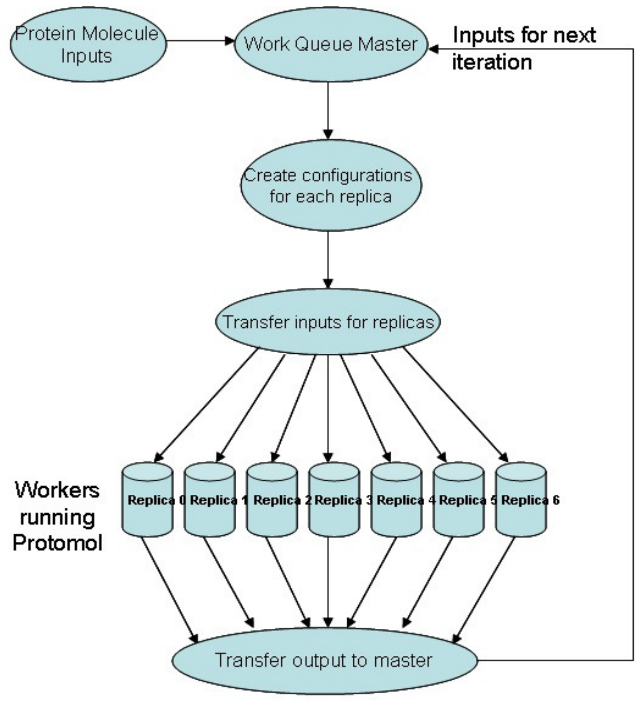

Fig. 2. Elastic implementation of Replica Exchange using Work Queue.

as the underlying software that performs simulations of the protein molecules represented in each replica.

The replica exchange computations and simulations are typically implemented using MPI. Some of the simulation software such as ProtoMol and Gromacs offer built-in MPI implementations for this purpose. However, as we discussed earlier in Section II, such an MPI based implementation exhibits several disadvantages and inefficiencies. To overcome these shortcomings of an MPI based implementation of replica exchange, we built an elastic implementation using the Work Queue framework described above.

The implementation of elastic replica exchange required the creation of the master script as detailed in Section III. We built the master, which here is a wrapper script around ProtoMol, using the Work Queue API. This master script creates and specifies the configuration and input files required for each iteration of the simulation in ProtoMol and gathers the output files upon their completion by the workers. At the end of each iteration, the master checks to see if an exchange can be attempted between two replicas and if so swaps the necessary parameters of those replicas. The master then proceeds to generate the configuration and input files for the next iteration. The master, therefore, coordinates the entire simulation run across workers distributed and running inside cloud infrastructures. Figure 2 illustrates the work flow in the elastic implementation of replica exchange using the Work Queue framework. Elastic replica exchange is available as part of the cctools package that can be downloaded at *http://cse.nd.edu/∼ccl/software/download.shtml*.

In the next section, we experimentally study and evaluate this elastic implementation of replica exchange.

V. Experimental study

In this section, we first study the performance of elastic replica exchange in comparison to the MPI-based implementation. We then proceed to describe and compare our experiences in running elastic replica exchange on different cloud platforms including Amazon EC2 and Microsoft Azure.

A. *Elastic Replica Exchange*

We compare the performance of the elastic implementation of replica exchange using Work Queue against its MPI implementation. For the experiments in this section, we deployed and ran both implementations on the Sun Grid Engine [14] infrastructure at the University of Notre Dame. Each experimental run involved simulations over 100 Monte Carlo steps with each step running 10000 molecular dynamics steps. Figure 3 compares the running time of the MPI- and Work Queue-based implementations of replica exchange running simulations over several replicas. The number of workers deployed and run was equal to the number of replicas simulated in the experiment. For example, a run with 30 replicas had 30 workers being deployed and run. The running time of these experiments were measured from the start of simulation to its completion. Therefore, Figure 3 does not include the job queuing and scheduling delays. In this figure, we observe that the Work Queue implementation has a slightly higher running time than the MPI implementation. This is attributed to two main factors: (a) communication and data transfer overheads between the master and workers that running remotely, and (b) recovery from failures on the worker sites.

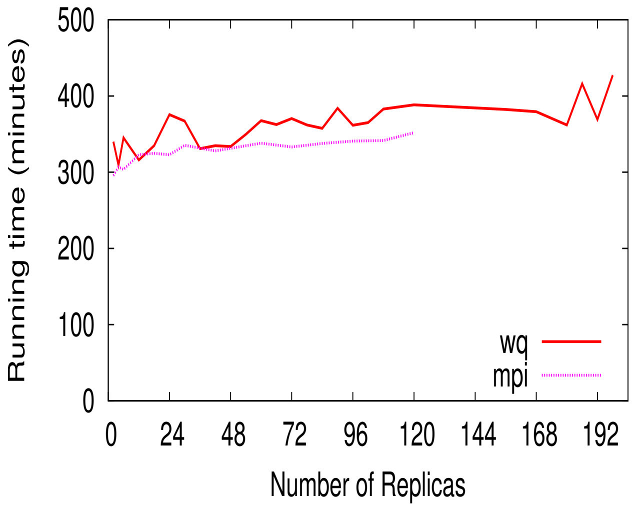

Fig. 3. Running time performance comparison of MPI- and Work Queue-based implementations of replica exchange.

An important observation we make in this figure is that the MPI runs do not scale well beyond 120 replicas. At the time of

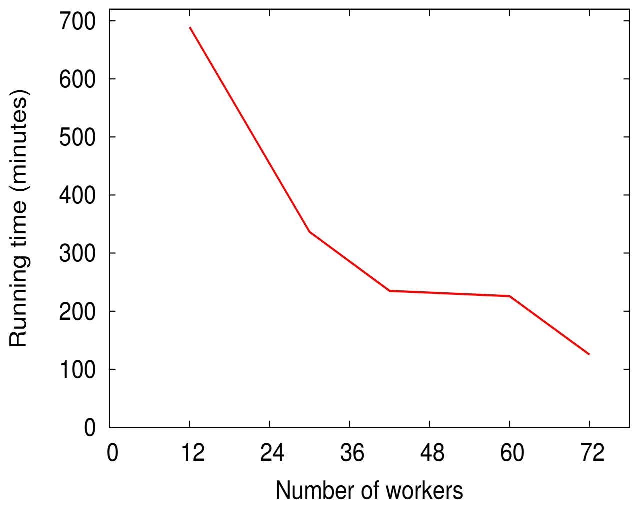

Fig. 4. Evaluation of running time performance with varying number of workers for an experimental run with 72 replicas simulated over 100 Monte Carlo steps.

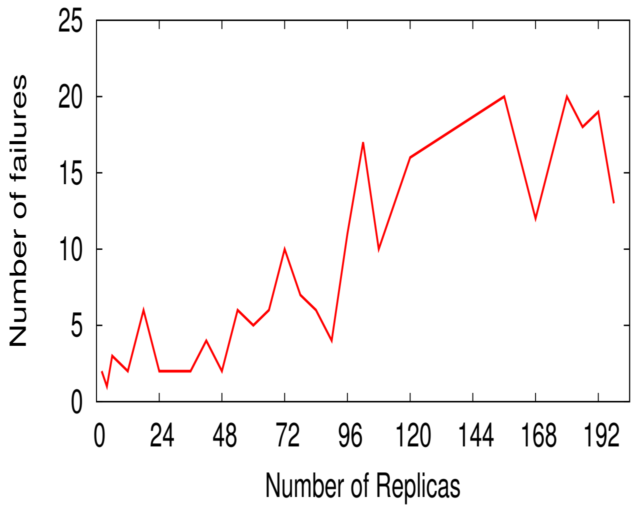

Fig. 5. Number of failures observed with Work Queue implementation of replica exchange.

this writing, they were still queued and waiting to be scheduled due to the lack of resources at this scale being available simultaneously on the Notre Dame SGE infrastructure. We also note that the Notre Dame SGE has a constraint on the resources available to a user at any given time, thereby limiting the scalability of the MPI implementation even with the availability of more resources. On the other hand, we see that the Work Queue implementation scales well due to its ability to scavenge and utilize resources as they become available. Experiments beyond 120 replicas were achieved by deploying multiple workers on allocated resources with multiple computing cores and invoking their execution on each core through MPI. This illustrates better scalability of the Work Queue implementation over the MPI-based implementation as required by Directive 1.

In Figure 4, we plot the running times of simulations run with 72 replicas over 100 Monte Carlo steps and 10000 molecular dynamics steps while the number of workers deployed is varied. We observe the running time to increase with a lower number of workers deployed. This is because each Monte Carlo step of a replica in the simulation has to be executed by a smaller set of workers and the parallelism in executing the steps of several replicas simultaneously is lowered. Nevertheless, we note the simulations successfully run to their completion even with a smaller pool of workers. This observation satisfies Directive 2 which requires the implementation to adapt to varying resource availability, harness available resources, and run to completion even with a limited set of available resources.

Figure 5 illustrates the number of failures that occurred on the worker sites during each of the experimental runs plotted in Figure 3. These failures are attributable to a variety of factors such as network errors, hardware failures, data corruption, etc. With the MPI-based implementation, these failures will stall the entire experiment and will need to be restarted and rerun. The Work Queue implementation offers fault-tolerance by rerunning only the failed task in the experimental run, and in the event of any unrecoverable failures at a worker, migrating its execution to a different worker. Thus the Work Queue implementation satisfies Directive 3.

B. *Deployment on different cloud platforms*

We proceed to port and study the elastic implementation of replica exchange on two other cloud computing platforms, Amazon EC2 and Microsoft Azure. We describe the behavior of our elastic implementation on these different platforms. Our objective is *not* to compare the performance of the cloud systems against each other, as they all offer different cost-performance trade-offs using different hardware. Instead, our goal here is to show that our system functions correctly and portably across multiple different environments.

Campus SGE. The Sun Grid Engine (SGE) at the University of Notre Dame is maintained as a dedicated platform for running high performance scientific applications. The jobs are submitted to the compute nodes via the SGE batch submission system [14]. The compute nodes run Red Hat Enterprise Linux (RHEL) as their operating environment. The compute nodes here are typically composed of high-end hardware. The workers were queued and submitted as jobs to this grid. Upon being scheduled and run, the workers connect to the master and execute the assigned workloads.

Amazon EC2. The Elastic Compute Cloud or EC2, built by Amazon.com, is a platform that allows virtual machine instances to be requested, allocated, and deployed on demand by users. Different instance sizes are provided with varying hardware configurations to satisfy different requirements and workloads of the users [15], [16]. The instances allocated can

TABLE I
DESCRIPTION OF THE PLATFORMS USED.

Name	System	Processor	I/O	Cost
Platform A	Amazon EC2	2*2 x 1.0-1.2 GHz	7.5 GB memory	\$0.34/hour
Platform B	Notre Dame SGE	2*2 x 2.6 GHz	8-12 GB memory	\$0.0/hour
Platform C	Microsoft Azure	2 x 1.6 GHz	3.5 GB memory	\$0.24/hour

be installed and run with different Linux operating system flavors and kernels and their operating environments can also be customized. Since the instances can be installed and customized to run a Linux environment, the migration of our implementation to EC2 was similar to SGE.

Microsoft Azure. The Windows Azure platform, from Microsoft, offers virtual instances running an image of the Azure operating system. The virtualized instance is offered through the Azure hypervisor and provides an operating environment based off the Windows Server 2008 R2 VM system [17], [18]. There are two computational roles offered in the platform - the web role that serves as the front end interface to the allocated compute instances, and the worker role that serves as the core computing unit that runs tasks and applications. As a result of this two tiered architecture, we built wrapper scripts that communicate to the web role and invoke workers on the worker roles. We also used cygwin-compiled executables in migrating our implementation to this environment.

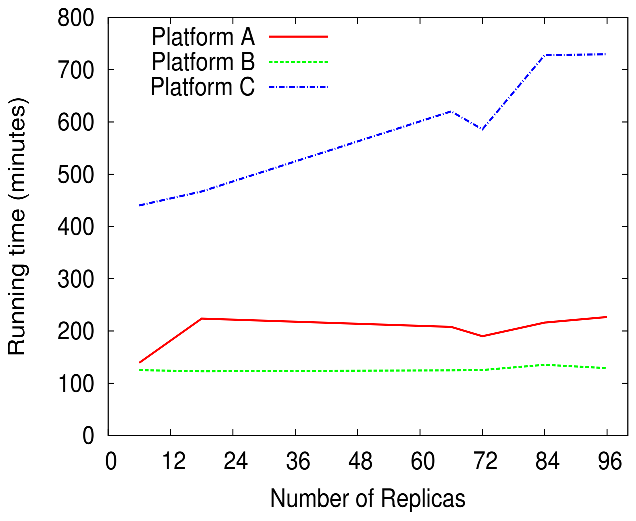

Fig. 6. Comparison of running times on the three platforms described in Table 1.

The above experiences in porting and deploying our implementation on different cloud computing platforms proves its adherence to Directive 4.

We now study the behavior of our implementation on the different cloud platforms. In the experiments described below, the number of workers deployed were again equivalent to the number of replicas involved in the simulation run. Each experiment ran simulations performed over 100 Monte Carlo steps involving 10000 molecular dynamics steps each. Figure

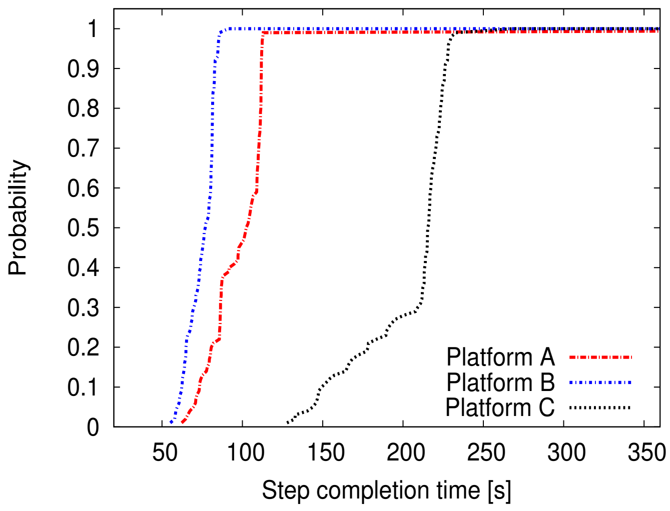

Fig. 7. Cumulative distribution function of the step completion times in the experimental run involving 18 replicas on the three platforms.

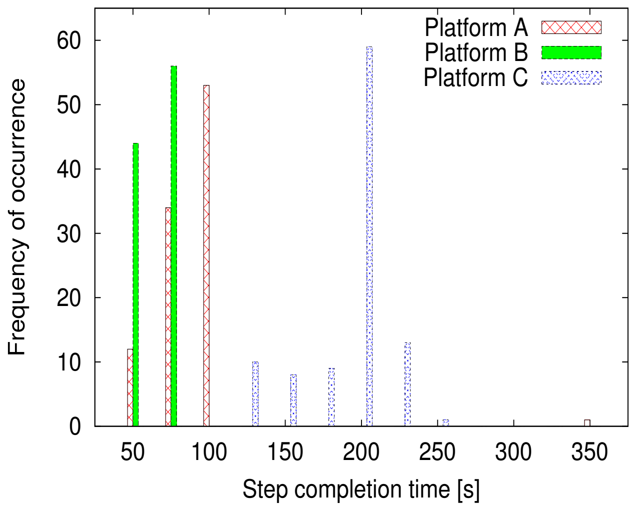

Fig. 8. Histogram plot of the step completion times in the experimental run involving 18 replicas on the three platforms. The bins used in the plot were of size 25 time units (seconds) and each bin consists of all values that are greater than or equal to the corresponding bin label. The bin labels are plotted on the x-axis.

6 shows the running times of experimental runs with varying replica sizes on these platforms. The x-axis represents the number of replicas involved in each run. Figures 7 and 8 plot statistical data on the completion time of the individual Monte Carlo steps on the three platforms from experimental

runs involving 18 replicas. Figure 7 plots the cumulative distribution function of the completion time of each step. Figure 8 gives the corresponding histogram plotting the distribution of completion times after being classified in bins.

From Figures 7 and 8, we notice significant variations in the completion times of the Monte Carlo steps on one of the platforms (Platform C) as compared to the other two platforms. While these variations can be attributed to one or more of several factors including network latencies and jitter, virtualization effects, firewall, load balancing etc., this is good evidence that our implementation is impervious to any peculiar platform and network characteristics of a cloud computing system. Our implementation demonstrates the ability to hide differences in design, implementation, and behavior of different cloud computing platforms from the application and therefore satisfies the requirements in Directive 5.

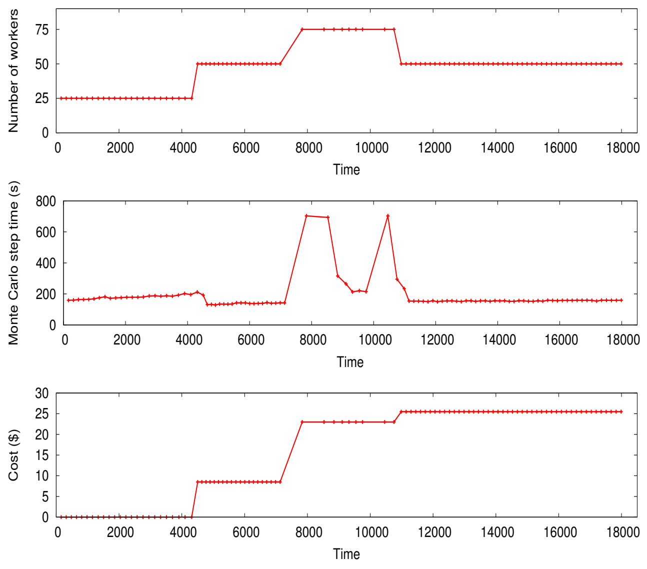

Fig. 9. Illustration of a run across all three platforms described in Table 1.

We also demonstrate the execution of elastic replica exchange across multiple cloud platforms. We show this by deploying workers across all three platforms in Table 1. The experimental run involved 75 replicas simulated over 100 Monte Carlo steps running 10000 molecular dynamics steps. Figure 9 presents the number of workers connected, the completion time of each Monte Carlo step, and the running cost during this experimental run. The experiment was started with 25 workers being submitted and run on the Platform B. Since there are 75 tasks, as a task corresponds to a simulation step of one replica, it takes three round-trips of task execution for these workers to finish a Monte Carlo step.

After an hour (around 4200 seconds in Figure 9), we deploy and run 25 workers on instances in Platform A bringing the total number of workers to 50. This addition of resources lowers the completion time of Monte Carlo steps as it requires only two round-trips of task executions across the workers. We then deploy and run another 25 workers on Platform C after two hours of run-time (around 7600 seconds). We immediately observe a spike in the running time which we attribute to the long transfer times in sending the executable of the simulation program, Protomol, to the added workers. We also observe from Figure 9 that the addition of these workers on Platform C results in an increase in the running time of each step. This is because the running time of simulation steps on Platform C significantly varies from the other platforms as we observe in Figure 6. We attribute these to differences in the hardware, network, and system characteristics and specifications of these platforms. As a result, adding workers on Platform C negates any benefits gained from the parallelism in running 75 tasks simultaneously. We manually removed these workers on Platform C after an hour of their deployment (around 10100 seconds) to speed up completion of the experimental run. This results in the failure of tasks running on the removed workers. The spike in the running time following the removal of these workers is attributed to the failed tasks being scheduled and rerun on the remaining workers.

Our future work will explore techniques that will measure the benefits of adding more workers to an ongoing application execution and automatically remove workers that have any negative impact on the execution without requiring manual intervention. This experimental run further illustrates adherence of our framework to Directives 2 (resource adaptability), 4 (portability), and 5 (platform independence).

VI. Related Work

There have been several efforts in studying the deployment of scientific and high performance applications on various cloud computing platforms [19], [20], [21]. Our work differs by showing the deployment of a high performance application on multiple cloud computing platforms through a cloud computing framework. We also establish guidelines for the design, implementation, and identification of cloud computing frameworks in this work.

There have also been several efforts in building and migrating bio-molecular applications to distributed computing environments [22], [23], [24]. The work in [22] present a framework that provides fault-tolerance and failure recovery for running replica-exchange simulations on distributed systems. This is achieved through checkpointing and an external interface that monitors the execution of distributed applications. Work Queue differs by offering these functionalities inherently without overheads. The authors in [23] describe their experiences in running a replica-exchange simulation software, NAMD, on the Condor grid. They add a dedicated set of resources to speedup slow replicas executing on the grid and notice improvement in the efficient usage of available resources. The authors go on to present a database architecture for storing and retrieving bio-molecular simulation results.

The work in [24] describes experiences in using Legion, an operating system that provides abstractions for managing and utilizing grid resources, to run replica-exchange simulations built using MPI. This work provides good insights on the effectiveness of abstractions in providing a seamless transition for users porting applications to run on grids.

The system in [25] provides shorter time-to-result of a simulation of large protein systems. It uses a combination of distributed and parallel computing techniques to achieve this.

Our work differs from related efforts by establishing guidelines and deploying an high performance application on multiple cloud platforms. It illustrates the benefits of the guidelines in providing enhanced scalability, fault tolerance, portability to different platforms, and better resource harnessing and usage.

VII. Conclusion and Future Work

We studied the disadvantages of parallel computing as a platform to build and run build high performance and scientific applications. We find that parallel computing is not conducive to building elastic applications that can adapt to resource availability, recover from failures, scale, and are portable across multiple platforms. On the other hand, distributed computing provides mechanisms that can overcome the disadvantages with parallel computing. We show cloud computing, which is built on a distributed computing model, as a viable platform for building elastic applications of high performance computations. To build or migrate applications to run in the cloud, cloud computing frameworks and abstractions are employed to lower cost and effort. In this paper, we established directives to help in the design, implementation, and identification of cloud computing frameworks for building and running high performance applications. These directives were formulated to overcome the disadvantages with parallel computing and leverage the benefits of cloud computing. By converting an existing replica exchange molecular dynamics application to an elastic application using a framework that satisfies these directives, we show the challenges of high performance computing such as scalability, resource adaptability during runtime, fault-tolerance, and failure recovery are more elegantly addressed using cloud computing.

Our future work will consider scalability in terms of the application complexity. We also plan on migrating high performance applications from different scientific fields to the cloud using our framework and thereby illustrating adherence to Directive 6. Another direction we aim to explore is identifying how our framework can be leveraged to optimize certain high performance applications and allow computations that are not possible in parallel computing environments to be performed.

VIII. Acknowledgements

We thank Jonathan Schuster for his contributions in porting our implementation to the Microsoft Azure platform. We also thank Connor Kennan for his help in running the MPI-based implementation of replica exchange.

This work was supported by the National Science Foundation under grants NSF-CNS-0643229, NSF-CNS-0855047, and partially under grant NSF-CCF-1018570.

References

[1] P. S. Pacheco, *Parallel programming with MPI*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1996.

[2] L. Youseff, M. Butrico, and D. Da Silva, "Toward a unified ontology of cloud computing," in *Grid Computing Environments Workshop, 2008. GCE '08*, 2008, pp. 1 –10.

[3] I. Foster, Y. Zhao, I. Raicu, and S. Lu, "Cloud computing and grid computing 360-degree compared," in *Grid Computing Environments Workshop, 2008. GCE '08*, 2008, pp. 1 –10.

[4] L. Yu and et al., "Harnessing parallelism in multicore clusters with the all-pairs, wavefront, and makeflow abstractions," *Journal of Cluster Computing*, vol. 13, no. 3, pp. 243–256, 2010.

[5] "The directed acyclic graph manager," http://www.cs.wisc.edu/condor/dagman, 2002.

[6] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," in *Operating Systems Design and Implementation*, 2004.

[7] Y. Sugita and Y. Okamoto, "Replica-exchange molecular dynamics method for protein folding," *Chemical Physics Letters*, vol. 314, no. 1-2, pp. 141 – 151, 1999.

[8] P. Brenner, C. R. Sweet, D. VonHandorf, and J. A. Izaguirre, "Accelerating the replica exchange method through an efficient all-pairs exchange," *Journal of Chemical Physics*, vol. 126, p. 074103, February 2007.

[9] K. Al-Tawil and C. A. Moritz, "Performance modeling and evaluation of mpi," *Journal of Parallel and Distributed Computing*, vol. 61, no. 2, pp. 202 – 223, 2001.

[10] M. Resch, H. Berger, and T. Bönisch, "A comparison of mpi performance on different mpps," in *Proceedings of the 4th European PVM/MPI Users' Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing Interface*. Springer-Verlag, 1997, pp. 25–32.

[11] T. Matthey and et al., "Protomol, an object-oriented framework for prototyping novel algorithms for molecular dynamics," *ACM Transactions on Mathematical Software*, vol. 30, pp. 237–265, September 2004.

[12] J. Phillips, G. Zheng, S. Kumar, and L. Kale, "Namd: Biomolecular simulation on thousands of processors," in *Supercomputing, ACM/IEEE 2002 Conference*, November 2002, p. 36.

[13] E. Lindahl, B. Hess, and D. van der Spoel, "Gromacs 3.0: a package for molecular simulation and trajectory analysis," *Journal of Molecular Modeling*, vol. 7, pp. 306–317, 2001.

[14] W. Gentzsch, "Sun grid engine: Towards creating a compute power grid," in *Proceedings of the 1st International Symposium on Cluster Computing and the Grid*, ser. CCGRID '01, 2001, pp. 35–.

[15] Amazon.com, "Elastic compute cloud (ec2)," http://www.aws.amazon.com/ec2.

[16] S. Ostermann, A. Iosup, N. Yigitbasi, R. Prodan, T. Fahringer, and D. Epema, "A performance analysis of ec2 cloud computing services for scientific computing," in *Cloud Computing*, ser. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, vol. 34, 2010, pp. 115–131.

[17] Microsoft Corporation, "Microsoft windows azure platform," http://www.microsoft.com/windowsazure.

[18] Z. Hill, J. Li, M. Mao, A. Ruiz-Alvarez, and M. Humphrey, "Early observations on the performance of windows azure," in *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, 2010, pp. 367–376.

[19] K. Keahey, R. Figueiredo, J. Fortes, T. Freeman, and M. Tsugawa, "Science clouds: Early experiences in cloud computing for scientific applications," 2008.

[20] G. Juve, E. Deelman, K. Vahi, G. Mehta, B. Berriman, B. Berman, and P. Maechling, "Scientific workflow applications on amazon ec2," in *E-Science Workshops, 2009 5th IEEE International Conference on*, December 2009, pp. 59–66.

[21] 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*, 2008, pp. 640 –645.

[22] A. Luckow and et al., "Distributed replica-exchange simulations on production environments using saga and migol," in *IEEE Fourth International Conference on eScience, 2008*, December 2008, pp. 253–260.

[23] C. J. Woods and et al., "Grid computing and biomolecular simulation," *Philosophical Transactions: Mathematical, Physical and Engineering Sciences*, vol. 363, pp. 2017–2035, 2009.

[24] A. Natrajan, M. Crowley, N. Wilkins-Diehr, M. Humphrey, A. Fox, A. Grimshaw, and I. Brooks, C.L., "Studying protein folding on the grid: Experiences using charmm on npaci resources under legion," in *Proceedings of the 10th IEEE International Symposium on High Performance Distributed Computing*, 2001, pp. 14–21.

[25] S. Pronk and et al., "Copernicus: A new paradigm for parallel adaptive molecular dynamics," in *Proceedings of the 2011 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, November 2011.