Abstractions for Loosely-Coupled and Ensemble-based Simulations on Azure

André Luckow[1], Shantenu Jha[1,2,3,*]

[1]*Center for Computation & Technology, Louisiana State University, USA*

[2]*Department of Computer Science, Louisiana State University, USA*

[3]*e-Science Institute, Edinburgh, UK*

[*]*Contact Author:* sjha@cct.lsu.edu

Abstract

Azure is an emerging cloud platform developed and operated by Microsoft. It provides a range of abstractions and building blocks for creating scalable and reliable scientific applications. In this paper we investigate the applicability of the Azure abstractions to the well-known class of loosely coupled and ensemble-based applications. We propose the BigJob API as a novel abstraction for managing groups of Azure worker roles and for remotely executing tasks on them. We demonstrate that Azure enhanced with BigJob functionality provides performance comparable to other grid and cloud offerings loosely-coupled applications.

I. Introduction

Emerging cloud platforms such as Microsoft's Azure [1], present relatively simple computing environments compared to traditional grid computing environments, in terms of resource management, capacity planning capabilities, software environment & control etc. In spite of the promise of clouds as a viable platform for Science & Engineering applications, many fundamental questions persist about how scientific applications will utilize clouds as presented, both now and into the future? Some existing legacy applications no doubt will adapt and take advantage of new capabilities, but it is unclear how clouds as currently presented are likely to change (fundamentally?) the development and deployment of scientific applications, and the process of scientific investigation. The challenges surrounding the effective uptake of clouds are both broad and deep, in that, there are both application-level questions as well as system-level questions. For example, are clouds viable alternatives to production grid infrastructure or will they be part of a larger production cyberinfrastructure? What kind of scientific services can clouds support and how should clouds be provisioned and provided to the scientific community?

To understand some of these questions we investigate Ensemble-based (enMD) approaches, which are commonly used in simulations to facilitate drug discovery and to provide fundamental insight into molecular structure, dynamics and interactions. Different kinds of ensemble-based approaches are commonly deployed: Ensembles of independent simulations aim to improve the statistical sampling of MD simulations. In some scenarios, sampling of physical-states can be improved by infrequent attempts to exchange partial state information between pairs of replicas; the replica-exchange (RE) algorithm [2] is a well-known example of this class.

In Luckow et al. [3] efficient scale-out of RE simulations was established for multiple TeraGrid (TG) resources. We used the BigJob framework to decouple workload submission from resource assignment; this results in a flexible execution model, which in turn enables the distributed scale-out of applications on multiple and possibly heterogeneous resources *concurrently*. In Ref. [4] we extended this work to support production grid infrastructures based on Condor as well as EC2-style clouds (e. g. FutureGrid).

The primary aim of this paper is to explore the abstractions that the Windows Azure platform provides, and their suitability to for orchestrating distributed, loosely-coupled workloads, as required by enMD simulations. Furthermore, we determine how existing and established abstractions for distributed computing can be extended to Azure. We analyze the suitability of the different Azure abstractions, such as the Azure Queue-service (AQS), Azure Blob Storage (ABS) and the Worker Role API.We propose the existing BigJob API as an viable and efficient abstraction for managing a group of Azure worker roles and for remotely executing tasks on them.

This paper is structured as follows: In § II we present an overview of the Azure platform. In § III the pilot-job concept and the BigJob framework is introduced. The Azure BigJob implementation is presented in § IV. In § V we discuss the architecture of the Azure-based enMD and RE application. A significant issue in cloud computing is performance. We provide an in-depth performance analysis in § VI.

II. Windows Azure

Windows Azure offers a platform for on-demand computing and for hosting generic server-side applications. In contrast to infrastructure as a service clouds (IaaS),

such as Amazon EC2/S3, Azure follows the platform as a service paradigm (PaaS). The platform encapsulates common application patterns into so-called *roles*, removing the need to manually manage low-level details on virtual machine (VM) level. The so-called Azure fabric controller automatically monitors all VMs, reacts to hardware and software failures and manages application upgrades.

1) Compute: Windows Azure currently provides two kind of roles: Web roles e. g. are used to host web applications and front-end code, while worker roles are well suited for background processing. While these roles target specific scenarios and are not as flexible as bare EC2 images, they are also customizable, e.g., Worker roles can run native code. The application must implement the worker role API, which provides a `run` method as a defined entry point for executing the actual compute task. Further, several additional API callbacks, e. g., for state or configuration change notifications, are provided.The Azure fabric controller automatically manages and monitors applications, handles hardware and software failures as well as updates to the operating system or the application. Scientific applications commonly utilize worker roles for compute and/or data-intensive tasks. AzureBlast [5] e. g. heavily relies on worker roles for computing bio-sequences.

2) Storage: Azure provides three key services or storing large amounts of data: the *Azure Blob Storage* (ABS) for storing large objects of raw data; the *Azure Table Storage* (ATS) for semi-structured data, and the *Azure Queue Storage* (AQS) for implementing message queues. The stored data is replicated across multiple data centers to protect it against hardware and software failures. In contrast to other services (e. g. Amazon S3) [6], the Azure Storage Services provide strong consistency guarantees, i. e., all changes are immediately visible to all future calls. While eventual consistency offers a better performance and scalability, it has some disadvantages mainly caused by the fact that the complexity is moved to the application space.

ABS can store file up to a size of 1 TB, which makes it particularly well suited for data-intensive applications. S3 e. g. restricts the maximum file size to 5 GB. Further, the access to the ABS blob storage can be optimized for certain usage modes: *block blob* can be split into chunks which can be uploaded and downloaded separately and in parallel. Thus, block blobs are well suited for large amounts of data. *Page blob* manage the storage as an array of pages. Each of these pages can be addressed individually, which makes page blobs a good tool for random read/write scenarios.

AQS provides reliable storage for the delivery of messages for distributed applications. It is ideal to orchestrate the various components of a distributed application, e. g. by distributing work packages or collecting results.

III. Pilot-Jobs and SAGA BigJob

Pilot-Jobs are a useful abstraction for efficiently executing an ensemble of batch jobs on distributed infrastructures without the necessity to queue each individual job. The pilot-job itself is a regular grid job, which is started through a grid resource manager, such as the Globus GRAM. Once the batch queue assigns the requested resources to the pilot-job, the pilot-job circumvents the necessity to queue each individual sub-job, and is responsible for managing the resources. Thus queuing times for sub-jobs can be reduced and the predictability for application execution can be increased. In effect, pilot-jobs decouple resource allocation from resource binding and allow the efficient utilization of resources. By delaying the resource binding, and enabling scheduling decision at the application-level, dynamic execution and usage modes can be supported. For example, the load-dependent sizing of sub-jobs, or the dynamic addition of resources to meet deadlines [4].

As more applications take advantage of dynamic execution, the pilot-job concept has grown in popularity and has been extensively researched and implemented for different usage scenarios and infrastructure including clouds. Nimbus [7] e. g. provides a pilot-job like abstractions for clouds. For this purpose, Nimbus allows the launch of auto-configured virtual machine clusters that contain a Torque and Globus installation. The Atlas computing framework developed at CERN also heavily relies on the PanDA pilot-job framework [8] to implement resource leases. Using the VIRM API, PanDA was extended to support different virtualization backends, e. g. OpenNebula and Nimbus. Both frameworks are strongly coupled to a particular backend infrastructure – Globus in the case of Nimbus and gLite in the case of PanDA. The advantage of the SAGA pilot-job is that it allows applications to seamlessly utilize different backend infrastructure, e. g. Condor, Globus and different kinds of clouds, at the same time.

SAGA [9] is a simple, POSIX-style API to the most common distributed functions, which is a sufficiently high-level of abstraction so as to be independent of the diverse and dynamic grid environments. *BigJob* is a SAGA-based pilot-job implementation [4], which can, in contrast to other pilot-job implementations, natively works independent of the underlying distributed infrastructure and across different heterogeneous backend, e. g. grids, Condor pools as well as EC2-based and Azure clouds, reflecting the advantage of using a SAGA-based approach (see Figure 2). Furthermore, the framework is extensible and provides several hooks that can be used to support other resource types, pilot-job frameworks and application-specific customization.

Figure 1 shows an overview of the SAGA BigJob implementation for computational grids. The grid BigJob comprises of three components: (i) the BigJob Manager

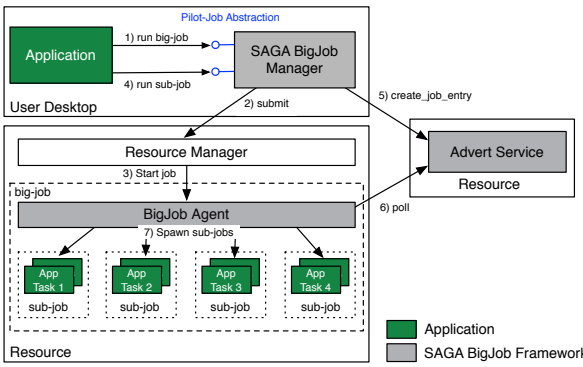

Fig. 1: BigJob Architecture: The core of the framework, the BigJob Manager, orchestrates a set of sub-jobs via a BigJob Agent using the SAGA job and file APIs. The BigJob Agent is responsible for managing and monitoring sub-jobs.

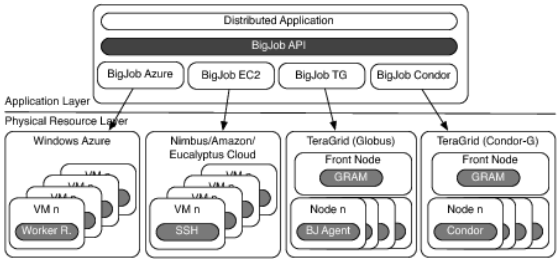

Fig. 2: **Overview of the BigJob Architecture:** BigJob can currently be used with four different infrastructures: grids, Condor pools, EC2-style and Azure clouds.

(BM) that provides the pilot-job abstraction to the application, and manages the orchestration and scheduling of BigJobs (which in turn allows the management of both big-job objects and sub-jobs), (ii) the BigJob Agent (BA) that represents the pilot-job, and (iii) the advert service which is used for communication between the BM and BA.

Applications can utilize the framework via the big-job and sub-job classes. Both interfaces are syntactically and semantically consistent with the other SAGA APIs: a sub-job for example, is described using the standard SAGA job description; job states are expressed using the SAGA state model. Before running sub-jobs, an application must initialize a big-job object. The BM then queues a job, which actually runs a BigJob Agent on the respective remote resource. For this agent a specified number of resources is requested. Subsequently, sub-jobs can be submitted through the BM using the job-id of the BigJob as reference. The BM ensures that sub-jobs are launched onto the correct resource based upon the specified job-id using the right number of processes.

IV. Azure BigJob

Figure 3 illustrates the architecture of the Azure-based BigJob. Similarly to the grid BigJob BM, the Azure BM is responsible for accepting simulation requests from the

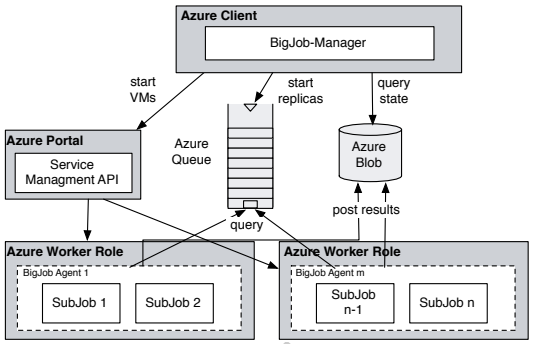

Fig. 3: **Azure-based BigJob:** The framework utilizes the queue storage for distributing work packages from the BigJob Manager to the BigJob agents running on multiple worker roles.

end-user and for orchestrating the sub-job runs. In contrast, to the grid BigJob, which utilizes different packages of the SAGA API [9], the Azure BigJob is built directly on top of the Azure APIs. With the exception of using the BigJob API – and thus presenting an interface that is identical to the SAGA-based pilot-job and that which is used for other infrastructures, Azure BigJob is self-contained and independent of any (SAGA) details used to implement a SAGA-based pilot-job.

Initially, the BM launches the requested number of worker roles using the Service Management API – a RESTful HTTP API. As part of this project we developed a Python library that can be used to access this Azure capability [10]. The BigJob Agents run within the worker roles and are implemented in C#/.NET. The main responsibility of the agent is to execute the specified MD code, (e. g. NAMD [11], or AMBER [12]), on the worker. Azure restricts communication between worker roles to a set of pre-defined endpoints. Since MPI utilizes dynamically assigned ports, MPI computations cannot be run across multiple worker roles. The size of a parallel simulation is therefore limited to the VM size (the largest Azure VM has currently 8 cores).

For each sub-job the BM creates a work-package. These are distributed to the agents using the AQS, which by offering a reliable and scalable way for delivering messages to distributed components, provide an ideal abstraction. These can, but do not have to be hosted on Azure.

Once the agents are started, they query the AQS for new work packages. If a package is found, a simulation task is started, e. g. by running the requested MD code with right parameters. The framework supports both the staging of parameter files as well as of the executable using ABS. If staging is requested in the job description, the manager uploads the respective files to the storage and the agents downloads them before the run.

The framework supports Azure affinity groups. If requested, BigJob uses affinity groups to place the VMs for the sub-jobs and the Azure storage for file staging close to

each other in the same data center. This is particularly useful for data-intensive tasks, e. g. MapReduce applications. At the same time the BM can also orchestrate worker roles that are distributed across multiple affinity groups.

V. EnMD and RE Simulations on Azure

Using the BigJob framework we have deployed and executed both the independent and coupled ensemble scenarios (enMD & REMD). Both scenarios utilize the BigJob API to create pilot-jobs, and subsequently to submit subjobs. When a pilot-job is created the BigJob framework initializes the requested number of worker roles with the specified size and launches a virtual cluster of Azure workers. The replicas themselves are submitted as subjobs. After the sub-job terminates, the application can collect results via the BigJob API. NAMD is used to perform MD simulations. In the REMD case the output is parsed to obtain the energy level of the replica. The Metropolis scheme is used to determine whether an exchange is accepted. Finally, a new generation of replicas is launched.

VI. Performance Analysis

The aim of this section is not to perform detailed systematic performance measures and analysis, but to illustrate the representative usage modes BigJob can support, and to explain how they are supported. We focus our attention on RE MD-simulations, and a workload of 8 replicas of a Hepatitis-C virus (HCV) model, with each replica running for 500 timesteps.

A. BigJob Startup Times

Initially, we analyze the overhead resulting from the usage of BigJob across different infrastructures. Typically, the main overhead when using BigJob results from either the queueing time at individual resources for grids or from the VM creation time for cloud environments. Figure 4 compares the average startup times of the different BigJob backends for grids, Condor pools and clouds. Each experiment was repeated at least 10 times.

Interestingly for experiments conducted, startup times for the cloud environments were observed to be larger than the queue waiting time for the LONI resource Poseidon. Obviously, the startup of a VM involves higher overheads than spawning a job on an already running machine: a resource for the VM must be allocated, the VM must be staged to the target and booted up. The data show that the already over-subscribed intra-cloud network and thus, the staging of the VM can be a bottleneck. Azure VMs had the longest waiting time (> 10 min; about 200 sec longer than EC2). Further, high fluctuations in waiting times can be observed. These waiting time usually correlate to the current load of the chosen data center, which again depends on external factors, such as the location of the data center and/or the date/time of the request.

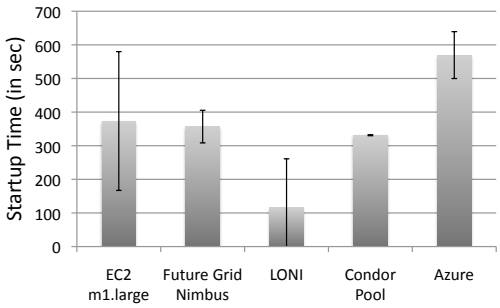

Fig. 4: **BigJob Startup Times:** In grids the startup time greatly depends upon the queuing time at the local resource manager. However, in our experiments also clouds showed a high fluctuation in the queueing time. Azure shows a slightly higher startup time than the science and the EC2 clouds.

BigJob can manage sets of VMs, and thus removes the need for applications to manage individual VMs. By utilizing the different BigJob cloud and grid backends, fluctuations in the waiting times can be smoothened out. Furthermore, BigJob provides the possibility to dynamically add resources to an application, e. g. in case a resource is heavily loaded and thus has long wait-times.

Another critical issue is the time needed to launch a subjob onto a remote resource. Figure 5 compares the grid and Azure sub-job launch times. Both BigJob implementation are comparable and introduce a low overhead of 1-2 sec for launching a single sub-job. A sub-job submission comprises of two phases: the submission phase, i. e., the writing of the sub-job meta-data to the advert or to the AQS service, and the spawning phase. In the spawning phase the agent reads the sub-job meta-data from the information service and starts the sub-job process. The Azure BigJob requires about 0.7 sec more time for submitting a subjob mainly due to the fact that Azure submissions were conducted from a remote machine outside of an Azure datacenter. In the grid case the submission machine, advert service and HPC resource were located on the same site. However, the spawning time of the Azure BigJob is much lower than of the grid BigJob. The grid BigJob agent is required to eagerly poll the advert service for new sub-job entries, while the Azure BigJob can utilize AQS, which allows threads to sleep until new queue messages arrive.

B. Ensembles on Different Resource Types

We ran experiments with NAMD in different environments with varying characteristics: FutureGrid (Nimbus), Amazon EC2, Azure and LONI/TG. Each FutureGrid VM provides 2 virtual cores and 3.7 GB memory. Amazon offers different VM types with up to 8 cores. We used, (i) the largest VM type (m2.4xlarge) with 8 cores and 68.4 GB of memory, (ii) the m1.large instance type with 2 cores and 7.5 GB, and (iii) the recently launched cluster

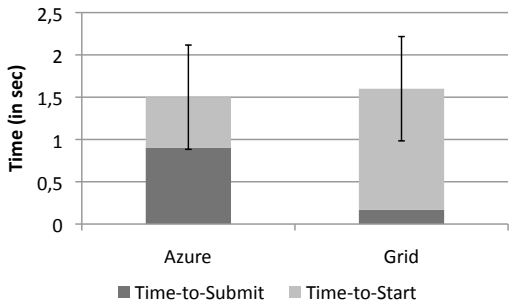

Fig. 5: BigJob Sub-Job Submission: The submission time for a single sub-job. The submission times to Azure and a grid resource are comparable. Experiments are repeated 20 times.

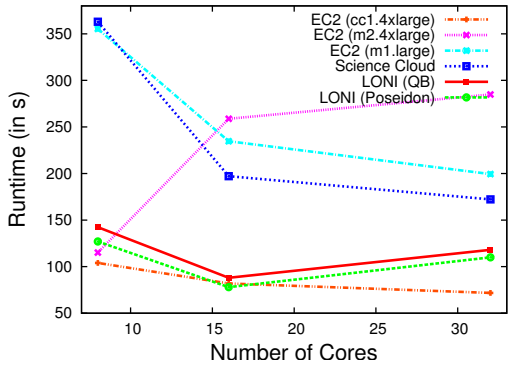

Fig. 6: **NAMD Runtimes on Different Resource Types:** The graph shows that the new EC2 cluster compute instances are able to outperform other cloud resources as well as traditional HPC resources as QB and Poseidon.

compute instances [13], which provide a pair of quad-core Intel X5570 (Nehalem) processors with 23 GB memory. In contrast to other instance types, cluster compute instances are connected using 10 Gbps Ethernet. The results of this experiment are depicted in figure 6 and show that cloud resources can achieve a comparable and in some cases even a better performance than the used grid resources.

Figure 7 compares the performances of Azure and EC2. For this purpose, an EC2 instance type with a similar core count and price is chosen. Azure outperforms EC2 Windows instances in most cases; which is noteworthy, since the costs for 2, 4 and 8 core VMs are drastically lower on Azure. Table I summarizes the costs of selected EC2 and Azure scenarios. In particular, the EC2 Linux instances show a good price/performance ratio. For Windows instances Azure provides a favorable optimum.

VM Type	CPU h	#Core	#VM	T_C	Costs
EC2 m2.4xlarge (Lin)	2.40$	8	1	128 sec	0.08$
EC2 cc1.4xlarge (Lin)	1.60$	8	1	45 sec	0.04$
EC2 c1.xlarge (Win)	1.16$	8	1	290 sec	0.09$
Azure XL (Win)	0.96$	8	1	301 sec	0.08$

TABLE I: EC2 vs. Azure Costs

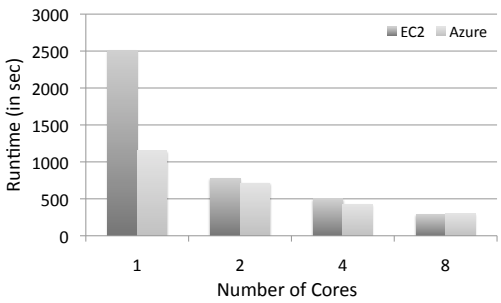

Fig. 7: **NAMD Performance on Azure and EC2:** In particular on smaller VM sizes (1-4 cores) Azure outperforms EC2. The 8 core EC2 VM cc1.xlarge shows a slightly better performance than the Azure 8 core VM, however at a higher cost.

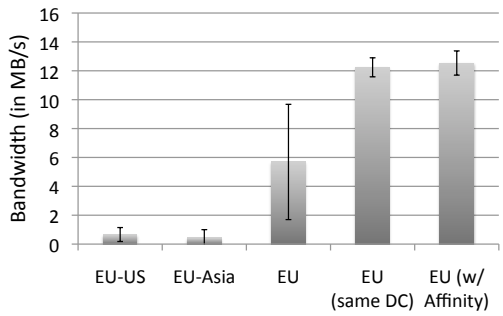

Fig. 8: **ABS Bandwidths Between Different Regions:** The achievable vary greatly with the distance between data and compute. Using affinity groups a bandwidth of approx. 12 MiB/s is achievable.

C. Data-Management on Azure

Azure provides various options to distribute data and compute. Users can either select one of the six data centers (as of Sept. 2010), or a geographic region (i. e. US, Europe, Asia). Further, Azure offers so-called affinity groups as abstractions to control the co-location of data/compute.

To evaluate the available bandwidths and storage options, we conducted the following experiment: a 4.3 GB file is stored in the ABS in the "West-EU'" data center. This file is then downloaded by worker roles in different locations: the same data center, "Anywhere US", "Anywhere Asia", "Anywhere Europe" and the same affinity group.

Figure 8 presents the results of this experiment. Obviously, the further away the worker role, the smaller the available bandwidth. Interestingly, even when staying in the same region ("Anywhere Europe") the most optimal bandwidth is not achieved.

The best result (approx. 12 MB/s) can be achieved when placing worker roles and storage in the same data center or the same affinity group. While affinity groups provide a good abstraction to manage compute/storage locations, it influences the placement solely at the data-center level. The

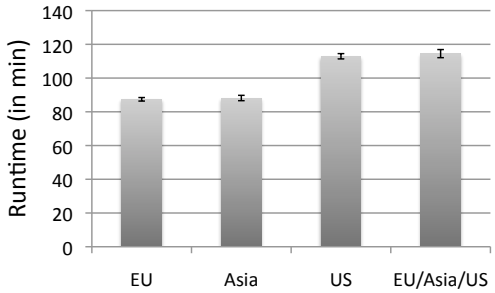

Fig. 9: T_C **for RE Run in Different Azure Regions:** Runtime for a RE simulation with 16 replicas each running on a small VM with 1 core for 500 timesteps and a total of 4 generations. The performance of the Azure fluctuates with the data center region.

same bandwidths can be achieved by a manual placement. Enhanced affinity groups that can be used to control the locality on rack/switch-level would be desirable.

While data locality is not critical for the EnMD and RE use case – the Azure service package has a size of 8 MByte, for data-intensive applications, such as MapReduce-based applications, this is an important feature. By supporting the Azure affinities within the BigJob framework, these requirements can be addressed.

D. Replica-Exchange on Azure

Figure 9 shows the time-to-completion (T_C) of an RE simulation with 16 replicas each running on a single core VM for 500 NAMD timesteps on Azure. In total, we measured the time for 4 generations, i. e., for 64 attempted exchanges. As shown in the graph, the runtime of this scenario fluctuates with the chosen Azure region. The EU and Asia show a better performance than the US data centers. BigJob also supports the distribution of sub-jobs across multiple data centers. However, due to the coupling between the replicas, the performance in this case depends on the slowest machine.

Figure 10 shows T_C for different numbers of replicas and VM types. Again, each replica is run for 500 timesteps before an exchange is attempted. As the number of replicas increases, it is expected that the coordination overhead will increase. As seen in Figure 10, this overhead for upto 32 replicas is *at maximum* 2.3 percent of the overall runtime.

The chosen VM type greatly influences T_C. Azure currently offers four types: small VMs with 1 core, medium VMs with 2 cores, large VMs with 8 cores and extra-large VMs with 8 cores. The larger the VM, the shorter the overall runtime. However, the efficiency – defined as, the run-time on one resource divided by the run-time on multiple resources scaled by the number of resources, for this problem instance on going from the 4 to 8 cores drops to less than 0.4. This is mainly a limitation of the used setup: in this scenario the replicas consist of relatively short-running NAMD tasks. For longer-running

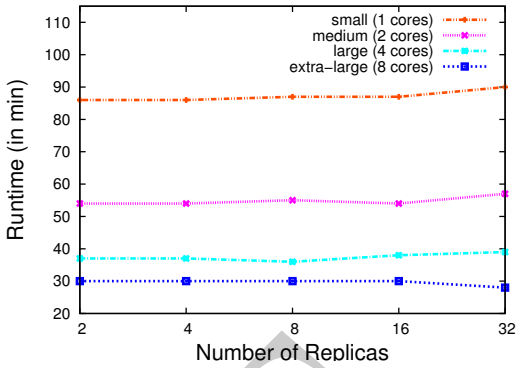

Fig. 10: T_C **for Different VM Sizes and Number of Replicas:** With an increasing number of replicas a small coordination can be observed in most scenarios. The VM type has a great impact on the overall runtime. The larger the VM, the shorter T_C .

tasks greater efficiency can be expected.

Nevertheless, the best performance is achieved on an extra-large VM. The largest ensemble comprised of 32 replicas running on extra-large VMs on a total of 256 cores. With a runtime of 30 minutes for the 16 replica case this setup is able to almost match the performance of the TG resource QueenBee, where the same scenario was executed in 26 minutes (see [3] for details).

VII. Conclusion and Future Directions

Ensemble-based MD simulations are commonly used bio-molecular simulation approaches. We have implemented the computational and coordination pattern represented by RE to Azure, by extending the BigJob framework to utilize the native abstractions provided by Azure, such as worker roles, Azure storage and affinity groups.

In contrast to other cloud offerings, Azure provides not only bare-metal VMs to applications, but a managed PaaS environment for running them. It also monitors and automatically restarts VMs and applications if necessary. Additional resources can be dynamically spawned, e. g., if a higher accuracy is required or a deadline must be met. Further, Azure provides an integrated development environment and a higher-level API, which simplifies the development considerably, in particular compared to the efforts necessary when running an application distributed across multiple TG sites (which to date does not have a system-level tool for multi-site data or job coordination). Using Azure we were able to achieve almost the same sampling speed as on comparable grid resources, such as QueenBee. Thus given its "rich" features, and simplicity invoking them, not only is Azure a viable alternative to the TG for medium-level parallel tasks, but it may even be the preferred alternative!

Future Directions: SAGA [9] provides a simple, single interface to the common distributed functionality (file/data, job/VM launch etc.) for a range of distributed infras-

tructure. Given the benefit of using the same application across different infrastructures, it makes eminent sense to support Azure from within SAGA and to provide SAGA adaptors to important Azure services. We are exploring novel abstractions, as well as extensions to those already provided by Azure for data-intensive applications; we are also investigating task placement algorithms in conjunction with different Azure worker roles, storage and affinity group setups.

Acknowledgements

SJ acknowledges the e-Science Institute, Edinburgh for supporting the research theme. "Distributed Programming Abstractions" & 3DPAS. We thank J Kim (CCT) for assistance with the RNA models, and H Kaiser (CCT) for assistance with SAGA/Windows. We thank FutureGrid and Microsoft for providing the resources for the experiments.

References

[1] http://www.microsoft.com/windowsazure/.

[2] U. Hansmann, "Parallel Tempering Algorithm for Conformational Studies of Biological Molecules," *Chemical Physics Letters*, vol. 281, pp. 140–150, 1997.

[3] A. *et al*. Luckow, "Adaptive Distributed Replica–Exchange Simulations," in *Theme Issue of the Philosophical Transactions of the Royal Society A*, vol. 367, 2009.

[4] A. Luckow and *et al*, "Saga bigjob: An extensible and interoperable pilot-job abstraction," *CCGrid10*, pp. 135–144, 2010.

[5] W. Lu and *et al*, "AzureBlast: A Case Study of Developing Science Applications on the Cloud," in *First Workshop on Scientific Cloud Computing (Science Cloud 2010)*. ACM, 2010.

[6] G. *et al*. DeCandia, "Dynamo: amazon's highly available key-value store," *SIGOPS Oper. Syst. Rev.*, vol. 41, no. 6, pp. 205–220, 2007.

[7] K. Keahey, M. Tsugawa, A. Matsunaga, and J. Fortes, "Sky computing," *IEEE Internet Computing*, vol. 13, no. 5, pp. 43–51, 2009.

[8] P.-H. Chiu and M. Potekhin, "Pilot factory – a condor-based system for scalable pilot job generation in the panda wms framework," *Journal of Physics: Conference Series*, vol. 219, no. 6, p. 062041, 2010. [Online]. Available: http://stacks.iop.org/1742-6596/219/i=6/a=062041

[9] http://saga.cct.lsu.edu.

[10] A. Luckow, "Azure Service Management API Implementation for Python," http://github.com/drelu/winazurestorage/blob/master/winazureservice.py, 2010.

[11] J. P. *et al*, "Scalable molecular dynamics with NAMD," *Journal of Computational Chemistry*, vol. 26, pp. 1781–1802, 2005.

[12] D. Case, T. Cheatham et al., "The amber biomolecular simulation programs," *J. Comp. Chem.*, vol. 26, pp. 1668–1688, 2005.

[13] W. Vogels, "Expanding the Cloud - Cluster Compute Instances for Amazon EC2," http://www.allthingsdistributed.com/2010/07/cluster_compute_instance_amazon_ec2.html, 2010.