Running Many Molecular Dynamics Simulations on Many Supercomputers

Rajib Mukherjee[1], Abhinav Thota[2], Hideki Fujioka[3],
Thomas C. Bishop[1], Shantenu Jha[*2,4]
[1]*Departments of Chemistry and Physics, Louisiana Tech University, Ruston, LA 71272*
[2]*Center for Computation and Technology, Louisiana State University, Baton Rouge, LA 70803*
[3]*Center for Computational Science, Tulane University, New Orleans, LA 70118*
[4]*Rutgers University, Piscataway, NJ 08854*
[*]*Contact Author*

***Abstract*—The challenges facing biomolecular simulations are manyfold. In addition to long time simulations of a single large system, an important challenge is the ability to run a large number of identical copies (ensembles) of the same system. Ensemble-based simulations are important for effective sampling and due to the low-level of coupling between them, ensemble-based simulations are good candidates to utilize distributed cyberinfrastructure. The problem for the practitioner is thus effectively marshaling thousands if not millions of high-performance simulations on distributed cyberinfrastructure. Here we assess the ability of an *interoperable* and *extensible* pilot-job tool (BigJob), to support high-throughput simulations of high-performance molecular dynamics simulations across distributed supercomputing infrastructure. Using a nucleosome positioning problem as an exemplar, we demonstrate how we have addressed this challenge on the TeraGrid/XSEDE. Specifically, we compute 336 independent trajectories of 20 ns each. Each trajectory is further divided into twenty 1 ns long simulation tasks. A single task requires $\approx$ 42 MB of input, 9 hours of compute time on 32 cores, and generates 3.8 GB of data. In total we have 6,720 tasks (6.7 μs) and approximately 25 TB to manage. There is natural task-level concurrency, as these 6,720 can be executed with 336-way task concurrency. Using NAMD 2.7, this project requires approximately 2 million hours of CPU time and could be completed in just over 1 month on a dedicated supercomputer containing 3,000 cores. In practice even such a *modest* supercomputer is a shared resource and our experience suggests that a simple scheme to automatically batch queue the tasks, might require several years to complete the project. In order to reduce the total time-to-completion, we need to scale-up, out and across various resources. Our approach is to aggregate many ensemble members into pilot-jobs, distribute pilot-jobs over multiple compute resources concurrently, and dynamically assign tasks across the available resources.**

I. Introduction

There are several reasons for running multiple identical simulations – often referred to as replicas or ensemble-based simulations. These range from the desire to overcome the fundamental limitation of serial execution along the dimension of time, to the need for higher accuracy and/or reduced time-to-solution. In biomolecular simulations for example, ensembles consisting of multiple replicas of the same physical system enable better sampling [1], [2], [3]. In some cases, such as free-energy calculations, multiple ensembles of the same system need to be utilized [4], [5]. Ensembles may also be required by the underlying biological complexity. Thus an important challenge in modern biomolecular simulations is the scalable, effective management and execution of ensembles – multiple instances of the same (or very similar) physical systems.

The concurrent execution of multiple ensembles thus provides an opportunity not afforded to a single long-running simulation, viz., that of task-level concurrency. Depending upon the specific replica/ensemble methodology employed, the individual ensemble-members may be either de-coupled or loosely-coupled (relative to the tight-coupling as represented by message-passing between concurrently executing threads). Whether loosely-coupled or de-coupled, the ensembles are amenable to concurrent execution – either logically or physically distributed as multiple tasks over multiple computers. Thus, in principle, the time-to-solution can be reduced on shared resources by more effectively distributing computational tasks among available resources.

Advances in computing power will always allow larger systems to be simulated on a single resource. However, this approach is fundamentally limited by the requirement of serial execution along the time dimension and to the sampling of one or a few realizations of a given system. Nonetheless, the problem for the practitioner is not running a single simulation – as the most widely utilized simulation codes (Amber, CHARMM, Gromacs, NAMD, LAMMPS) have each been optimized to run efficiently on nearly all supercomputing platforms including GPU based systems – but managing a collection of simulation tasks and their associated data. However, managing multiple simulations tasks concurrently (i.e., task-level parallelism), has not received the same attention as thread/process-level parallelism, a la message-passing.

Illustrating this is the simple fact that there exist multiple *standard* approaches to message-passing, but task-level parallelism is currently performed ad-hoc, and often with non-scalable approaches. This is true, irrespective of whether a single supercomputer is used or multiple supercomputers. Given the potential for impact, arising from task-level parallelism, the lack of scalable, efficient and general-purpose tools and approaches for managing task-level parallelism – either logically or physically distributed, is in our opinion a serious barrier facing the biomolecular simulation community.

Thus the absence of tools for the effective distribution

of high-throughput (HT), high-performance simulations is an increasingly critical gap.

Before we present the contribution and outline of the paper, we clarify our usage of the different types of scaling that this paper will be concerned with: *scale-up* – or the most common type of scaling behavior, is when a single simulation is capable of using many cores efficiently; *scale-out* is a measure of the number of ensembles that can be concurrently executed & managed; *scale-across* is a measure of the number of distinct compute resources that the ensembles can be executed on.

In this paper, we assess the ability of an *interoperable, extensible* pilot-job framework (SAGA-based BigJob [6]), to support HT simulations of high-performance molecular dynamics (MD) simulations. We demonstrate that BigJob can support hundreds of ensemble-members concurrently – where each ensemble-member is a MD simulation running on 64, 128 or more cores (scale-up). A large number of ensemble-members (scale-out) can be efficiently executed either on a single resource or across distributed supercomputing infrastructure (scale-across).

The ability to scale-across multiple machines, in principle comes with the ability to simulate more ensembles, but introduces new challenges of distributed management and coordination. We will discuss these challenges and the trade-off between the simplicity of only scaling-out versus the advantages of also scaling-across.

Section II gives an overview of related work and the computational challenges. In section III we define the scientific problem we investigate, and discuss its computational requirements and estimate the amount of compute time required. Section IV describes different methods of running multiple tasks on high-performance computing (HPC) machines. In section V we describe the experiments on various supercomputers, discuss the compute and data requirements of the scientific problem under investigation and analyze the compute and data cyber-infrastructure. In section VI we describe the results from the experiments, which demonstrate the effectiveness in scaling-out and across. In section VII, we conclude with a discussion of our work, the combined data and compute challenge, and some future work.

II. Context and Related Work

As discussed there are scientifically compelling reasons to support both the efficient simulations of long-running single simulations, as well as support ensembles comprised of many ensemble-members. Modest MD simulations by current standards include ≈100,000 to 250,000 atoms, represent over 100 ns of time and require 100's to 1000's of CPUs or GPUs in order to complete in less than one week. The associated trajectory data is in the order of 100's of GB to TB. On the most advanced supercomputers or special purpose machines, e.g. ANTON, the largest simulations include 10-100,000,000 atoms, represent microsecond time scales, and may generate 10 to 100 TB of data.

However there are growing indications that single long-running simulations maybe ineffective [7] or insufficient, and that there is a fundamental need for greater statistical averaging. Therefore a comprehensive solution needs to support efficient scale-up (thus reach long time-scales) of individual simulations, scale-out of multiple simulations as well as scale-across multiple machines.

Shared distributed infrastructure, can be considered to be of two varieties: (i) distributed computing across *best-effort resources*, such as desktop and cluster grids (e.g., @HOME) and, (ii) or distributed computing across *dedicated* HPC machines such as on TG/XSEDE or PRACE. The former typically have no fixed allocation and are purely best-effort (Open Science Grid (OSG) and @HOME); the latter have a guaranteed allocation (TG/XSEDE) but no guaranteed response time. In this paper, we will focus on the latter type, i.e., resources that are shared, and where the user has a guaranteed allocation, but no well-defined upper-bound by when tasks will execute.

The traditional method of supporting the execution of a large-number of tasks – which is referred to as high-throughput computing (HTC), has been to use a large number of small compute resources, as these have been easier to marshall. This method has been used to scale-out [8]. However, both infrastructure types can be used to support HTC; the ability to support HPC simulations in HTC mode is required in order to enable researchers to address the problem of properly sampling the complexity inherent in biomolecular systems head-on, However, few tools currently exist to effectively utilize HPC resources in HTC mode; even less to utilize multiple distributed HPC resources.

The workflow and data management for a typical molecular dynamic study is well described, see e.g. NAMD-G [9] describes how to tackle large number of independent simulations. However, NAMD-G does not provide an efficient way of using the resources as it requires manual intervention to identify availability on individual machines for simulations.

Before we outline our approach to address the challenge of HT simulations on multiple HPC resources, i.e., support the scale-up, scale-out and scale-across we will discuss the fundamental challenges inherent in ensemble-based simulations.

Computational Challenges of Multiple Ensemble Simulations

In addition to the challenge of data-management associated with HTC and HPC, a fundamental problem is the slow-down of HTC simulations on HPC machines, which arises when each job is submitted individually to the resource manager. But it does not address the root issue of typical long queue waiting times on HPC machines, In addition, in order to increase throughput, support for enhanced (distributed) resource usage or dynamic resource allocation are critical. Thus fundamental support for an ensemble of simulations with the following characters is required:

1) Usable on a range of underlying distributed resources and independent of the machine/resource-specific middleware and services (i.e. scale-across)
2) Efficiently manage both scale-up and scale-out of ensembles

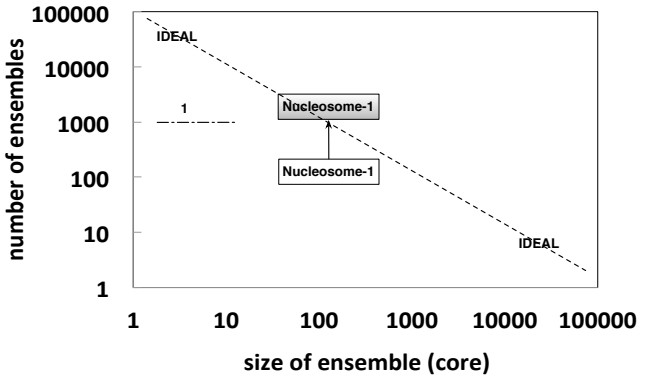

Fig. 1: Line 1 shows how most HTC execution systems behave with increasing core count of each individual ensemble-member. Most HPC ensemble-management system – consisting of home-grown (shell) scripts would typically manage O(1) ensemble-members at most, where each ensemble-member could be O(10^4) cores. In either case, with current state-of-the-practise, there is a fundamental limitation to scalability and usability. In an ideal ensemble execution system, a scientist would be able to use the same execution system over the entire range, and the scaling property would be represented by IDEAL.

3) Effective for a range of physical model sizes – from thousands of atoms to hundreds of thousands of atoms, and for varying ensemble-member size
4) Effectively manage large data transfers
5) All of the above without being tied to a specific underlying MD kernel
6) Extensible and Interoperable with emerging computational platforms such as clouds and middleware

Our analysis suggests that most existing approaches meet one or two of the criteria presented in § II, but few address more than two. A number of solutions work only on a specific resource; few, if any, ensemble-based management tools exist that provide both the interoperability necessary to seamlessly transition from one computing platform to another, across scales, or the extensibility required to utilize resources in a flexible (e.g., static resource versus dynamic resource assignment) mode.

Additionally, most methods – programming systems, tools and services, used to address these solutions are strongly associated with the underlying infrastructure. For example, many HTC on the OSG are Condor-based which does not support the fine-grained parallelism *at the scales required* required for physical problems addressed in this paper. A similar situation exists for the European Grid Infrastructure (EGI) [10], as is evidenced by Ref. [11], wherein the WISDOM [12] project uses EGEE/EGI infrastructure, but it is an infrastructure specific implementation tied to EGEE/EGI; additionally the infrastructure does not support different physical model sizes.

Fig 1 illustrates several of these points. "Nucleosome-1" is the problem we are interested in; say it currently requires O(100) concurrently running ensembles on TG/XSEDE, each requiring O(100) cores. Subsequent studies will different and thus the ensemble size (measured in number of cores used), as well as the number of ensemble-members will vary. Future studies may require 4 orders of magnitude more ensemble members; existing solution will not scale to the required level of O(10,000) ensembles.

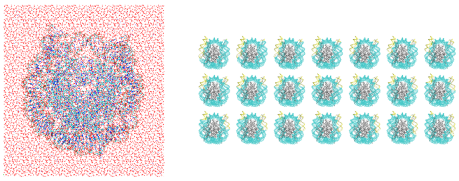

Fig. 2: Left: Each individual simulation task is a 1 ns simulation of a system containing approximately 158,000 atoms, mostly water. Right: To investigate nucleosome stability as a function of sequence we thread a 167bp long sequence onto the histone core, 147bp at a time. This yields 21 separate systems that contain a common 126bp sequence kernel. (water not shown for clarity)

What is required is an approach that enables the slicing-and-dicing of say a 10^6 cores in any way – whether 100 members of 10^4 cores each, or 10^4 members of 100 cores each. Ideally, the same ensemble execution system (Line "Ideal") would be used for all studies, rather than different execution systems for each study as currently required.

It is important to establish that the challenges of supporting ensemble-based MD simulations are different from traditional workflows (e.g, SCEC [13]), where the emphasis is on co-ordination of pre-ordered tasks; by contrast ensemble-based simulations need to be designed to support the optimal scale-out and scale-up of tasks across multiple machines. In other words, in the latter, the makespan has to be reduced by executing as many tasks concurrently as possible, whereas in traditional workflows the makespan has to be reduced by the efficient scheduling and placement of ordered tasks.

III. Scientific Problem: Computational Requirements

Nucleosomes are the building blocks of chromatin. They contain 147 base-pairs of DNA wrapped ≈1.7 turns around a protein core composed of eight histones. *In vitro* the histones occupy preferred locations on lengths of DNA greater than 147bp (i.e. positioning). The histones also exhibit preferential binding in mixtures containing different 147bp long oligomers (i.e. affinity). The physical basis for nucleosome stability and relationships between positioning and affinity remain unclear. Simple theoretical models, e.g. elastic rod models or bioinformatics based sequence analysis approaches, are not adequate for proper understanding. We therefore seek to investigate nucleosome positioning by means of all atom molecular dynamics simulations.

For this purpose, we chose the most highly occupied and least variable nucleosome, as determined by experiment, for each of the sixteen chromosomes in the yeast genome. For each of the sixteen positions, we model 21 systems, see Figure 2. One system represents the *ideal position*. The other twenty represent 10 neighboring positions on each side of the *ideal position*, i.e. one full helical repeat of the DNA in either direction. Each position is represented by an all atom

model containing ≈158,000 atoms including: 13,046 atoms of protein, ≈9600 DNA , 426 ions, and ≈135,360 water atoms (differences due to differences in DNA sequence).

In total, there are 336 systems to model, and each is simulated for 20 ns. Each 20 ns trajectory is divided into twenty tasks where each task is a 1 ns long simulation of a given position. We thus have 6,720 tasks consisting of 336 independent threads. Each of the 6,720 tasks requires ≈ 42MB of input data, runs for ≈8 hours on 32 processors, and generates 3.8GB of data or ≈25TB of data in total.

IV. HPC Resource Usage Modes (UM)

Traditionally, users manually submit jobs one at a time, to the queuing system either as a remote job from the user's desktop via SSH, or directly by logging into each resource and submitting the job locally. We refer to single job submission mode as UM-I. Each job has a task determined by the user at job-submission. For dependent tasks, a user can use dependency options provided by the batch queuing system. For example, PBS provides the '-W depend=*dependency_list*' option to define that a job run only after another job completes. The batch queue on one resource is typically unaware of the batch queue on another resource. Thus, there is no built-in mechanism to distribute M×N tasks to R number of resources efficiently. On most systems, the tasks in a simulation pipeline, which use output from one task as input for the next must be computed on the same resource if simple batch dependency rules are utilized.

The other usage mode (UM-II) is the submission of multiple task using pilot-jobs. A pilot-job decouples the task/workload execution from resource assignment, in that a pilot-job [6] is submitted without assigning any specific tasks upfront. Actual executable tasks are pulled (or pushed) from a task list after the pilot-job becomes active. An immediate advantage is that multiple tasks can be flexibly executed, but only the container job encounters any waiting in the queue. Additional advantages include support for dynamic execution models and advanced task-level scheduling[14].

BigJob [6] is a SAGA-based pilot-job that supports many pilot-jobs running concurrently on multiple heterogeneous resources; each pilot-job can support many tasks within it (sub-jobs). BigJob natively supports MPI sub-jobs. The user first *pushes* a BigJob onto a machine and once the BigJob becomes active, it pulls tasks (called *sub-jobs* in this framework), from a centralized data/task store – which in SAGA parlance is called the *Advert-Server*. Multiple BigJobs can be marshaled on distributed resources and coordinated at run-time through the advert server [15].

When using multiple BigJobs, tasks can be assigned in at least two different ways. All tasks can be manually assigned to a specific BigJob or a set of tasks can be defined such that BigJob can be instructed to pull tasks from this set of tasks. In principle, more BigJobs than needed can be launched on multiple machines, and have the sub-jobs pulled by the BigJob; *extra* BigJobs can be cancelled once all the sub-jobs are pulled; however, this violates most HPC usage policies and is *not* used in this work.

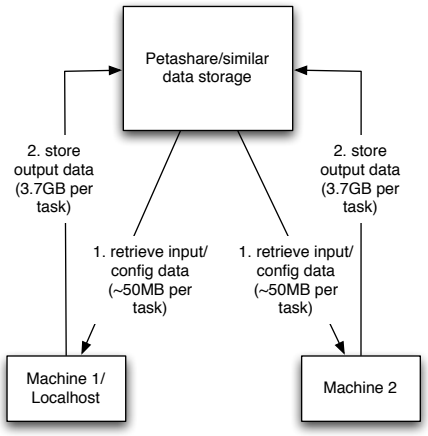

Fig. 3: This figure shows the typical data-flow in a simulation. The input data is retrieved from the data storage and copied to the machines where we are going to run the simulation. After the simulation is complete, the output data is moved to the storage. We are currently using PetaShare, which is a service provided by LONI.

SAGA-BigJob can be extended to include additional features such as data management and can easily be customized. Using this customized version, which we call the Simulation Manager (SM), we retrieve job-related files from storage resources (such as PetaShare [16]) and place them in their relevant working directories. Once that is done, we submit a pilot-job to the resource manager. When the pilot-job becomes active, the SM runs the sub-jobs according to their job-descriptions. After all the sub-jobs have finished running, the data generated is placed in storage by SM.

V. Experiments

We use BigJob for the ability to marshal multiple simulations onto multiple supercomputers. Practical information on how to use BigJob on multiple machines can be found at Ref [17].

A. Infrastructure Configuration

We used the supercomputing resources of TG/XSEDE for our simulations. XSEDE is the US NSF's flagship national cyberinfrastructure facility. The TG/XSEDE machines used in this work are, Lonestar and Ranger at Texas Advance Computing Center (TACC) and Kraken at National Institute of Computational Science (NICS).

From the benchmarks it was found that 36 cores on Kraken gives ≈3 ns/day, thus 6,720 ns would require 2,240 days on a single 36 core machine, or 22.40 days on a 3,600 core machine. Similar numbers would be applicable for Ranger. However, scaling properties varied considerably from machine to machine, as we increased the processor count from 8 to 256 cores. This provide an additional degree of freedom in our efforts to achieved the shortest time to completion of our entire ensemble. As opposed to a single simulation, our

	Lonestar	Ranger	Kraken
scratch limitation	250GB	350GB	-
CPU limit	25,656	62,976	99,072

TABLE I: Scratch space and CPU limits on different machine

concern is not with single ensemble-member performance, but a core-count for each simulation that provides greatest overall throughput. *To achieve the target in reasonable time, we need to scale-up, out and across various resources.*

B. Data Management

The input data required to start any task is ≈42MB: parm (29MB), crd (5.6MB), vel (3.7MB), coor (3.7MB), xsc (4.0KB). Each task generates 3.8GB of output data including: dcd (1.8GB) coordinate file in highly compression binary format, dvd (1.8GB) velocity file in highly compressed binary format, out (167MB) text based log file, xst (68KB) dimensions for the period cell size as function time, restart files coor (3.7MB) and vel (3.7MB) containing coordinate and velocity information. The total project represents 6.7 μs of nucleosome dynamics and ≈25TB of data. Running 50 tasks at a time would mean handling ≈2.1GB of input data and ≈188GB of output data. It is worth noting that the scratch space available and the total number of CPU's on TG/XSEDE machines are given in Table I.

For data storage and handling, we used PetaShare as a central data storage resource. Data in PetaShare can be accessed from LONI and TG/XSEDE supercomputers using a command line interface (p-commands). Figure 3 is a schematic representation of data flow.

We handled the input in different ways such as (i) by moving the data before the BigJob(s) was submitted to the queue, (ii) After the BigJob becomes active, by moving the data for each task before it was started. The first method is only viable when the tasks are assigned to the BigJob before the start of the experiment, but the compute time is not effected by data management. In the second method, the data movement happens while the job is active and therefore affects the compute time requested.

C. Run-time Configuration

TG/XSEDE machines like Lonestar, Ranger and Kraken have more than twenty thousand, sixty thousand and hundred thousand cores, respectively. For a given machine, the wait time for a BigJob varied with machine load. In some experiments, the size of the BigJob requested were roughly proportional to the core limit of the machine. In designing the experimental runs for each machine, we have taken into consideration the CPU limits and scratch space limits. The CPU limit helped to identify how we should bundle tasks so as to keep wait-times reasonable. The scratch space helped to identify the number of tasks that can be run concurrently. On some machines, scratch space limitations also dictate the maximum number of tasks that can be run concurrently, e.g. a 100GB scratch limit allows for only about 25 simulations in an ensemble.

In the first set of experiments, the number of cores assigned per task was 32 on Ranger and Lonestar and 36 Kraken. We varied the number of tasks in an ensemble to achieve various total core counts. On Ranger and Kraken, a maximum of 2,016 and 2,268 cores (about 3.5 and 1.8 percent of total cores) per BigJob for 13 hours were requested. On Lonestar we requested maximum 2,016 cores (about 10 percent of total cores) per BigJob for 12 hours.

In the second set of experiments, when there is alarge number of independent tasks, we launched ≈ 24K core BigJob on Kraken requesting 192 cores per task and ran hundreds of our simulations concurrently. In the third set of experiments, we distributed BigJobs across multiple machines. Each machine has three BigJobs of 2,016 core each. The number of cores assigned per task was 96.

VI. Results on Multiple TG/XSEDE Machines

We have analyzed individual and collective performance of three different machines, Kraken, Lonestar and Ranger. To find individual machine performance, a single BigJob was submitted on each machine and tasks were assigned to individual BigJobs. To find collective performance, multiple BigJobs were submitted from a single workstation to multiple machines, and tasks were pulled from a set of tasks as soon as a BigJob became active on any machine. The performance is measured by the total time-to-completion of the tasks, which is comprised of the sum of the wait time and run time of a BigJob in that machine.

Figure 4 shows results from the first set of experiments. Here the scale-out performance of Kraken, Ranger and Lonestar are shown. Experiments consisted of submission of individual BigJobs onto different machines. Resources for 21, 42 and 63 NAMD tasks with 32 cores per task requiring 672, 1,344 and 2,016 cores respectively were requested on Ranger and Lonestar. Same number of NAMD tasks with 36 cores per tasks were requested on Kraken. Each NAMD task was run for 1 ns. For a given machine, the average run-time remained almost constant but the average wait time varied considerably.

Wait time depends on a number of factors — system-loads, policies specific to the machine, number of processors requested (e.g., fair-share, the number of running jobs belonging to the user), wall-clock time requested. Based on run time alone, Lonestar is the fastest machine. However, when the total time of completion is considered, Ranger shows better performance. On Ranger, 63 tasks, each of whose run time was ≈12 hours, completed as one BigJob in around 15 hours, including ≈3 hours of wait time. On Lonestar, the same workload completed in ≈ 33 hours, including a wait time of ≈26 hours.

Figure 5, shows results from the second set of experiments. Here the abiltiy of BigJob to support multiple concurrenlty

	Kraken, Lonestar	Kraken, Ranger	Kraken, Ranger, Lonestar	Kraken, Ranger, Lonestar with 1/3 workload on L
1	KKK	KKK	RRR	KKK
2	LLL	KKR	KKK	KRR
3	KKL	KKR	KKR	KKK
4	KKK	KKK	RRR	KRR
5	KKK	RRR	RRR	KKK
6	KKK	RRR	KRR	LRR

TABLE II: Should be read together with the graph. Shows where the bigjobs started when run on multiple machines. R for Ranger, K for Kraken and L for Lonestar. RRR would mean all Bigjobs started on Ranger. KKR would mean the BigJobs started on Kraken and Ranger in that order.

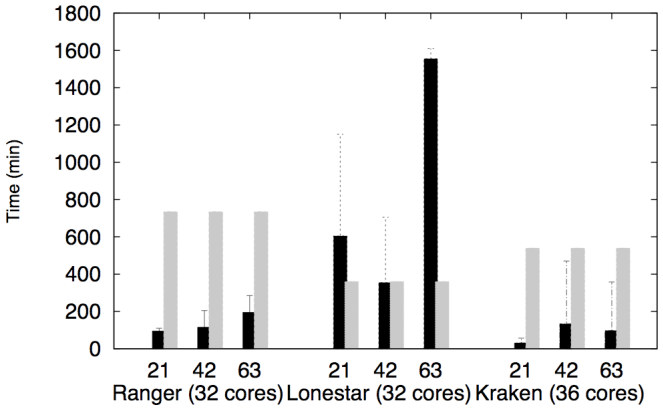

Fig. 4: The graph shows average run-times (grey) and wait-times (black) observed when running simulations using BigJob on Ranger, Lonestar and Kraken. The core count in parenthesis is the number of cores per ensemble and 21, 42 and 63 are the number of ensembles run concurrently. The error-bars show the maximum value observed in the experiments. Each run was repeated two or three times.

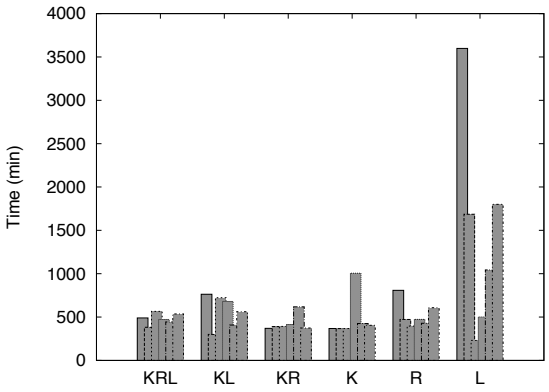

Fig. 6: The figure compares the time to completion (including waiting time) when the BigJobs are submitted to only one machine and when BigJobs are submitted to multiple machines. Here we submit jobs to Kraken, Ranger and Lonestar and assign the workload to the BigJob that becomes active first. This way we are able to select the BigJob with the least waiting time. The time to completion includes queue waiting time. In each case we submitted 3 BigJobs per machine of size 2,016 processors and assigned 96 cores per sub-job.

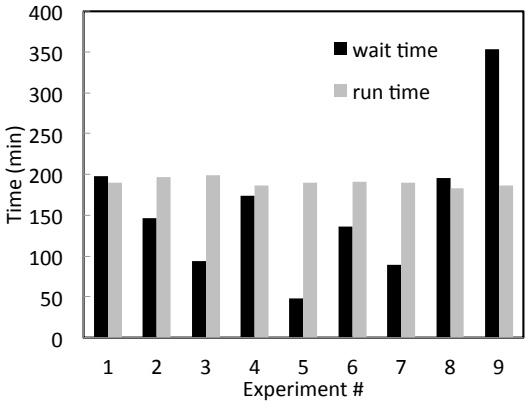

Fig. 5: BigJob simulations on Kraken: In each run, 24,192 cores were requested with a single BigJob and 126 sub-jobs were launched with 196 cores each. The run time of the sub-jobs varies and the median run time is shown. Total time to completion for each run would be the sum of wait-time and run-time.

executing tasks/sub-jobs (scale-out) is shown on Kraken. A BigJob of 24,192 cores was launched with 126 sub-jobs with 192 cores each. We repeated this experiment nine times. The waiting time and median run time of each experimental run is shown on the figure. The wait time is comparable to BigJobs requesting 1,512 or 2,268 cores (see Fig. 4) and found to be as low as fifty minutes. Thus we have achieved ten times more throughput with no additional cost. The run time was not identical for all the sub-jobs; the difference between the first and last to finish was however negligible compared to the total run-time.

High throughput simulations can be obtained by not only scaling-out, but also by scaling-across multiple machines. Fig. 6 shows the results from the third set of experiments. We observed the performance of different machines taken together and compared it with individual machine. We observed the performance using two machines, Kraken-Lonestar or Kraken-Ranger, and using three machines Kraken-Ranger-Lonestar.

Figure 6 shows the scale-across performance. It was measured by the time to completion of three BigJobs with 2,016 cores each. When using only one machine, all three BigJobs were submitted to that particular machine and the time to completion was noted. In case of experiments where multiple machines were involved, three BigJobs were submitted to each machine, and the time to completion of the first three BigJobs were noted. The first three BigJobs to execute in different configurations, i.e., the set of machine combination was varied. In this way we were able to find the minimum time to completion for three BigJobs and all the machines involved in that particular experiment were in use and none of them were idle. Note, that although we refer to these as "experiments", these were in fact real production science runs hence we did not terminate the remaining 6 submitted BigJobs but used their output for our science results.

Several experimental runs were performed when multiple machines were involved as seen in Fig. 6. When individual machines were used, Kraken had better performance than Ranger in three of the four cases and both Kraken and Ranger had better performance than Lonestar except for two cases. As the size of the BigJobs were similar, different waiting-times are to be expected; in fact, the wait-times observed are consistent with the relative sizes of the machines. These differences in performance can be expected from the capacity of the three machines.

Table 2 shows how the first three BigJobs to execute were distributed among the machines involved in that particular experiment. Most of the BigJobs started on Kraken before Ranger or Lonestar. The dominant size of Kraken explains why the performance of Kraken-Ranger together was better than Ranger but only better than Kraken in one case. The performance of Kraken-Lonestar was better than Lonestar but not better than Kraken alone. This is also from the fact that Kraken dominates and only once all the three BigJobs started in Lonestar before that of Kraken.

When all three machines were used, the performance was comparable with Kraken-Ranger and slightly better than Kraken or Ranger or Kraken-Lonestar but much better than Lonestar alone. This was because both Kraken and Ranger performed similarly and BigJob never started on Lonestar before Kraken or Ranger. Table 2 also shows data from Kraken-Ranger-Lonestar together with Lonestar having only 1/3 workload as that of Kraken or Ranger. This was keeping in mind the number of cores on Lonestar which is about 1/3 of Ranger and 1/5 of Kraken. During these experiments, BigJob has started only once in Lonestar before Kraken or Ranger.

Running on multiple machines showed that the overall performance was governed by the machine with shortest average wait-time. Using multiple machines resulted in better performanace, as a consequence of being able to avoid instances when a particular machine might be busy. A detail observation of Kraken, Ranger and Kraken-Ranger together showed that we never encountered a long run time using multiple machines even for a single case. For almost similar performing machines like Kraken and Ranger, the instance when one machine was busy, BigJobs had started in another machine.

These experiments were repeated over a relatively large window of time (i.e., weeks), and thus we believe have represented system-loads. It is difficult to determine the optimal BigJob size to submit to a given machine; however given the flexibility in BigJob size choice (not typical of static resource execution models with single MPI jobs), the availability of advanced information services (such as BQP, which provide information on typical wait-times as a function of requested number of cores etc.), will make it easy to "tune" the size of a BigJob on a given machine at a given instant of time so as to have optimal characteristics (i.e, minimal wait-time with some bounds on the number of cores).

VII. Conclusions

We describe the various issues that we had to address in order to use these resources effectively, some of which are unique to the fact that we used multiple distributed resources and not just a single resource.

Deployment: Software deployment on HPC is non-trivial; we faced challenges associated with deploying software to support distributed computing in user-space. Motivated by some of our problems, SAGA was deployed the equivalent of system-space (in a "CSA" area).

As our "experimental runs" for collecting performance measures were actually science production runs, we desired to use the same version of NAMD on all machines, and thus had to ensure the availability of the correct version of NAMD [18] – as default versions of NAMD on different XSEDE supercomputers are different. We installed NAMD 2.7 on all machines ourselves. We needed to make some small changes to the SAGA-BigJob framework as NAMD ibverbs 2.7 with charmrun reads the nodefile (the file which contains the list of processors to run on) differently.

Run-time Experience: The run-time issues are primarily related to data-management, failures and data-transfer times. We also encountered hardware failures, network/transfer failures and application-level failures (either NAMD or communication with the SAGA-based advert service). After a job starts, a task may fail due to corrupt nodes or a failure of data transfer from PetaShare. On Ranger, we witnessed a socket close before a receive, which is a segmentation fault and this happened randomly. In some cases a sub-job (simulation task) within BigJob did not run to completion in the requested wall-time, whilst other tasks completed successfully. The task in question ran slowly due to file input/output problem. Jobs also failed if/when PetaShare was unavailable and the SM was not able to dock necessary input files.

When there are a large number of ensembles under a single BigJob, there is an overhead associated with managing the sub-jobs. After a BigJob becomes active in queue, the actual time it takes to launch a sub-job and mark a sub-job as done depends on the HPC machine used and location of the advert-server. However, even for over a hundred concurrently executing sub-jobs, that adds up to less than a few minutes, while the total run-time is $\approx$3 hours. In case both the advert server and the HPC machine are co-located and are on the same network, this overhead is further reduced.

Even though the scale-across of many ensembles, is susceptible to both distributed latencies and slowdown due to "inefficient" distributed communication, the overall performance is well within acceptable ranges, especially when set against the significant reduction in time-to-solution due to increased throughput. Thus, our results show that distributed cyberinfrastructure provides a promising environment to support the execution of high-throughput of high-performance simulations. Amongst other advantages, we have demonstrated a reduced time-to-solution compared to when a single resource was used.

However there are new challenges that arise as a consequence of using high-end cyberinfrastructure in distributed

mode. Sophisticated software/middleware requirements and a myriad range of faults are just some. By its very nature scale-across introduces a data locality challenge.

It is a fair conclusion that we have "tamed" some of the challenges (though some of the issues related to distributed coordination on distributed systems remain vexing) of supporting the computing requirements of ensemble-based molecular simulations; however, as our ability to simluate more ensembles increases over distributed resources increases the data-compute locality challenge increases.

In our case the data shared between tasks (simulation restart files) was relatively small compared to the output data set. This is the ideal scenario as we can schedule our tasks to run anywhere and merely retrieve the necessary inputs from where ever they reside once the job is active. Upon completion we can schedule the deposition of return data into an archive, giving priority to the small subset of data needed to restart the next task. However, this is not always the case. In fact, data analysis of the output of biomolecular simulations has contrasting characteristics, and thus the opposite problem: it involves transferring large data-sets for analysis and the return data set is comparatively small.

In general, it is not just storage capacity, whether remote or local, that matters, but the network bandwidth that connects compute resources and storage resources; it is possible that bandwidth become the limiting factor for analysis problems involving large data-sets.

Future Work: As alluded to, an information service that can be utilized intelligently to automate the selection of machines and the size of the BigJob would greatly improve overall performance. Such a system must also be *data aware*, i.e. sensitive to data locality issues. In fact this has been shown to be the case for an analogous problems, although it has not been investigated at the scale of the problem here [19]. Intelligent task placement will only increase the pressure for intelligent data-placement and fault-tolerance.

Not only will we increase the throughput on homogeneous distributed cyberinfrastructure, we will extend our capabilities to include heterogeneous (in size and scale, not just middleware/software environment) cyberinfrastructure (possibly via interoperation across grids). The abstractions and approaches employed here are likely to be applicable, as valid programming models and development methodologies for using high-end infrastructure i.e., treat thousands, if not millions of large-scale ensembles as a *logically distributed* scientific application on peta/exa-scale machines.

ACKNOWLEDGEMENT

This work is part of the Cybertools (http://cybertools .loni.org) project (PI Jha) and primarily funded by NSF/LEQSF (2007-10)-CyberRII-01 and NIH Grant Number P20RR016456 from the National Center For Research Resources. Important funding for SAGA has been provided by the UK EPSRC grant number GR/D0766171/1 (via OMII-UK) and HPCOPS NSF-OCI 071087j. Bishop were supported by NIH-R01GM076356. Mukherjee were supported by NSF award number EPS-1003897 for LA-SiGMA. This work has also been made possible thanks to computer resources provided by TeraGrid TRAC award TG-MCB090174 (Jha) and TG-MCB100111 (Bishop), and LONI resources.

REFERENCES

[1] U. Hansmann, "Parallel tempering algorithm for conformational studies of biological molecules," *CHEMICAL PHYSICS LETTERS*, vol. 281, no. 1-3, pp. 140–150, DEC 19 1997.

[2] D. Earl and M. Deem, "Parallel tempering: Theory, applications, and new perspectives," *PHYSICAL CHEMISTRY CHEMICAL PHYSICS*, vol. 7, no. 23, pp. 3910–3916, 2005.

[3] Y. Sugita and Y. Okamoto, "Replica-exchange molecular dynamics method for protein folding," *CHEMICAL PHYSICS LETTERS*, vol. 314, no. 1-2, pp. 141–151, NOV 26 1999.

[4] E. Darve and A. Pohorille, "Calculating free energies using average force," *The Journal of Chemical Physics*, vol. 115, no. 20, pp. 9169–9183, 2001. [Online]. Available: http://link.aip.org/link/?JCP/115/9169/1

[5] C. Chipot and J. Henin, "Exploring the free-energy landscape of a short peptide using an average force," *The Journal of Chemical Physics*, vol. 123, no. 24, p. 244906, 2005. [Online]. Available: http://link.aip.org/link/?JCP/123/244906/1

[6] A. Luckow, L. Lacinski, and S. Jha, "SAGA BigJob: An Extensible and Interoperable Pilot-Job Abstraction for Distributed Applications and Systems," in *The 10th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, 2010, pp. 135–144. [Online]. Available: http://doi.ieeecomputersociety.org/10.1109/CCGRID.2010.91

[7] J. Roy and C. A. Laughton, "Long-timescale molecular-dynamics simulations of the major urinary protein provide atomistic interpretations of the unusual thermodynamicsof ligand binding," *Biophysical Journal*, vol. 99, no. 1, pp. 218 – 226, 2010. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0006349510004212

[8] S. M. Larson, C. D. Snow, M. Shirts, V. S. P, and V. S. Pande, "Folding@home and genome@home: Using distributed computing to tackle previously intractable problems in computational biology."

[9] M. Gower, J. Cohen, J. Phillips, R. Kufrin, and K. Schulten, "Managing biomolecular simulations in a grid environment with namd-g," *In Proceedings of the 2006 TeraGrid Conference, 2006.*, 2006.

[10] F. Gagliardi, B. Jones, F. Grey, M.-E. Bgin, and M. Heikkurinen, "Building an infrastructure for scientific grid computing: status and goals of the egee project," *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 363, no. 1833, pp. 1729–1742, 2005.

[11] N. Jacq, V. Breton, H.-Y. Chen, L.-Y. Ho, M. Hofmann, V. Kasam, H.-C. Lee, Y. Legr, S. C. Lin, A. Maa, E. Medernach, I. Merelli, L. Milanesi, G. Rastelli, M. Reichstadt, J. Salzemann, H. Schwichtenberg, Y.-T. Wu, and M. Zimmermann, "Virtual screening on large scale grids," *Parallel Computing*, vol. 33, no. 4-5, pp. 289 – 301, 2007, large Scale Grids. [Online]. Available: http://www.sciencedirect.com/science/article/B6V12-4N3WYF2-1/2/e5c8d221738d976fc4596e62b703d027

[12] H.-C. Lee, J. Salzemann, N. Jacq, H.-Y. Chen, L.-Y. Ho, I. Merelli, L. Milanesi, V. Breton, S. Lin, and Y.-T. Wu, "Grid-enabled high-throughput in silico screening against influenza a neuraminidase," *NanoBioscience, IEEE Transactions on*, vol. 5, no. 4, pp. 288 –295, dec. 2006.

[13] Y. Cui, K. B. Olsen, T. H. Jordan, K. Lee, J. Zhou, P. Small, D. Roten, G. Ely, D. K. Panda, A. Chourasia, J. Levesque, S. M. Day, and P. Maechling, "Scalable earthquake simulation on petascale supercomputers," in *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 1–20. [Online]. Available: http://dx.doi.org/10.1109/SC.2010.45

[14] S.-H. Ko, N. Kim, J. Kim, A. Thota, and S. Jha, "Efficient runtime environment for coupled multi-physics simulations: Dynamic resource allocation and load-balancing," in *Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*, ser. CCGRID '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 349–358. [Online]. Available: http://dx.doi.org/10.1109/CCGRID.2010.107

[15] A. Thota, A. Luckow and S. Jha, *Efficient Large-Scale Replica-Exchange Simulations on Production Infrastructure*, submitted to Philosophical Transactions of the Royal Society A: Mathematica l, Physical and Engineering Sciences; draft at http://saga.cct.lsu.edu/publications/papers/confpapers/erelpri.

[16] M. Balman, I. Suslu, and T. Kosar, "Distributed data management with petashare," in *Proceedings of the 15th ACM Mardi Gras conference: From lightweight mash-ups to lambda grids: Understanding the*

spectrum of distributed computing requirements, applications, tools, infrastructures, interoperability, and the incremental adoption of key capabilities, ser. MG '08. New York, NY, USA: ACM, 2008, pp. 19:1–19:1. [Online]. Available: http://doi.acm.org/10.1145/1341811.1341833

[17] "SAGA-BigJob," http://faust.cct.lsu.edu/trac/bigjob.

[18] J. Phillips, R. Braun, W. Wang, J. Gumbart, E. Tajkhorshid, E. Villa, C. Chipot, R. Skeel, L. Kale, and K. Schulten, "Scalable molecular dynamics with NAMD," *Journal of Computational Chemistry*, vol. 26, pp. 1781–1802, 2005.

[19] Y. E. Khamra and S. Jha, "Title: Developing Autonomic Distributed Scientific Applications: A Case Study From History Matching Using Ensemble Kalman-Filters," in *GMAC '09: Proceedings of the 6th international conference industry session on Grids meets autonomic computing*. New York, NY, USA: IEEE, 2009, pp. 19–28.