Scalable Online Comparative Genomics of Mononucleosomes: A BigJob

Jack A. Smith
Marshall University
Huntington, WV 25755
smith1106@marshall.edu

Melissa Romanus
Rutgers University
Piscataway, NJ 08854
melissa.romanus@rutgers.edu

Pradeep Kumar Mantha
Lawrence Berkeley Lab
Berkeley, CA 94720
pkmantha@lbl.gov

Yaakoub El Khamra
University of Texas - TACC
Austin, TX 78758
yye00@tacc.utexas.edu

Thomas C. Bishop
Louisiana Tech University
Ruston, LA 71272
bishop@latech.edu

Shantenu Jha
Rutgers University
Piscataway, NJ 08854
shantenu.jha@rutgers.edu

ABSTRACT

Our goal is to develop workflows for simulating arbitrary collections of mononucleosomes in atomic detail as an on demand analysis tool for online comparative genomics. The limiting factor is resource availability. The aim of this paper is to document and share our experiences in providing a general-purpose, easy-to-use and extensible solution for such computations. At the core it involves supporting the execution of high-throughput workloads of high-performance biomolecular simulations on one or more XSEDE machines. Although conceptually simple, it is still a difficult practical problem to solve, especially in a flexible, robust, scalable manner. Specifically, we employ BigJob– an interoperable Pilot-Job. The bulk of this paper is about our experience in executing a very large number of ensembles including the associated non-trivial data management problem. Our experience suggests that although a nascent and fledgling technology, BigJob provides a flexible and scalable Pilot-Job to support workloads that were hitherto not easy, if not impossible.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous; D.2.8 [**Software Engineering**]: Metrics—*complexity measures, performance measures*

General Terms

Experience, Technology

Keywords

HPC, Distributed Computing, NAMD, MD, Large Scale, BigJob, SAGA, Python, Computational Workflow, XSEDE resources, comparative genomics, chromatin, nucleosome positioning

XSEDE13 '13 San Diego, CA USA

1. INTRODUCTION

The need to run many distinct instances of molecular dynamics (MD) simulations concurrently arise in many different scientific contexts: enhanced sampling and configurational space exploration to name just a couple. As a specific critical example of a scientific problem that depends upon the ability to execute and control large-scale ensembles of high-performance computing, we investigate workflows for simulating ensembles of mononucleosomes in atomic detail. Nucleosomes are the fundamental structural unit of eukaryotic genomes. As such, the nucleosome's structure and dynamics is relevant to all genomic processes. By online comparative genomics, we mean the utilization of DNA sequence data, molecular modeling and biophysical data, namely nucleosome positioning and x-ray crystallographic structures, to compare the structure and dynamics of mononucleosomes in atomic detail. Such comparisons cannot be obtained with traditional bioinformatics techniques or experimental methods.

Our goal is to develop workflows for simulating arbitrary collections of mononucleosomes in atomic detail as an online tool for comparative genomics that complements and extends traditional bioinformatics tools or coarse grain models such as our Interactive Chromatin Modeling (ICM) server at Louisiana Tech (www.latech.edu/b̃ishop) [21].

In this study, we seek to localize and compare 5 potential nucleosome sites, where we need to simulate 105 mononucleosome configurations for 20 nanoseconds This is a non-trivial undertaking and requires support for scalable, flexible and advanced execution modes. Since the 105 configurations are independent, a properly implemented solution, give sufficient resources, allows us to complete the study in as little as 1 day. Here we demonstrate that on-line comparative genomics of mononucleosomes is feasible. The aim of this paper is to document and share our experience in an attempt to execute high-throughput workloads of high-performance biomolecular simulations on multiple XSEDE machines, Lonestar and Kraken.

Whereas single high-performance simulations are commonplace and the norm, it still remains a challenge to execute many instances of a high-performance simulation concur-

rently while also managing their dependencies. Contributing to the problem is the fact that most supercomputing centers, including XSEDE, have their environments tuned to support mostly single-job oriented work loads. On the one hand, although workflows and gateways have improved the mix of work loads supported, they do not present a natural user interface for multiple dependent instances. In response to the deficiency, there are many tools and *ad hoc* solutions. There does not exist a general purpose, uniform and flexible approach that can be utilized across XSEDE resources. Our approach builds upon important but incremental advances – both conceptual and implementation, of Pilot-Jobs (a conceptual model of which can be described by the P Model [14]). Specifically, we employ BigJob– an interoperable Pilot-Job. We develop a pilot script using the Pilot-API to express the abstract workflow associated with the high-throughput workload, an ensemble of MD simulations in this case.

An important difference between this paper and a previous related paper [18], investigating BigJob's use for nucleosomes, is that this work is focused on understanding and improving simulation workflow (with dependencies), on either a single machine or distributed across multiple machines; additionally, the science problem being addressed in the current publication is distinct from Ref. [18].

Our paper is logically organized into the following sections. After the introduction and the underlying scientific motivation (§1), we discuss the basics of Pilot-Jobs, we introduce BigJob and the interoperability layer (SAGA) and how they couple to provide an interoperable and flexible Pilot-Job. We then outline the abstract workflow of the scientific problem of interest, and examine how we use the Pilot-API to implement it. The bulk of this paper is about our experience in executing a very large number of ensembles including the associated non-trivial data management problem. Our experience suggests that although a nascent and fledgling technology, BigJob provides a flexible and scalable Pilot-Job to support workloads that were hitherto not easy, if not impossible.

2. SCIENTIFIC PROBLEM

A genome is more than just a sequence of DNA. In eukaryotic organisms it exists as a biomolecular complex of DNA and proteins called chromatin that resides inside the cell nucleus[16]. The length of DNA is typically many orders of magnitude larger than the diameter of the nucleus. But, the width of DNA is about 2 nm so its volume is less than the nucleus. The trick it to wind and fold the DNA strand into a compact form. Herein lies the problem. Folding affects access to the instructions encoded in the DNA and therefore all biologic mechanisms that require these instructions. The proteins that affect DNA folding are called histones. Eight histones associated as two dimers and a tetramer ($[H2A-H2B][H3-H4]_2[H2A-H2B]$), wrap 147 base pairs of DNA into 1.7 turns of a superhelix. The histone-DNA complex is known as the nucleosome. To a first approximation, folding of the DNA, or at least local access, is determined by the location and conformation of all nucleosomes on a chromosome. X-ray crystallographic studies provide a number of atomic resolution structures of the nucleosome[19], but because of the experimental difficulties in growing crystals for an arbitrary nucleosome, all structures to date contain essentially the same sequence of DNA and

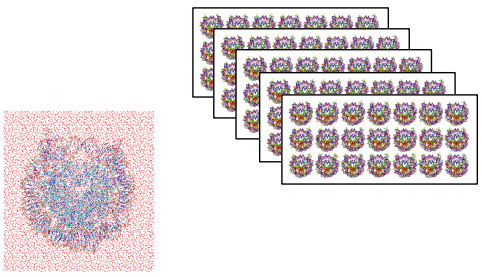

Figure 1: Left: Each individual simulation task is a 1 ns simulation of a system containing approximately 158,000 atoms, mostly water. Right: To investigate nucleosome stability as a function of sequence we thread a 167bp long segment of DNA onto the histone core, 147bp at a time. This yields 21 separate systems that must be simulated. The entire simulation ensemble includes 5 sets of 21 systems, as pictured. (Water not shown for clarity.)

represent only one conformation of the nucleosome. The nucleosome is a dynamic entity[7] that can exist in various states of association[23]. A canonical octasome can be formed with $\mathcal{O}(4^{147}) \approx (10^{88})$ different sequences of DNA. An exhaustive study of nucleosome structure and dynamics is not possible. Our goal is to develop computational techniques that allow us to compare ensembles of nucleosomes for an arbitrary realization of the nucleosome. Having such capabilities available on demand provides a powerful tool for comparative genomics. Our current studies extend our initial investigations of the relationships between nucleosome conformation and dynamics and the material properties of DNA as a function of DNA sequence [17, 18]. To achieve this end we thread different sequences of DNA onto available x-ray structures of the nucleosome. Computationally, we are deliberately mis-positioning (translating) nucleosomes to determine the rules governing nucleosome positioning.

2.1 Nucleosome Modeling Study

In our most recent study [18], we simulated 336 nucleosome candidates (comprising 21 translations about 16 different sites) that were experimentally determined to be associated with the most highly occupied and least variable nucleosome positions in the yeast genome[11]. This simulation study contained 50 times more nucleosome systems than in any other molecular dynamics study of the nucleosome, and collectively it represents 6.7 microseconds of dynamics, approximately 8 times more than any other nucleosome simulation study [6].

In the current study, we chose five sequences representing 167 base pairs of DNA (147 base pairs bound to a histone core plus 20 translations), that are known via experiment to be nucleosome free [11]. For each of the five nucleosome-free regions, we model 21 systems, see Figure 1. One system represents the 147 base pair in the center of the sequence. The other twenty represent the 10 neighboring positions on each side of the center position, i.e. one full helical repeat of the DNA in either direction. Threading one helix repeat of the DNA around the histone core allows us to investigate the contribution from sequence specific defects in the DNA, e.g. DNA bends or highly flexible regions. As DNA is thread around the histone core such defects will be oriented toward, away and then back toward the histone core.

Here we model 105 systems (comprising 21 translations

about 5 different sites). Each is simulated for 20 ns using NAMD [20] as the compute engine and the force field parameters distributed with Amber [8]. This is the typical simulation methodology for nucleosome simulations [6]. To keep the simulations tractable, each 20 ns trajectory is divided into twenty 1 ns long simulations (tasks). We thus have 2,100 tasks consisting of 105 independent threads (chained sequences). Each of the 2,100 tasks requires 41MB of input data, runs for about 1 hour on Lonestar (or about 4 hours on Kraken) using 240 processors, and generates 3.6GB of data. In total this is nearly 7.6TB of data and over 500,000 SUs of compute time on Lonestar (or 2,000,000 SUs on Kraken).

Aside from the DNA sequence, the systems simulated are nearly identical to our previous study[18]. Each system is represented by an all-atom model containing ≈158,000 atoms including: 13,046 atoms of protein, ≈9600 atoms of DNA, 426 ions, and ≈135,084 water atoms (differences due to differences in DNA sequence).

3. SOFTWARE TOOLS AND COMPUTATIONAL INFRASTRUCTURE

XSEDE is inherently a complex infrastructure with heterogeneous resources. In order to harness the power of such a distributed environment, we utilize Pilot-Jobs. A Pilot-Job is a mechanism by which a proxy for the actual simulations is submitted on the resource to be utilized; this proxy, in turn, conveys to the application the availability of resources and also influences which tasks are executed. The abstraction of a Pilot-Job generalizes the reoccurring concept of utilizing a placeholder job as a container for a set of compute tasks; instances of that placeholder job are commonly referred to as Pilot-Jobs or pilots.

In general, Pilot-Abstractions provide a suitable means to orchestrate heterogeneous sets of both compute and data resources and support the efficient utilization of different kinds of commercial as well as science cloud resources. Pilot-Abstractions have been extensively used [14, 13, 15] on both HPC and HTC infrastructures for a range of application scenarios as a resource management abstraction to, (i) improve the utilization of resources, (ii) to reduce wait times of a collection of tasks, (iii) to facilitate bulk or high-throughput simulations where multiple jobs need to be submitted which would otherwise saturate the queuing system, and (iv) as a basis to implement application-specific execution, scheduling and policy decisions

The P* model [14], a model for Pilot-Abstractions, works to clearly define the computation and data components of a distributed application as 'compute units' and 'data units' in the context of Pilot-Jobs and Pilot-Data. A compute unit describes a self-containing piece of work, e.g. a computational task that potentially operates on a set of input data, while a data unit is a container for a logical group of data that is often accessed together or comprises a larger set of data; e.g. a data file or chunk.

3.1 BigJob: A Pilot-based Framework

BigJob is a Pilot-Job system implementation which provides a framework for running many types of distributed applications – including but not limited to very-large scale parallel simulations, many small high-throughput simulations, or ensemble-based workflows. Consistent with the P* model, BigJob [4, 1, 12] provides a unified run-time environment for Pilot-Jobs on heterogeneous infrastructures. For this purpose, BigJob provides a higher-level, unifying interface to heterogeneous and/or distributed data and compute resources. The framework is accessed via the Pilot-API, which provides two key abstractions: Pilot-Job and Pilot-Data.

Applications can specify their resource requirements using a Pilot description. In the compute case, the user typically specifies the application to run as well as the number of cores required by their application. Pilots are started via the Pilot-Compute Service. BigJob eliminates the need to interact with different kinds of compute resources, e. g. batch-style HPC/HTC resources as well as cloud resources, and provides a unified abstraction for allocating resources.

BigJob has seen its widest usage across the heterogeneous resources that XSEDE provides. Simple installation into user space on any resource that supports Python 2.5 or greater makes the uptake of BigJob easy for the end user. BigJob supports thousands of jobs and millions of SUs on XSEDE. It has been at the heart of two recent and successful ECSS projects [18].

3.2 SAGA: Interoperability Layer

In order for BigJob to work on heterogeneous resources, it requires an interoperability layer which provides access to a variety of middleware. This is achieved through the use of the Simple API for Grid Applications (SAGA). SAGA defines a high-level access mechanism for distributed infrastructure components like job schedulers, file transfers, and resource provisioning services. Given the heterogeneity of distributed infrastructures, SAGA provides a much needed interoperability layer that lowers the complexity and improves the simplicity of using distributed infrastructure whilst enhancing the sustainability of distributed applications, services, and tools.

SAGA is an Open Grid Forum (OGF) recognized standard (GFD.90). It allows developers of distributed applications to construct higher-level functionality and abstractions, such as gateways, workflows, application management systems, and run-time environments. The key advantages to running with SAGA on XSEDE is that users do not need to worry about the individual batch queuing systems implemented on the various machines. Using the SAGA API and appropriate job adaptors, the different submission mechanisms for these queuing systems is handled on the SAGA backend, which is transparent to the user.

The SAGA API has been used to provide almost complete coverage over nearly all grid and distributed computing middleware/systems, including but not limited to Condor, Genesis, Globus, UNICORE, SGE, LSF/PBS/Torque, and Amazon EC2.

3.3 Deployment of BigJob

Both SAGA and BigJob are designed in such a way that they can be easily installed into the home directory of a user using the Python Package Index (PyPi). This eliminates the need to have root access to a system and alleviates many common errors that arise from installation dependencies. SAGA is packaged within BigJob, so users do not have to install two separate modules.

The main deployment of BigJob is on the XSEDE infrastructure. XSEDE is a heterogeneous environment, and oftentimes users have custom python versions or installations. In order to not disturb these custom environments, BigJob is

recommended to be installed using a virtual environment. A virtual environment allows a user to create a local Python software repository in his or her home directory that behaves exactly like the global Python repository, except that it grants the user *write* access to it. In order to use the virtual environment, the Python version must be 2.5 or greater. Since some XSEDE machines use Python 2.4 as the default python version, it may be required to load a python module file before installing BigJob.

After activating the virtual environment, the BigJob python package can be installed by typing:

```
easy_install bigjob
```

In addition to the BigJob package, the BigJob python dependencies, including the SAGA package, are also installed. The SAGA package includes the proper adaptors for a wide variety of middleware systems. This allows the user to submit jobs to any of the XSEDE batch queuing systems.

BigJob requires SSH password-less login to the machines and a redis server [2] running either locally or on a remote server. The redis server is used for coordination of the Pilot-Job and its compute and data units. For the purposes of this project, we utilize a private redis server hosted on a virtual machine at Indiana University. The BigJob documentation [3] provides instructions to users for setting up their own redis server.

In order to provide a more seamless uptake of BigJob by users, we will provide an open-access redis server available on XSEDE. This avoids the overhead of new users having to start a redis server on an XSEDE machine's head node or on their local machines. This effort is currently underway with XSEDE ECSS staff to make the server only accessible to registered users of XSEDE.

After following the aforementioned steps, users will be able to write their own BigJob submission scripts using Python. These scripts can range from simple ensemble-based simulations to more complicated workflows based on the users' needs.

4. COMPUTATIONAL WORKFLOW

This section covers the specific implementation of BigJob (a Python script employing the Pilot-API) to address the workload described in Section 2. It begins with an overview, and then presents the approach as originally planned, followed by the actual implementation.

4.1 Overview

As described earlier, our workflow requires 105 chained sequences with each task in a chain being dependent upon successful completion and output of the previous task. The data (both inputs and outputs) for these systems are pre-organized in a simple hierarchy with a common base directory. The first tier represents the 5 chromosome sites, and the second tier represents the 21 locations (translations) along the DNA sequence representing the start of the nucleosome. All 20 chained simulations (1 ns each) for the same system (chromosome/location) share the same directory, using a file-naming convention that reflects the task sequence (`dyn0 - dyn20`), where files corresponding to the same dynamic step (task) share the same file prefix, and each file-type is represented by its own extension (`.coor, .vel, .xsc, .dcd, .dvd, .xst`). Some inputs, such as the force-field parameters (`sys.parm`) and atom connectivity information (`sys.pdb`), are common to all tasks within a chained sequence and needn't be duplicated for each step. Yet other inputs, such as the configuration (`.conf`) files, are the same for all systems, but differ from step to step. These configuration files are stored in a separate subdirectory off the base directory. The following is a brief schematic of the file organization.

```
|-dyn-conf-files      [NAMD configuration files common to all systems]
|---dyn1.conf         [NAMD configuration file for first dynamic step]
|---dyn2.conf
    :
|   dyn20.conf
|-chr02               [1st of 5 chromosome sites]
|---0068270           [1st of 21 nucleosome locations on 1st chromosome]
|-----sys.crd         [Parameter files common to all 20 dynamic steps]
|-----sys.parm
|-----min-eq.coor     [Input to 1st dynamic step]
|-----min-eq.vel
|-----min-eq.xsc
|-----dyn1.coor       [Output of 1st dynamic step, input to next]
|-----dyn1.vel
|-----dyn1.xsc
|-----dyn1.out        [Output-only files]
|-----dyn1.err
|-----dyn1.xst
|-----dyn1.dcd
|-----dyn1.dvd
            :
|-----dyn20.out       [Output of last dynamic step]
    :
|---0068290           [Last of 21 nucleosome locations on 1st chromosome]
  :
|-chr16               [Last of 5 chromosome sites]
```

- Input to step *X* (1-20):
  - `dynX.conf`: same for all systems, one for each nanosecond MD step, kept in a separate directory
  - `dynX-1.coor, dynX-1.vel, dynX-1.xsc`: coordinates, velocities and periodic cell data passed from previous step (7.2MB)
  - `sys.pdb, sys.crd, sys.parm`: topological and parameter data, same for all simulations on the same system (33.6MB)
- Output from step *X*:
  - `dynX.dcd, dynX.cvd, dynX.xst`: trajectory data in compressed binary format, used for further analysis (3.6GB)
  - `dynX.coor, dynX.vel, dynX.xsc`: passed to next step
  - `dynX.out, dynX.err`: *stdout, stderr*: primarily diagnostic information (167MB)

To help automate the workflow under BigJob (particularly for data staging with Pilot-Data), a naming convention had to be assumed for certain file-prefix keywords within the `dynX.conf` files, since these files can not in general be inspected during a Pilot-Job's run-time (unless they are local) to extract the filenames and orchestrate the file staging. So the assumption is that input file prefix within a `dynX.conf` file would be set to `dynX-1` and that the output file prefix would be set to `dynX`. The only exception was for the first step (`dyn1`), where the input file prefix is assumed to be `min-eq`.

4.2 Planned Approach

The initial plan was to use the Python-based BigJob API (Pilot-Job, Pilot-Compute, and Pilot-Data) to develop a BigJob script that would orchestrate an ensemble of tasks as batch pilot jobs on one or more XSEDE resources (like Lonestar and/or Kraken) using as many internal sub-queues to

distribute independent tasks in a round-robin fashion, while incorporating a shared workflow mechanism to help manage the dependent tasks, deferring their submission and staging files from output Data-Units to input Data-Units as necessary.

The workflow was to be defined in an external *Config* file, independent of the BigJob script, and read by the BigJob script. This workflow *Config* file would be generated semi-automatically based on the inspection of output files from prior BigJob runs.

The initial goal was to make use of the Pilot-Data API to stage data at run-time to and from a remote source, where the data originally resided and where further post-processing was to be done. In addition, an alternate approach where all files are accessible locally was also to be explored and evaluated against the remote staging approach.

4.3 Implementation

The actual implementation went mostly as planned, including the development of a BigJob script (`namd_bigwork.py`) in Python, using an `INI` style *Config* file to hold both BigJob configuration settings and the workflow definition, and a shell script (`gen_workflow`) was also written to generate the workflow section of the *Config* file. Each of these will be briefly described, starting with the format and generation of the workflow definition component.

4.3.1 *Workflow defined using* `INI` *style* Config *file*

Below is an example of a `WORKFLOW` section of the *Config* file. The `WORKFLOW` section contains subsections for the `REMOTE_DATA_RESOURCE`, `DEFAULTS`, and the `TASKS`. The `TASKS` section contains subsections for each task. Each task has a unique label, entries to help locate the various files, and a list of prerequisite tasks (`DEPENDENCIES`) identified by their labels. Most entries have default values if not specified. The *Config* entries are parsed and converted into a *Dictionary* object within the Python script using the `ConfigObj` library (favored over the built-in `ConfigParser` library).

```
[WORKFLOW]
   [[REMOTE_DATA_RESOURCE]]
     PROTOCOL = "ssh"
     USER = "jacks"
     MACHINE = "lonestar"
     DATACENTER = "tacc.utexas.edu"
     BASEPATH = "/scratch/02059/jsolow"
     CONF_DIR = "/scratch/02059/jsolow/dyn-conf-files"
     PARM_DIR = ""  # Default to DATAPATH
   [[DEFAULTS]]
   [[TASKS]]
     [[[chr04-1357521-4]]]
       DATAPATH = "chr04/1357521"
       CONF = "dyn4"
       PARM = ""  # Default to "SYS"
       IN = ""  # Default to "dyn3" (CONF-1)
       OUT = ""  # Default to "dyn4" (CONF)
       DEPENDENCIES = "chr04-1357521-3",
     [[[chr04-1357521-5]]]
       DATAPATH = "chr04/1357521"
       CONF = "dyn5"
       DEPENDENCIES = "chr04-1357521-4",
```

4.3.2 *Script to generate workflow* Config *file*

To help automate the generation of the workflow definition, a shell script (`gen_workflow`) was written to traverse the directory tree containing all the input and output files looking for *signatures* that a simulation was completed and skipping it as a task. For this particular study, these *signatures* included:

- `dynX.out` NAMD logfile contains "WallClock" in one of the last two lines upon successful completion
- `dynX.dcd` and `dynX.dvd` files are greater than 1.8GB (proportional to the size of the `dynX.coor` file and the number of dynamics steps)

The script uses the directory path (chromosome/location) and the dynamic step (1-20) to generate the task label. It assumes that the task is dependent upon on the task that precedes it, X-1, except for the first step, which assumes `min-eq.*` as its input. The base paths for the data tree, the NAMD configuration files, and parameter files are set in the `REMOTE_DATA_RESOURCE` section.

The script takes as arguments the range of dynamics steps to include in its traversal, with the default of 1-20. Restarting an incomplete run is usually just a matter of rerunning the `gen_workflow` script, which automatically omits the completed tasks.

The output of this script can be appended to the main *Config* file or kept as a separate file and referenced by the `FILE` keyword in the `WORKFLOW` section. The latter is the cleaner and preferred method.

4.3.3 *NAMD_BigWork Python script*

The primary deliverable of this exercise is a Python script (`namd_bigwork.py`) that uses the BigJob framework to orchestrate the submission of an ensemble of NAMD MD simulations (a collection of independent and dependent tasks) to one or more batch Pilot-Compute job instances with a significant reservation of resources for maximum high-throughput performance.

In this particular study, that reservation of resources is 2400 cores on Lonestar for a 24-hour period, sub-divided internally into 10 sub-queues, to handle 200+ simulations. Similar runs were done on Kraken, but it could only handle about 50 simulations in a 24-hour period. In principle, both Lonestar and Kraken could have been used together, but the data management issues could not be worked out (as discussed below).

The BigJob script sets up a Pilot-Job with both Pilot-Compute and Pilot-Data components, with the latter only used if the data resource is remote from the compute resource.

If the data is local to the computer resource, no data staging is necessary, and the Compute-Unit is directed to use the input data directory as the working directory, which is also the destination directory for the output and also where the next dynamic step expects its input data to be.

If the data is remote, then staging is needed to get the data to and from the Compute-Unit's temporary working directory. Three Data-Units are used to stage the data: an input Data-Unit, and output Data-Unit, and a chained Data-Unit. The chained Data-Unit is used to selectively redirect output data from one task to another dependent task as input. The Data-Unit-to-Data-Unit transfer is generally done by reference only (using symbolic links) and no data is actually moved, unless the Compute-Units are on different resources. However, with data staging, the input Data-Unit does need to copy data from the remote resource to the local input Data-Unit and then to the Compute-Unit's working directory, using SCP or SFTP (determined by the SAGA-BigJob framework). The output data is shuttled from the Compute-Unit's working directory back to the output Data-Unit, where it remains until the completion of the Pilot-Job. It then becomes the responsibility of the user, unfortunately, to rummage through the contents of the cryptic Data-Unit directories and move them to their final destination.

The introduction of workflow management to a BigJob script is the primary contribution from this effort [NAMD + *BigJob* + *Workflow* = NAMD_*BigWork*]. The task workflow is defined in the *Config* file (`namd_bigwork.conf` by default). The `WORKFLOW` section of the *Config* file is parsed and converted to a hierarchy of Python *Dictionary* structures for easy processing. The main construct is an array of `task` *Dictionary* elements that contain all the information about a particular task, including it's current status, which goes from `WAITING` to `PENDING` (if it has unsatisfied dependencies) to `New` to `Unknown` to `Running` to `Done` to `COMPLETED`, or `FAILED`, as it progresses along. The script continually loops through the tasks looking for changes in Compute-Unit and Data-Unit (if data staging) statuses and releasing `PENDING` tasks when their dependencies are satisfied. Any task that is in a WAITING (not PENDING) state is set up for submission to the Pilot-Job's agent. This set up involves the creation of the input, output, and chained Data-Units (if data staging) and a Compute-Unit. If data staging is being done, any requisite chained Data-Units are transferred to the input Data-Unit. Once all the Data-Units are processed, the Compute-Unit is submitted.

The Pilot-Compute component of a Pilot-Job is generally a batch job submitted to the system's batch queuing system (like PBS, SGE, or SLURM), and it can sit in the system queue for an indefinite period of time. Meanwhile, the Pilot-Job agent queues up Compute-Units for execution in its internal queuing system as sub-jobs. Each sub-job will have its own sub-allocation of cores from the total allocation given to the Pilot-Compute job. Once the batch Pilot-Compute job starts running, the backlog of sub-jobs will begin to get spawned as cores become available. In this particular study, there were 2400 cores distributed among 10 sub-jobs, 240 cores each. Once a sub-job is completed, those cores become available for another sub-job held in the Pilot-Compute's internal queue. When there are no remaining tasks (sub-jobs) to submit, the Pilot-Compute, Pilot-Data, and Pilot-Job agents are all gracefully shutdown and the batch job is terminated.

4.3.4 NAMD_BigWork log and status files

To help monitor, perform diagnostics, and do some post analytics (performance measures) on all the tasks, both a running log and a snapshot status file are generated during the run. The names of the log and status files are prefixed by the *Config* file name, followed by the a date and PID of the script, and suffixed with `.log` and `.status`, respectively.

The status file is written out with every state change showing details of each task organized by state. It contains the rather cryptic Compute-Unit and Data-Unit directory names to help map them to specific tasks. It also contains the current run time for each task since it entered the `Running)` state.

The running (accumulative) log file contains every task status change and the details of each submission, followed by a summary (tallied by state) with time-stamps. The overall level of detail for logging to *stdout* is set in a BigJob configuration file (`bigjob.conf`) or by an environment variable (`BIGJOB_VERBOSE`), but the written log file generally only contains `INFO` level output, although it can be overriden up to the overall logging level.

Filtering the log file (with *grep*, for example) can generate individual task summaries. Filtering for the state summaries provides a trace of the task throughput, which can be plotted against the time-stamps for a visual summary like that in Figure 2.

4.3.5 NAMD_BigWork *Config file*

The BigJob script is generally free of specifics about the compute and data resources, details of the NAMD executable, or of any workflow details. All that is externalized to the NAMD_BigWork *Config* file (`namd_bigwork.conf` by default). The *Config* file is broken into sections - `PILOT`, `NAMD`, and `WORKFLOW`. The `WORKFLOW` section was described earlier, and its content can be further externalized using the `FILE` keyword to reference a separate *Config* file for just the workflow details. An example *Config* file is given below.

```
[PILOT]
    COORDINATION_URL = "redis://localhost:6379"
    COMPUTE_USER = "jacks"
    COMPUTE_MACHINE = "lonestar"
    COMPUTE_DATACENTER = "tacc.utexas.edu"
    COMPUTE_SERVICE_PROTOCOL = "sge+ssh"
    COMPUTE_ACCOUNT = "TG-MCB100111"
    COMPUTE_WALLTIME = 10
    COMPUTE_QUEUE = "development"
    DATA_USER = "jacks"
    DATA_MACHINE = "lonestar"
    DATA_DATACENTER = "tacc.utexas.edu"
    DATA_SERVICE_PROTOCOL = "ssh"
    DATA_SIZE = 100
    SUBJOBS = 2
    PROCESSORS_PER_JOB = 48
    PROCESSORS_PER_NODE = 12
[NAMD]
    ROOT_DIR = "/home1/00288/tg455591/NAMD_2.9"
    BIN_DIR = "NAMD_2.9_Linux-x86_64-MVAPICH-Intel-Lonestar"
    EXEC = "namd2"
[WORKFLOW]
    FILE = "namd_workflow_jsoloe.conf"
```

4.4 Results

This section discusses the project outcomes, first looking at a typical run, then the study as a whole.

4.4.1 Typical run

A typical run for this study was a 24-hour 2400-core job on Lonestar broken into 10 240-core sub-job queues (within the Pilot-Job), which handled 210 MD simulations, representing 2 1-nanosecond simulations for each of 105 systems (21 locations along 5 chromosomes). Each simulation was 500,000 MD timesteps and took about 1 hour on 240 cores (on Lonestar). 2400 cores represents almost 10% of Lonestar's 28,800 cores.

Figure 2 shows a typical run profile for an ensemble of sub-jobs progressing through their states within the Pilot-Job. Note that BigJob tends to use a LIFO-like queuing mechanism, as the dependent tasks get interlaced early in the queue even though they were submitted much later (after their prerequisite tasks were completed).

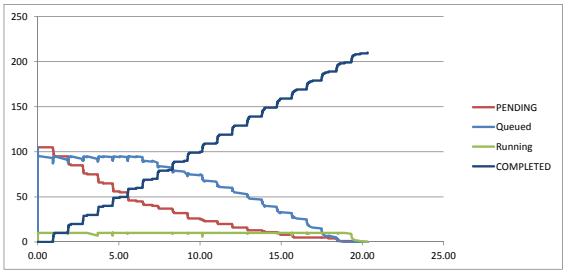

Figure 2: Progression of 210 sub-jobs (number in each state) over time (hrs)

Figure 3 shows the progression of the last seven BigJob runs over a ten day period. The typical BigJob completed all

210 tasks assigned to it. Exceptions are the first and last two BigJobs. The first BigJob was only assigned 185 tasks (the other 25 having been completed in an earlier run). The next to last BigJob completed only 209 sub-jobs. The failed sub-job was then automatically added to the last BigJob run. Careful inspection of the running simulations indicates that during the last hour of each BigJob run less than 10 sub-jobs might be running. For example, the number of running jobs for the next to last BigJob do not abruptly drop to zero at the end of day 69. It turns out that that sub-job failed to run due to an `mpiexec` problem and was never detected by the Pilot-Job, hence left to run out the clock - essentially waiting for a "dead" sub-job to complete.

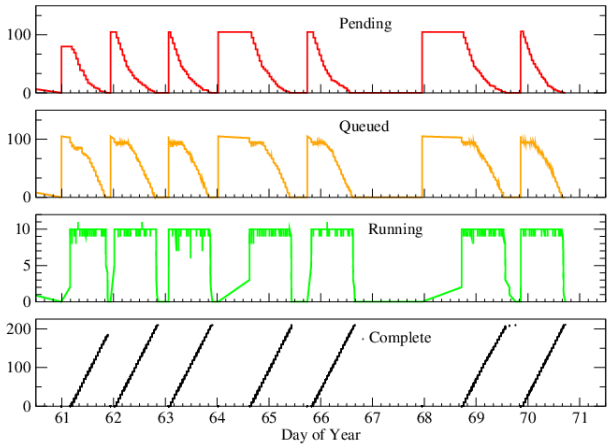

Figure 3: Progression of the last seven BigJob runs over a ten day period.

4.4.2 *Summary*

The total study took about 11 runs to cover the full 20 nanoseconds of simulation for each system, plus an initial minimization-equilibration step, for a total of 2310 simulations. At 240 cores for 1 hour each (on Lonestar), that's about 550,000 SUs, or 55,000 SUs per run (per day). On Kraken, this takes about 4 times longer and thus uses 4 times as many SUs.

There were a few runs that did not complete for various reasons (see *Issues* below) and had to be restarted using a regenerated workflow *Config* file.

Most simulations ran on Lonestar in just under an hour, but there were some outliers. Figure 4 shows the statistical distribution (histogram) of run-times (wall times) for all 2100 NAMD simulations. The shortest run-time was 54 minutes while 2027 tasks completed in under 1 hour. A total of eight simulations required longer than 2 hours to successfully complete. The longest single task (for a 1-nanosecond simulation) was 8 hours long. The mean run-time for all simulations was 57 minutes. If the simulations that required longer than 2 hours are excluded, the variation in run-times is less than 4 minutes or less than 6% of the expected run time. These results indicate that BigJob could proactively kill any sub-job taking longer than 1.5 hours (on Lonestar) and resubmit it, with the assumption being that the sub-job has encountered a hardware or system-level failure. Interestingly, when sub-jobs with anomalous run-times are manually rerun they usually complete within the expected time frame, indicating a hardware or software error rather than a fundamental problem with the simulated system.

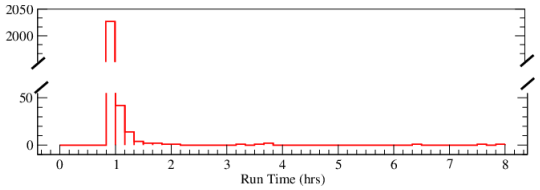

Figure 4: Histogram of run-times for the 2100 NAMD simulation tasks.

Similar results in variability were obtained on Kraken (with a different set of nucleosomes) but with about 4 times the effort.

5. DISCUSSION

This Section wraps up the study with a discussion about some of the challenges faced, remaining issues, some suggestions for future work, and some final conclusions.

5.1 Challenges

This experiment with BigJob and the integration of a workflow was not without its issues. The following highlight some of those issues.

One of the first challenges was to identify the right NAMD executable to use within the BigJob framework. NAMD is usually run via the `charmrun` frontend, which handles all the MPI details. Furthermore, the executable is usually built with the `Charm++` pre-processor/compiler/library, which interacts with the `charmrun` frontend. So it is not compatible with `mpirun`, which is used by BigJob. Fortunately, the NAMD developers provide a builder for an MVAPICH version of the NAMD executable, which appears to work appropriately under the BigJob environment.

The next big surprise was to discover that data staging only works one way, from the remote data resource to compute resource, and not the other way. Furthermore, Pilot-Data is not even supported on Kraken yet. Even where Pilot-Data is supported, as it is on Lonestar, getting data back to the remote data source is expected to be done external to BigJob. One can export to the local system (where the BigJob script is running), but unless that is the original data source, it is of limited value. The workflow in this study requires a good deal of data shuffling between dependent tasks, and the directory structure is non-trivial, so an alternate scheme needed to be explored, at least temporarily. The temporary solution (work-around) was to copy all the data to the compute resource (to scratch space on Lonestar in this case) and run everything local. As output was generated, it was `rsync`-ed back to the original source (in this case, a workstation at LA Tech) on a scheduled basis using `cron`. With the data and compute resources both being local, there is no need for data staging, and the working directory can be set to where the data resides - and run *in situ*. The volume of data in this study requires that scratch storage be used for the data, so long-term integrity of the data over the course of the study was a potential issue. It was important to keep all the data and its directory structure intact to help automate generation of the workflow *Config* file.

In an effort to work within the constraints of BigJob regarding output data staging (back to its remote origin),

another issue arose. It appears that SCP has issues with very large files. The trajectory files (`.dcd` and `.dvd`) for this study are around 1.8GB each, and it was discovered (after many SUs were consumed and scratch files purged) that they were getting truncated to about 500MB (without any warning or error messages) while they were being copied from the working directory to the Data-Unit store (or when being exported locally). So, any further data staging was abandoned until this issue could be resolved. This also precluded the ability to use multiple Pilot-Compute components concurrently on different compute resources (such as Lonestar and Kraken, as originally planned). However, it was demonstrated that selective Data-Unit-to-Data-Unit transfers (via a "chained" Data-Unit) can dramatically reduce the amount of data transfer (copying) that needs to take place for chained tasks. Sync-ing the data back to the original remote source is still a troubling issue, and the cryptic directory naming and flat directory structure used by BigJob don't help.

Going to an all-local solution avoids many issues, but it contradicts our goal of distributing jobs to multiple XSEDE machine. It also presents a few subtle issues of its own. For example, if the NAMD *Config* file is not in the working directory (which they are not in this study), one must force NAMD to ignore its location as the default location for the input files. This was done by adding a "`CWD getenv($PWD)`" entry to the NAMD *Config* file to force it to look in the current working directory.

Not only is exporting output data back to a remote resource the user's responsibility, but so is cleaning up afterward. The Compute-Unit's working directories and the Data-Unit directories can tend to grow rather quickly and need to be cleaned up by the user. One can sweep this issue under the rug by using scratch or temporary space, but it would be better citizenship to clean up manually if it can't be done automatically or programmatically within the BigJob script. Ideally, BigJob would support both remote exporting and the purging of working and staging directories.

Although most of BigJob's computational work is offloaded to one or more batch Pilot-Compute jobs, the launch script remains a key component in orchestrating the complex workflow, and often that script is launched on the head (login) node of the compute resource. If care isn't taken to throttle some of its activities, it becomes prone to be canceled by the system's administrator. This project experienced that first hand and put nearly 50,000 SUs at risk by having the batch Pilot-Job sit idle until the maximum wall time was exceeded. So care must be taken to throttle tight loops (using strategically placed `sleep()` calls) and minimize CPU usage if the script is being launched on a limited or controlled resource like a login node.

On the flip-side, if the batch Pilot-Compute job should somehow be aborted or terminated (e.g., after a series of `mpiexec` problems, which happened more than once during this study), the launch script can be left running unaware. In one instance a single sub-job got hung with some `mpiexec` problem and the Pilot-Job was unaware, causing the Pilot-Compute job to run with all cores idle until the wall time was exceeded. It would be helpful if the `get_state()` method for the Pilot-Job agents could detect such events or allow user-configured criteria to terminate the batch job (e.g., terminate if X% of the cores are idle for more than Y minutes).

BigJob uses an *Advert* service (usually a *Redis* server) to communicate and transfer information between various distributed components of the BigJob framework. The status of the *Advert* service is usually checked upon initiation of the BigJob environment; however, if the Avert service shuts down later, the batch Pilot-Compute job can abort and leave the BigJob launch script running unaware of the problem. Again, it would be helpful to be able to detect such events and shut everything down gracefully.

Much of the system-dependent details of working in a distributed heterogeneous environment are handled by using SAGA and the BigJob framework as middleware, and much of that is done using somewhat generic adapters; however, various subtle and system-specific details often remain, which need to be managed or configured by the user. This can defeat much of the transparency intended by SAGA and BigJob. For example, a user shouldn't have to know what particular batch queuing system a compute resource uses (Torque/PBS, SGE, SLUM,...) or whether to use SSH or GSISSH or fork. System-specific configuration files should be able to handle such nuances. He/she shouldn't have to completely overhaul his/her strategy based on whether the data is local to the compute resource. Data staging ought to be smart enough to figure that out. A global file system model/view or abstraction might be helpful - where everything looks local.

The MPI environment can vary between systems, especially when they're adapted to take advantage of certain architectures or interconnect fabric. The use of `aprun` on Kraken and `ibrun` on TACC machines in place of mpirun are good examples. The use of `charmrun` for NAMD is an even more extreme example. These are tough issues to deal with even for a veteran MPI user, let alone for someone trying to be shielded from such details and nuances. SAGA/BigJob needs to develop some generic wrapper mechanism and/or configuration file to mask this level of detail.

One final issue, which is more of a workflow management issue than a BigJob issue, is the potentially inefficient use of cores at the end of a run, when sub-queues aren't being kept full, leaving more and more cores idle as it approaches the end. Either the workflow manager (the BigJob script) needs to allocate more cores to tasks (sub-jobs) near the end of a run, or the Pilot-Compute job needs to be configurable to not accept any more sub-jobs if it can't utilize a certain portion of its available resources. This, of course, is very workflow dependent and probably shouldn't rely on the BigJob framework for anything but hints about resource utilization.

5.2 Future Work

This was just the first of a series (hopefully) of experiments with BigJob and distributing large ensembles of chained NAMD simulations across a heterogeneous mix of compute and data resources like those on XSEDE. The following are a few things suggested for future work.

The INI-style *Config* file format is rather limited in its ability to express scientific workflow. Workflow management theories often use directed acyclic graphs (DAG) to express workflow, where nodes are tasks and edges represent dependencies and/or data flow. There are now markup languages to express DAGs, such as DAX (DAG in XML), which can be used by Pegasus [9] and potentially other workflow management systems. Other graph-based workflow management systems (often based on web services) include: Taverna [22],

Galaxy [10], and Kepler [5]. There are even graphical programs like Triana (http://www.trianacode.org) that generate DAX. Adopting an extensible language/format like DAX would help broaden the scope of workflow management and allow some degree of plug-and-play with other workflow management tools.

The original goal was to demonstrate a more distributed example of using the BigJob framework, with multiple Pilot-Compute jobs running on different compute resources, data being staged from yet another resource (and back), and all controlled by a single workflow management front-end. Although each aspect was demonstrated independently, with a few caveats, the chore of tying it all together remains.

One thing that was not fully explored was the interlacing of data staging with compute processing to avoid blocking while large data files are being moved around. This will require a better understanding of the states and their transitions within the Pilot-Data framework to help eliminate the blocking `wait()` calls.

The workflow management needs to better address the "end game" by re-allocating cores (reducing the number of sub-queues and redistributing allotted cores) near the end of a run to avoid idling resources as the number of tasks (sub-jobs) drops below the number of available sub-queues. Not all tasks scale well, however, so this is a non-trivial matter. More is not always better, and being busy isn't the same as being productive - even for computers.

Many of the issues faced in this exercise were due to lack of monitor and control points among the various components involved. Exposing more of the underlying Pilot-API could help trap and respond to some of these events inside a BigJob script. However, exposing more of the Pilot-API (and the some of the other APIs, like to the Advert service) to the command language interface (CLI) could provide a separate external layer of monitoring and control that would help dynamically manage sub-jobs and tweak the workflow as needed. In that regard, building a GUI client for the CLI could be a valuable tool, especially if it's portable across platforms and robust to connection loss and restoration - a REST-based interface, perhaps, that runs in a web browser. These are obviously outside the scope of the end-user of BigJob, but some food for thought for the BigJob development team.

5.3 Conclusions

To date, less than ten molecular dynamics studies of the nucleosome have been reported [6]. All except a control study performed by this group [17, 18] used nominally the same sequence of DNA. The longest all-atom trajectory reported for a nucleosome is 500ns. By using BigJob to marshall resources distributed across XSEDE we have achieved a sampling of nucleosome sequence variability and dynamics that is over two orders of magnitude higher than any previous effort. We note that the "experiments" reported here were actually production runs for our nucleosome study that successfully completed. Successful completion alone supports what is known biologically: that any sequence of DNA can be wrapped around a histone core to form a nucleosome, albeit with varying affinities. Achieving the level of throughput reported here is a necessary first step in dissecting DNA sequence specific properties of nucleosomes in atomic detail which, because of the fundamental role of nucleosomes, impacts our understanding of virtually all genetic processes.

Acknowledgement

Important funding for SAGA has been provided by NSF-ExTENCI (OCI-1007115). Additional support for Bishop provided through LA-SiGMA by NSF EPSCoR Cooperative Agreement No. EPS-1003897 and the Louisiana Board of Regents. Support for Smith was provided by XSEDE (OCI-1053575) under the Campus Champions Fellowship Program. This work has also been made possible thanks to computer resources provided by several XSEDE awards: TG-MCB090174, TG-MCB100111 and TG-CHE120039. We are grateful to Tommy Minyard (TACC) for help with debugging and performance tuning of BigJob on Lonestar, Ranger and Stampede and to James Solow (LA Tech) for setting up some of the simulations.

6. REFERENCES

[1] SAGA Project at XSEDE, `http://xsede.org/bigjob`.

[2] Redis, `http://redis.io`.

[3] Setting up a Redis Server, `http://saga-project.github.io/BigJob/sphinxdoc/install/redis.html`.

[4] SAGA-BigJob. https://github.com/saga-project/BigJob/wiki.

[5] I. Altintas, C. Berkley, E. Jaeger, M. Jones, B. Ludascher, and M. S. Kepler. An extensible system for design and execution of scientific workflow. *16th Intl. Conf. on Scientific and Statistical Database Management*, 11(8):R86, June 2004.

[6] M. Biswas, J. Langowski, and T. C. Bishop. Atomistic simulations of nucleosomes. *Wiley Interdisciplinary Reviews: Computational Molecular Science*, 2013.

[7] R. Buning and J. Van Noort. Single-pair FRET experiments on nucleosome conformational dynamics. *Biochimie*, 92(12):1729–1740, 2010.

[8] D. Case, T. C. III, T. Ddarden, H. Gohlke, R. Luo, K. M. Jr., A. Onufriev, C. Simmerling, B. Wang, and R. Woods. The Amber Biomolecular Simulation Programs. *Journal of Computational Chemistry*, 26(16):1668–1688, 2005.

[9] E. Deelman, G. Singh, M.-H. Su, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, G. Berriman, J. Good, A. Laity, J. Jacob, and D. Katz. Pegasus: A Framework for Mapping Complex Scientific Workflows onto Distributed Systems. *Sci. Program.*, 13(3):219–237, 2005.

[10] J. Goecks, A. Nekrutenko, J. Taylor, and T. G. Team. Galaxy: a comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences. *Genome Biol.*, 11(8):R86, August 2010.

[11] C. Jiang and B. F. Pugh. A compiled and systematic reference map of nucleosome positions across the saccharomyces cerevisiae genome. *Genome Biol*, 10(10):R109, 2009.

[12] 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*, pages 135–144, 2010.

[13] A. Luckow, P. Mantha, M. Romanus, and S. Jha. Pilot-Abstractions for Data-Intensive Cloud Applications. 2012. Computing and Concurrency: Practise and Experience (in press).

[14] A. Luckow, M. Santcroos, A. Merzky, O. Weidner, P. Mantha, and S. Jha. P *: A model of pilot-abstractions. In *E-Science (e-Science), 2012 IEEE 8th International Conference on*, pages 1–10, Oct.

[15] A. Luckow, M. Santcroos, O. Weidner, A. Merzky, S. Maddineni, and S. Jha. Towards a common model for pilot-jobs. In *Proceedings of the 21st International Symposium on High-Performance Parallel and Distributed Computing,HPDC'12*. ACM, June 2012. `http://dl.acm.org/citation.cfm?id=2287094`.

[16] K. Luger, M. L. Dechassa, and D. J. Tremethick. New insights into nucleosome and chromatin structure: an ordered state or a disordered affair? *Nat Rev Mol Cell Biol*, 13(7):436–447, Jul 2012.

[17] R. Mukherjee and T. C. Bishop. *Nucleosomal DNA: Kinked, Not Kinked, or Self-Healing Material?*, chapter 6, pages 69–92. American Chemical Society, 2011.

[18] R. Mukherjee, A. Thota, H. Fujioka, T. C. Bishop, and S. Jha. Running many molecular dynamics simulations on many supercomputers. In *Proceedings of the 1st Conference of the Extreme Science and Engineering Discovery Environment: Bridging from the eXtreme to the campus and beyond*, XSEDE '12, pages 2:1–2:9, New York, NY, USA, 2012. ACM.

[19] U. M. Muthurajan, Y.-J. Park, R. S. Edayathumangalam, R. K. Suto, S. Chakravarthy, P. N. Dyer, and K. Luger. Structure and dynamics of nucleosomal dna. *Biopolymers*, 68(4):547–556, Apr 2003.

[20] J. C. Phillips, R. Braun, W. Wang, J. Gumbart, E. Tajkhorshid, E. Villa, C. Chipot, R. D. Skeel, L. Kale', and K. Schulten. Scalable molecular dynamics with namd. *J Comput Chem*, 26(16):1781–1802, Dec 2005.

[21] R. C. Stolz and T. C. Bishop. ICM Web: The interactive chromatin modeling web server. *Nucleic Acids Research*, 38(2):W254–W261, 2010.

[22] K. Wolstencroft, R. Haines, D. Fellows, A. Williams, D. Withers, S. Owen, S. Soiland-Reyes, I. Dunlop, A. Nenadic, P. Fisher, J. Bhagat, K. Belhajjame, F. Bacall, A. Hardisty, A. N. de la Hidalga, M. P. B. Vargas, S. Sufi, , and C. Goble. The taverna workflow suite: designing and executing workflows of web services on the desktop, web or in the cloud. *Nucleic Acids Research*, 2013.

[23] J. Zlatanova, T. C. Bishop, J.-M. Victor, V. Jackson, and K. van Holde. The nucleosome family: dynamic and growing. *Structure*, 17(2):160–171, Feb 2009.