# The Anatomy of Successful ECSS Projects: Lessons of Supporting High-Throughput High-Performance Ensembles on XSEDE

Melissa Romanus
CAC/ECE
Rutgers University
94 Brett Road
Piscataway, NJ
melissa@cac.rutgers.edu

Pradeep Kumar Mantha
Center for Computation and Technology
Louisiana State University
216 Johnston
Baton Rouge, LA
pmanth2@cct.lsu.edu

Matt McKenzie
National Institute for Computational Sciences
Oak Ridge, TN
mmckenz9@utk.edu

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

Emilio Gallichio
BioMaPS Institute
Rutgers University
Piscataway, NJ
emilio@biomaps.rutgers.edu

Andre Merzky
Center for Computation and Technology
Louisiana State University
216 Johnston
Baton Rouge, LA
andre@merzky.net

Yaakoub El Khamra
Texas Advanced Computing Center
The University of Texas at Austin
Austin TX
yaakoub@tacc.utexas.edu

Shantenu Jha
CAC/ECE
Rutgers University
94 Brett Road
Piscataway, NJ
shantenu.jha@rutgers.edu

## ABSTRACT

The Extended Collaborative Support Service (ECSS) of XSEDE is a program to provide support for advanced user requirements that cannot and should not be supported via a regular ticketing system. Recently, two ECSS projects have been awarded by XSEDE management to support the high-throughput of high-performance (HTHP) molecular dynamics (MD) simulations; both of these ECSS projects use a SAGA-based Pilot-Jobs approach as the technology required to support the HTHP scenarios. Representative of the underlying ECSS philosophy, these projects were envisioned as three-way collaborations between the application stakeholders, advanced/research software development team, and the resource providers. In this paper, we describe the aims and objectives of these ECSS projects, how the deliverables have been met, and some preliminary results obtained. We believe the structure of the ECSS program enables targeted projects that address missing gaps in making distributed cyberinfrastructure systems more productive. We also describe how SAGA has been deployed on XSEDE in Community Software Area as a necessary precursor for these projects.


*XSEDE12,* July 16 - 20 2012, Chicago, Illinois, USA


## Categories and Subject Descriptors

D.1.3 [**Software**]: Concurrent Programming Distributed programming/parallel programming; J.3 [**Computer Applications**]: Computational Biology

## General Terms

Design,Measurement,Theory

## Keywords

Distributed Computing, High Performance, High Throughput, SAGA, ECSS Projects

## 1. INTRODUCTION

The Extended Collaborative Support Service (ECSS) pairs members of the XSEDE user community with expert staff for an extended period to work together to solve challenging science and engineering problems through the application of cyberinfrastructure ([6]). In depth staff support, lasting a few weeks up to a year in length, can be requested at any time through the XSEDE allocations process. Expertise is available in a wide range of areas, from performance analysis and petascale optimization to the development of community gateways and work and data flow systems.

For example, although individual Molecular Dynamics (MD) simulations are very common on XSEDE, it is not very common to execute multiple MD simulations concurrently, and even far less multiple MD simulations on multiple MD resources concurrently. Although the abilty to execute an ensemble of MD simulations is sought by many MD users, the abilty to do so in a simple, scalable and general-purpose fashion does not exists. The ECSS provides an excellent program to build upon base existing capabilties and transform them into a value-added service.

In this paper, we discuss two ECSS projects that overlap in both scientific endeavor and computational challenges. The success of these two projects is attributed to the strong interaction and familiarity between ECSS staff and infrastructure developers. The collaboration between system administrators and middleware developers has led to the deployment of production-grade infrastructure that is currently being used by scientists originally intended as beneficiaries of ECSS as well as other scientists.

The computational challenges were circumvented through innovative solutions in distributed and high throughput computing, thus satisfying the computational scientists' requirements. The development and deployment of these tools is left to computer scientists, developers, and programmers of infrastructure and middleware. The ECSS staff supports both scientists and tool developers in deployment, bug reporting, testing, fixes and system issues. This three-way split insulates the scientists from the "plumbing" and the ECSS consultants from middleware development. The successful distribution of labor translates to reduced effort across the board, higher scientific output, and less issues. Furthermore, the fact that middleware developers are in constant communication with ECSS consultants improves the middleware itself. The ECSS consultants report to the developers' issues ranging from middleware misbehavior, to choke points, scaling issues, unnecessary increases in file system load, and so on. The developers resolve these issues, test, and re-deploy while maintaining backwards compatibility, leaving the scientists unaffected.

The overlap of two ECSS projects with the same computational challenges lead to increased efficiency. The same tools, infrastructure, and documentation prepared for one ECSS project are being used by another, similar project. There was, however, a slight increase in the number of issues reported: twice the number of users meant more bugs reported. This is only expected, and to some extent quite welcome. At the end of the day, however, two ECSS projects are nearing completion ahead of schedule for the cost of just one project.

In this paper, we discuss the ECSS projects, the computational challenges involved and the middleware solution. The progress made by the end users is discussed in section 6 along with the summary of simulations run thus far. We also discuss in detail the lessons learned from testing and deploying on different machines and collaborating across the three disciplines of scientists, middleware developers and ECSS consultants.

# 2. SCIENTIFIC BACKGROUND AND MOTIVATION

In this section, we provide the scientific background of both ECSS projects and the computational challenges inherent in the two projects. We establish that there are common computational challenges and thus *coupling* ECSS projects with similar computational challenges with the same ECSS consultant and middleware developer team is an efficient use of XSEDE resources.

## 2.1 Distributed and Loosely Coupled Parallel Molecular Simulations

The first ECSS project ([14]) is part of an intense effort to understand important aspects of the physics of protein-ligand recognition by multidimensional replica exchange (RE) computer simulations. These are compute intensive calculations which require large numbers ($10^3$-$10^4$) of loosely coupled replicas and long simulation times (days to weeks). In conventional synchronous implementations of RE, simulations progress in unison and exchanges occur in a synchronous manner whereby all replicas must reach a predetermined state (typically the completion of a certain number of MD steps), before exchanges are performed. This synchronous approach has several severe limitations in terms of scalability and control. The first such limitation is that sufficient dedicated computational resources must be secured for all of the replicas before the simulation can begin execution. Secondly, the computational resources must be statically maintained until the simulation is completed. Thirdly, failure of any replica simulation typically causes the whole calculation to abort. These traditional limitations are overcome in the asynchronous RE formulations ([8]). The scientific aim is to study a range of physical systems using these advanced formulations and and to utilize the distributed high-performance capacity of XSEDE resources.

## 2.2 High Throughput, High Performance MD Studies of the Nucleosome

The second ECSS project focuses on high throughput, high performance molecular dynamics studies of the nucleosome ([3]). Genomes in higher organisms exist for most of the cell's cycle as a protein-DNA complex called chromatin. Nucleosomes are the building blocks of chromatin, thus, nucleosome stability and their positioning within chromatin impacts virtually all genetic processes including: transcription, replication, regulation, repair.

The primary goal of the proposed simulation studies in this project is to investigate by means of all atom molecular dynamics simulations variations in nucleosome structure and dynamics arising from DNA sequence, chemical modi cations and receptor binding. This is being accomplished by means of a hierarchical modeling and simulation strategy that utilizes high throughput, high performance all-atom molecular dynamics simulation techniques. To date the project has accumulated over 7 $\mu s$ (microseconds) of simulation for 350 different realizations of the nucleosome. It is on track to generate another 9 $\mu s$ for 85 additional models of the nucleosome.

## 2.3 Computational Challenges

Most runtime environments and tools for high-performance and distributed computing assume either that there is a single instance of a task/simulation that must be executed, or that there is a complex dependency (and thus workflow) associated with distinct "tasks". However, in an increasingly large number of scientific problems, there is a need for a large number of similar tasks to be run concurrently.

There is a fundamental need for supporting ensemble-based simulations (ES) across a range of disciplines, including but not limited to biological, chemical, environmental and material sciences; existing capabilities are insufficient to support ES at scale.

Depending upon the specific problem, the identical tasks may vary in the number of cores required, or in the degree of coupling between the tasks. Either way, there are missing capabilities to support the multiple similar tasks that need to run concurrently, and the management of such concurrent tasks is a challenge for ES.

Typically, for a given science problem, the number of tasks does not vary during the lifetime of the execution. The objective of ES is similar to traditional workflows in that it aims for a reduction of the makespan. However, in contrast to such workflows, there is not a multi-level dependency to solve and thus the execution of ES is not fundamentally a challenging scheduling problem.

The concurrent support of multiple similar simulations requires the efficient and transparent management of a large number of ensembles, possibly on many heterogeneous resources. The same runtime solution should be applicable to a range of physical model sizes – from thousands of atoms to hundreds of thousands of atoms. The above should be addressed without being tied to a specific underlying application kernel or physical infrastructure.

Although multi-site resources have the potential for substantial improvements in time-to-completion of ES, the capability to effectively utilize NSF's National Production Cyberinfrastructure XSEDE is missing, as developing, deploying and executing these capabilities to use the collective power of XSEDE is a challenging undertaking, which cannot be carried forth by a single group or sustained without the involvement of all stakeholders: scientists, resource providers and software developers.

# 3. ECSS PROJECTS

The two ECSS projects were aligned in infrastructure, tools and workflow. For example, the projects were synergistic in that the requirements of one were often also requirements of the other. This meant that the deployment of the infrastructure (SAGA and BigJob), the development of the actual workflows, and data management tools were requirements for both ECSS projects. This section outlines the level of effort involved in both projects.

## 3.1 Project 1

### *Distributed and Loosely Coupled Parallel Molecular Simulations*

The reliance on a static pool of computational resources and lack of fault tolerance prevents the synchronous RE approach from being a solution on XSEDE resources. We therefore implemented asynchronous parallel replica exchange conformational sampling algorithms together using the SAGA distributed computing framework to enable dynamic scheduling of resources and adaptive control of replica parameters. The approach allows pairs of replicas to contact each other to initiate and perform exchanges independently from the other replicas. Because replicas do not rely on centralized synchronization steps, asynchronous exchange algorithms are scalable to an arbitrary number of processors. These algorithms also circumvent the need to maintain a static pool of processors and therefore can be distributed logically and physically across XSEDE resources. In that case, the number of concurrent replicas changes dynamically depending on available resources. This mode of execution is particularly suitable for implementation using SAGA and tools based upon SAGA (i.e. BigJob). The technical details of both SAGA and BigJob are discussed in Sections 4.1 and 4.2.

In terms of scope, assistance from XSEDE personnel was needed to: (i) set up and harden the necessary SAGA, BigJob and Advert Service software infrastructures on Ranger, Lonestar, Kraken, and Nautilus and deploy the scientific software IMPACT ([1]) and AMBER ([4]) (ii) work with the users to enable launching NAMD ([18]), IMPACT, and AMBER distributed jobs using the SAGA-BigJob framework, (iii) work with users to develop customized RE scripts and with the SAGA team to test and validate relevant adaptors aimed at conducting synchronous and asynchronous file-based replica exchange simulations in the context of the SAGA-based Pilot-Job framework, and (iv) document and validate the usage of the Replica-Exchange framework on XSEDE.

Hardening the software infrastructures requires profiling and benchmarking the workflow management tools (SAGA, BigJob) to measure overhead and identify any load spikes on the filesystems. Once the bottlenecks and problem areas were identified, they were resolved by the SAGA-BigJob development teams with help from the ECSS consultants. The SAGA-BigJob development team handled all major feature implementations and bug fixes required for this effort including custom adaptors for Lonestar and Ranger. The level of commitment from the SAGA team was no less than two full time researchers and two graduate students working full time for a little over four months. The level of commitment from XSEDE was one FTE (50% of two consultants) working for the same duration. The high level of commitment ensured the project went off to a flying start and is wrapping up earlier than planned.

## 3.2 Project 2

### *High Throughput, High Performance MD Studies of the Nucleosome*

The workflow for the second ECSS project requires simulating as many as 336 independent systems simultaneously. For each system, every nanosecond of simulation is an HPC event that requires approximately 50 MB of input, generates 4 GB of output, and scales efficiently from as few as 32 to as many as 512 cores. On 64 processors, the runtime for a single 1 ns simulation task is approximately 6 hrs. The goal is to accumulate 20 ns of simulation for the entire set of 336 systems (6,720 1 ns simulation tasks in total) as quickly as possible, then select a subset of systems from the ensemble for which 500 ns of additional dynamics will be accumulated. If we chose only 10 systems for continued studies, there would still be 5,000 1 ns simulation tasks to be completed. To reduce the time-to-completion of the entire workload and to manage simulation inputs/outputs on scratch file systems, we intend to use multiple XSEDE resources simultaneously. This approach requires data staging.

In contrast to Project 1, this project requires: (i) a much larger number of ensembles running concurrently and for longer durations, (ii) the chaining of ensembles, (iii) much

larger data-volumes, (iv) BigJob infrastructure that currently does not support the co-movement and coordinated movement of data (files) in conjunction with ensemble placement. BigJob file movement capabilities are being enhanced to provide such support.

Support from ECSS consultants is necessary for disk and storage bottlenecks. Data management (especially varying transfer rates, less-than-portable transfer mechanisms, etc.) is a growing issue that must be addressed in order for this project to be successful.

### 3.3 Common Effort across Both Projects

Both projects share the same basic infrastructure in terms of software and hardware. Both projects share consultant teams, software developers and some of the scientific tools. The workflows are not dissimilar either: both ECSS projects intend to launch pilot jobs in a dynamic workflow on distributed resources, stage and collect data, etc.

With a view towards making these capabilities more publicly accessible and widely used, this ECSS effort also includes the development of a prototype Replica-Exchange Gateway. This gateway was developed using the DARE ([5]) scientific gateway framework. This scientific gateway prototype is hosted on IU's Data Quarry machine ([10]). Furthermore, all tools, utilities, and components resulting from this ECSS will be placed in the Community Software Area (CSA) space on Ranger, Lonestar, and Kraken. All code and corresponding documentation will also be maintained in a local revision control software repository which is publicly available.

## 4. METHODICAL SOLUTION AND TECHNOLOGY

Although the objectives of the two ECSS projects are distinct, there is certain amount of commonality in the technology employed. Both projects use SAGA and SAGA-based Pilot-Jobs to support the concurrent execution of multiple simulations (ensembles or replicas as the case maybe). In this section we briefly introduce the technology employed.

### 4.1 SAGA: Standards-based Access to the Resources Layer

XSEDE is inherently a very complex infrastructure. Given the wide range of user groups and application use cases it aims to address, and the large number and the diversity of participating resource providers, this is to be expected. Complex systems are though usually very difficult to use, as that complexity and the underlying resource diversity often translates into complicated user tools and interfaces. The relatively clean architecture of XSEDE is, to some extent, addressing this problem, but is, in itself and at this point in time, a moving target.

The Simple API for Grid Applications (SAGA) ([20]) aims to address a part of that problem, by providing a well defined and stable API, which exposes those operations which are required on application level, but encapsulates the complexity of translating them into the respective operations on the XSEDE infrastructure. In other words: SAGA tries to move the complexity of dealing with distributed cyberinfrastructures like XSEDE out of the application, and into the SAGA implementation layer, while providing the semantics necessary to efficiently implement distributed applications which can utilize XSEDE.

SAGA is an implementation of an Open Grid Forum (OGF) Technical Specification that provides a common and consistent high-level API for the most commonly required functionality to construct distributed applications. It also provides a high-level API to construct tools and frameworks to support distributed applications. The functional areas that are supported by SAGA include job-submission, file transfer and access, as well as support for data streaming and distributed coordination. SAGA provides both a syntactic and semantic unification via a single interface to access multiple, semantically distinct middleware distributions.

The key advantages of the development using SAGA include, but are not limited to: i) to provide a general-purpose, commonly used yet standardized functionality, while hiding complexity of heterogeneity of back-end resources, ii) to provide building blocks for constructing higher-level functionality and abstractions, iii) to provide the means for developing broad range of distributed applications such as gateways, workflows, application management systems, and runtime environments. Interestingly, SAGA provides an integrated, light-weight approach to support scripting for building distributed applications.

Different aspects of SAGA appeal to different groups. The standardization of SAGA as an OGF Standard is important because it makes it more likely that production infrastructures, like NSF XSEDE, EU PRACE and Open Science Grid, will support SAGA. Having SAGA deployed and tested on these systems makes it readily available for users and developers of national Cyberinfrastructure projects. The fact that SAGA is an OGF technical specification also makes SAGA highly appealing to application frameworks, services and tool developers, which is quite understandable, as it not only simplifies their development but also makes for scalable, extensible and sustainable development. Users find the simple and extensible interface providing the basic abstractions required for distributed computing is very appealing to add their own "functionality" to a core base of functionality. Furthermore, SAGA is now part of the "official" access layer for the $121M NSF TG/XSEDE project ([22]), as well as for the world largest distributed infrastructure EGI ([7]).

The SAGA API provides the base abstractions upon which tools and frameworks that provide higher-level functionality can be implemented. Ref. ([20]) discusses distributed application frameworks and run-time systems that SAGA has been used to develop successfully.

A SAGA implementation consists of a high-level API, the SAGA Engine providing that API, and backend, middleware/systems specific adaptors. Each of these adaptors implements the functionality of a functional package (e.g., job adaptors, file adaptors) for a specific middleware system. The engine is a dynamic library that manages call dispatching and the runtime loading of the middleware adaptors. Adaptors are also realized as dynamic libraries. The SAGA API has been used (in C++, Python and Java versions) to provide almost complete coverage over nearly all grid and distributed computing middleware/systems, including but not limited to Condor, Genesis, Globus, UNICORE, LSF/PBS/Torque and Amazon EC2.

SAGA is currently used on production Cyberinfrastructure in several ways. Admittedly the number of first-principle distributed applications developed is currently low, but SAGA has been used to develop "glue-code" and tools that are used

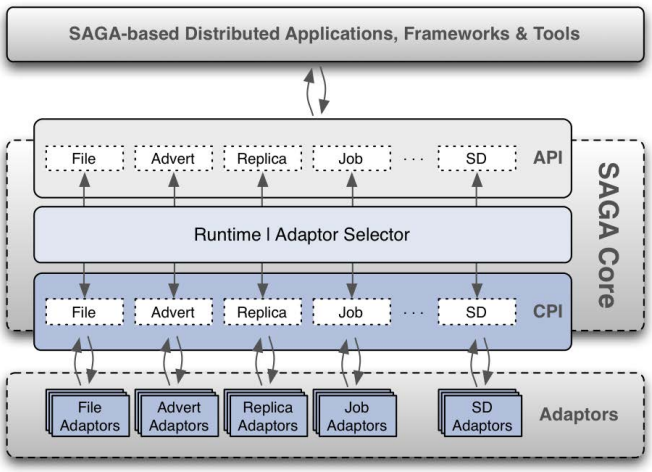


**Figure 1: SAGA Overview: SAGA is an OGF technical specification that provides a common interface to heterogeneous DCI – hitherto typically Grid systems. The implementation of the SAGA ([20]) specification consists of a high-level *API*, the SAGA *Engine* providing that API, and backend, system-specific *Adaptors*. The engine is a lightweight, highly-configurable dynamic library that manages call dispatching and the dynamic runtime loading of the middleware adaptors. Each of these adaptors implements the functionality of a specific functional package (e.g., job adaptors, file adaptors) for a specific middleware system. Adaptors are also realized as dynamic libraries.**

to submit and marshal jobs and data across and between heterogeneous resources. Specifically, it has been used to support multiple computational biology applications that use high-performance and high-throughput molecular dynamics (MD) simulations.

SAGA has been used to develop a standards-based library for Science Gateways to easily utilize different distributed resources; some science domains that are using SAGA-based Science Gateways include gateways to support next-generation sequencing, docking and high-throughput of ensembles.

## 4.2 SAGA-based Pilot-Job: BigJob

A common approach to circumvent the computational challenges faced by our ECSS projects is the use of container jobs with sophisticated workflow management to coordinate the launch and interaction of actual computational tasks within the container. This is a particular instance of a more general concept: *pilot-jobs (PJ)*. A pilot-job is a mechanism by which a proxy for the actual simulations is submitted on the resource to be utilized; this proxy agent in turn, is given responsibility to convey to the application the availabilty of resources and also influence 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.The SAGA-based pilot-job is a container job which can include within its fold multiple smaller tasks.

The pilot-job (PJ) abstraction is also a promising route to address additional requirements of distributed scientific applications ([13, 15]), such as application-level scheduling. 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.

A SAGA-based PilotJob, BigJob (BJ) ([2, 17]), is a general-purpose pilot-job framework. BigJob has been used to support various execution patterns and execution workflows ([16, 21]). For example, SAGA-BigJob was used to execute scientific applications categorized as embarrassingly parallel applications and loosely coupled applications on scalable distributed resources ([11, 12])

Figure 2 illustrates the architecture of BJ. BJ utilizes a Master-Worker coordination model. The BigJob-Manager is responsible for the orchestration of pilots and for the binding of sub-tasks. For submission of the pilots, SAGA relies on the SAGA Job API, and thus can be used in conjunction with different SAGA adaptors, e. g. the Globus, the PBS, the Condor and the Amazon Web Service adaptor. Each pilot initializes a so called BJ-agent. The agent is responsible for gathering local information and for executing tasks on its local resource. The SAGA Advert Service API is used for communication between manager and agent. The Advert Service (AS) exposes a shared data space that can be accessed by manager and agent, which use the AS to realize a push/pull communication pattern. The manager pushes a sub-job to the AS while the agents periodically pull for new sub-jobs. Results and state updates are similarly pushed back from the agent to the manager. Furthermore, BJ provides a pluggable communication and coordination layer and also supports alternative communication and coordination systems, e. g. Redis ([19]) and ZeroMQ ([23]).

In many scenarios it is beneficial to utilize multiple resources, e. g. to accelerate the time-to-completion or to provide resilience to resource failures and/or unexpected delays. BJ supports a wide range of application types, and is usable over a broad range of infrastructures, i. e. it is general-purpose and extensible (Figure 2).

High Performance Distributed Computing (HPDC) infrastructure is by definition comprised of a set of resources that is fluctuating – growing, shrinking, changing in load and capability. This is in contrast to a static resource utilization model traditionally a characteristic of parallel and cluster computing. The ability to utilize a dynamic resource pool is thus an important attribute of any application that needs to utilize HPDC infrastructure efficiently. Applications that support dynamic execution have the ability to respond to a fluctuating resource pool, i. e., the set of resources utilized at time (T), $T = 0$ is not the same as $T > 0$. Thus, the need to support dynamic execution is widespread for computational science applications; here we accomplish this by using the SAGA-based Pilot-Job.

It is worth mentioning that there are alternative "launcher" modules available on Ranger and Lonestar that share some of the functionality of BigJob. These modules are single machine only (i.e. no distributed computing). In addition, they are installed on a small subset of XSEDE (only Ranger and Lonestar, not Kraken or Trestles) resources and do not support workflows or mulitple container jobs. Furthermore, the "Launcher" module is only capable of launching single pro-

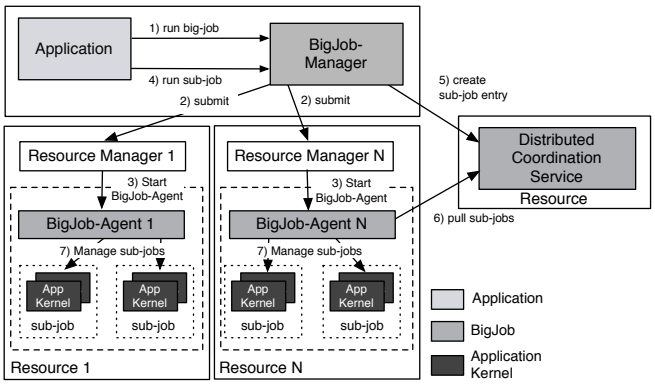


**Figure 2: BigJob Architecture: The core of the framework, the BigJob-Manager orchestrates a set of pilots. Pilots are started using the SAGA Job API. The application submits WUs, the so-called sub-jobs via the BigJob-Manager. Communication between the BJ-Manager and BJ-Agent is done via a shared data space, the Advert Service. The BJ-Agent is responsible for managing and monitoring sub-jobs. From Ref. ([17])**

cessor jobs. While this limitation has been removed with the "pyLauncher" module, all other limitations persist.

# 5. DEEP INTEGRATION OF SAGA/BIGJOB ON XSEDE

## 5.1 CSA Deployment of SAGA and BigJob

SAGA and BigJob are deployed on all major XSEDE machines (Table 1) (Ranger, Kraken, Lonestar, Trestles, Blacklight, and Steele) as well as FutureGrid machines (India, Sierra, Hotel and Alamo). While it is possible to install and use SAGA and BigJob in each user's directory, it would not be without a non-trivial amount of effort in configuration and maintenance. Furthermore, having a central location on each machine where SAGA is deployed as a community code saves ECSS consultants time and effort in maintaining the installation and supporting its users. XSEDE continues the TeraGrid model of providing Community Software Areas (CSA) to communities, which basically is a system level area where software packages can be shared among a community of users. Therefore it was only logical to deploy SAGA and BigJob in a CSA space on XSEDE and FutureGrid machines.

The installation and update are semi-automatic: a set of deployment scripts are used to manually trigger the update. The update can encompass an individual machine or all machines, an individual SAGA component or the entire SAGA/BigJob installation. The set of supported components includes the SAGA core libraries, different API packages, the supported middleware adaptors for each machine, the python bindings, and the BigJob package and its dependencies. Each installation automatically creates a README file that describes the installed components, and documents

| Machine | Adaptors Supported |
|---|---|
| Kraken | X509, Globus, SSH, Torque/PBS |
| Ranger | X509, SSH |
| Lonestar | X509, Globus, SSH |
| Blacklight | X509, Globus, SSH, Torque/PBS |
| Steele | X509, Globus, SSH, Torque/PBS |
| Trestles | X509, Globus, SSH, Torque/PBS |

**Table 1: CSA Deployments on XSEDE Resources**

settings required to use the installation, settings such as the `LD_LIBRARY_PATH` and so on. A module file is also generated and on some hosts is symbolically linked to the system default module path.

After each update, a set of unit tests is run *on the target machines* to ensure that not only the updated version is deployed, but more importantly, that it is correctly installed and configured for that target machine. The unit tests range from basic (low-level) environment testing, to application level test runs of a BigJob application. The test README, module file and test results are all committed to the central SAGA code repository, and (partially) used to document the current state of deployment on the SAGA deployment wiki.

## 5.2 Developments and Challenges

The use of multiple resources brings about certain challenges which tend to reappear in every distributed ECSS project. For example, each of the machines (Lonestar, Ranger, Kraken, and Trestles) may have different user environments, different shells, and different invocation order of startup profiles on both login and compute nodes. A major complication is the differences in system versions of Python. Most machines have an older version of Python supplemented by a Python module. The Python module did not typically include the tools required to easily install user-side python modules. Therefore, a fresh installation of Python is always present in the shared CSA space. A similar issue is encountered with GCC compiler versions as well.

A slightly more intricate issue is the use of custom "mpirun" wrappers. On Ranger/Lonestar, it is "ibrun", on Kraken, it is "aprun". These wrappers massage the nodelist files, aggregate important environment variables to launch with the application and so on. Modifications have to be made to the launch mechanism in BigJob to account for the use of these scripts.

Job submission is another interesting issue. Lonestar and Ranger use SGE. Kraken and Trestles use PBS. While SAGA retains the ability to submit jobs through the Globus ([9]) job adaptor, it is an unnecessary burden on users. Furthermore, when Globus submitted jobs fail, they generate a very lengthy error report without much useful information. Both projects needed an immediate, clear, and fail safe mechanism to submit jobs and this lead to the development of the *pbs-ssh* and *sge-ssh* plugins to support both PBS and SGE. The plugins enable local/remote launch of BigJob agents using traditional PBS/SGE script over SAGA ssh job adaptors.

The last issue is a user-side issue. The more diverse the machines and their environments are, the more diverse the

documentation and the higher the entry barrier. For example, Kraken requires the initialization of a "myproxy" for successful job submission, whereas Ranger and Lonestar do not. These small but critical differences can mean the difference between successful several thousand core jobs or a week of waiting in the queue to exit on an error.

### 5.3 Testing and Documentation Process

The source code for SAGA and BigJob is stored in a git repository ([2]). A github wiki is used to store user guides for each of the individual XSEDE machines. The BigJob wiki stores all information about the installation and configuration of BigJob. Only users of the BigJob development group can edit this wiki. This wiki is public and can be shared amongst all collaborators. Public wikis also serve as a way to promote other people to use and try BigJob for their scientific needs.

A BigJob CSA release is the result of a production pipeline. Any newly developed features, code modifications and bug fixes are created in branches. After review, branches are merged onto the master branch of the git repository. The main developer determines when a new version of BigJob will be released.

Members of the BigJob development team then checkout the release candidate version from git and test on XSEDE resources. Tests include checking output and error logs, ensuring that the installation is seamless, ensuring backwards compatibility with existing scripts and workflows, monitoring the submission environment and so on. This rigorous quality assurance process is repeated for every machine.

For every machine there are at least two machine-specific examples in the git repository and CSA space. These are example BigJob scripts that run a job in both single and multiple communication (i.e. MPI) mode. The first script simply runs a shell "/bin/date" command. Since the ECSS project supports molecular dynamics simulations, the second script runs a real AMBER MD simulation. These scripts test the basic functionality of BigJob using the CSA installations and serve as standard tests.

With each testing phase, the testing team follows the instructions in the user guide, step-by-step, in a sterile user environment to make sure the documentation is correct and the updates are backwards compatible. Naturally, each machine has scripts that are tailored to the batch queue submission system on that machine – i.e. Ranger uses an SGE submission while Kraken uses PBS. By testing these across multiple job submission systems, we are also exercising the backend job submission BigJob code to make sure that changes have not negatively affected the submission. After the jobs finish successfully, the output is analyzed to validate the results. The AMBER scripts also serve as a test for the AMBER installations on the appropriate machines, and allow the testing team to check that AMBER starts and runs normally.

After testing is complete, the python code is pushed to the Python Package Index (*pypi*). This code is then deployed into the CSA space using pypi. Careful consideration is taken to ensure that the updates to CSA space will not impact any current users of BigJob on the XSEDE resources. Another round of testing is then completed to verify that the CSA installations are working and no changes to the users' environments are required. Any corresponding documentation on the github wiki is updated to reflect the changes.

In addition to this release process, the BigJob team is engaged with members of the ECSS team in order to resolve any system-level issues that may arise. These issues may include but are not limited to differences in schedulers or MPI implementations. Additionally, the scientific collaborators use the wiki as a starting point to run their applications on XSEDE machines. The scripts and associated documentation explain how to use BigJob with their own applications simply by specifying the job description.

If the end users encounter any issues, the BigJob team works with the ECSS consultants to resolve the problems. A mailing list including ECSS consultants, BigJob development and deployment teams and the end users ensures the fastest possible response time. If any questions arise during the end user's use of BigJob, the BigJob team is available to fully assist them. This assistance can range from simple diagnostics of unexpected output to the creation of custom scripts for the researchers to use. For instance, some MD simulations run until a certain time step and then need to be restarted, thus requiring a custom BigJob script. The BigJob team then leveraged the existing BigJob functionality to create a custom script that can submit a number of jobs after the first set of jobs finishes. In the future, we plan to use a shared space where scientists can share their custom scripts and workflows with others.

## 6. COMPUTATIONAL PROGRESS

In this section, we aim to demonstrate how the end-users are utilizing the infrastructure in novel and interesting ways that were not possible previously. In order to do so, we provide a couple of representative configurations of the size of the BigJob submitted and the number of sub-jobs executed as part of the BigJob.

| Machine | # of BigJobs | BJ Size (cores) | Duration (hours) | # of SubJobs | Size (cores) |
|---|---|---|---|---|---|
| Lonestar | 2 | 1200 | 24 | 100 | 12 |
| Lonestar | 5 | 2400 | 24 | 50 | 48 |
| Ranger | 1 | 2400 | 24 | 50 | 48 |
| Ranger | 1 | 1800 | 24 | 50 | 48 |
| Kraken | 1 | 1800 | 24 | 50 | 36 |

**Table 2: Configuration of Jobs submitted as part of ECSS Project 2.**

| Machine | # of BigJobs | BJ Size (cores) | Duration (hours) | # of SubJobs | Size (cores) |
|---|---|---|---|---|---|
| Lonestar | 1 | 200 | 24 | 50 | 4 |
| Ranger | 1 | 800 | 6 | 50 | 16 |
| Ranger | 1 | 800 | 3 | 50 | 48 |

**Table 3: Configuration of jobs submitted as part of ECSS Project 1. Interestingly, these tasks were data analysis tasks (Normal Mode Analysis).**

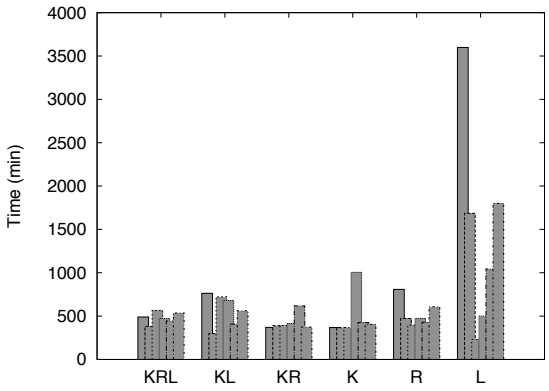


**Figure 3: This figure compares the time to completion (including waiting time) when the BigJobs are submitted to only one machine and when BigJobs are submitted to multiple machines. Here we submit jobs to Kraken, Ranger and Lonestar and assign the workload to the BigJob that becomes active first. 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.**

From the data in Table 2 and Table 3, it is evident that the end-users are able to marshal a very large number of simulations concurrently, which they were not able to do earlier. This is thanks to the fact that BigJob provides first-class support for ensembles of simulations.

BigJob has been used extensively for ensemble-based simulations as well as data analysis. Table 3 shows several hundred individual simulations launched within BigJob Pilot or container jobs. The first use case for example included several hundred independent analysis simulations that are part of the efforts to understand HIV drug resistance.

Simulations captured by the data in Table 3 were used to investigate mutational impact on drug binding in the HIV-1 reverse transcriptase enzyme. In this system mutations distal to the drug binding site induce resistance and the scientists are seeking plausible explanation for how this occurs.

Although the software operating environments of Kraken and Ranger are very different, the deep integration of SAGA and BigJob on XSEDE allow the operating environment (i.e. BigJob) to remain invariant for the end-user. This leads to seamless uptake of different machines for advanced capabilities, i.e., these capabilities are not tied to a specific machine as they historically were, and the end-user scientist can utilize these capabilities not only over individual different resources but multiple different resources concurrently. This can be seen from the plot shown in Figure 3, whereby, several BigJobs for Project 2 were executed on multiple XSEDE resources concurrently.

# 7. DISCUSSIONS AND LESSONS LEARNT

The strong collaboration between XSEDE system side consultants and middleware developers ensures rapid functionality development. In addition, bottlenecks and show-stopper-bugs are identified very early in the development cycle and squashed immediately. A specific aspect that can withstand improvement is the collaboration between XSEDE consultants and the end-user scientists. Having an ECSS consultant whose research overlaps with the scientific aims of both projects would have been welcome. The most effective aspect of this project is undoubtedly the overlap of the two ECSS projects in terms of tools and infrastructure (software and hardware). While the ECSS staff's research interests are not aligned with MD simulations, they are however familiar with the particular MD simulation tools and packages used.

The triple-pronged approach to the ECSS consultants insulated scientists from the "plumbing" of infrastructure allowing them to focus on the science. The middleware/tool developers were free to focus on the computational challenges presented by the scientific problem and ensured strong collaboration with the ECSS consultants. The end result is an infrastructure solution that satisfies the computational needs and is system/resource friendly.

The technical difficulties encountered in porting from the first machine to the second machine prepared the ECSS team to the possible pitfalls for porting to the third and fourth machines. Consequently, the testing procedure incrementally became more rigorous as the number of machines increased. Perhaps the most important technical lesson learnt is the need to keep track of common problems and expect them to repeat on new systems.

The importance of a rapid "hot-patching" deployment scheme cannot be underestimated. Whenever a bug is discovered it is immediately reported by the end users or testing team to the consultants. Issues relating to the system are quickly identified and workarounds suggested to the development team. The development team creates a hot-fix, the deployment team pushes it to the machines and it undergoes testing to ensure the user issue is resolved. Having a tightly knit group of developers and ECSS consultants makes the entire difference between resolving bugs in deployment or having end users abandon the infrastructure due to unresolved bugs. This is where involving the ECSS consultants in the infrastructure development cycle pays off.

In summary, we encourage other ECSS projects and ECSS project management to consider adopting the following best practices:

- Align multiple projects that use the same infrastructure with the appropriate consultants. This is the most ECSS consultant "bang" for scientific discovery "buck"
- Establish strong collaboration between middleware and infrastructure developers and the ECSS consultants as early as possible. Familiarity with the software makes for early identification of problematic issues, quick bug fixes and easier testing
- Establish strong collaboration between ECSS consultants and the science end-users whenever possible. In our project we have not been as lucky as to have an ECSS consultant whose scientific research is aligned with the end-users but experience from the ECSS consultants on other projects show that this is highly desirable.

# 8. ACKNOWLEDGMENTS

We are grateful to Abhinav Thota who provided much of the initial support (supported under Bishop by NIH-R01GM076356) and to Ole Weidner for support with SAGA/BigJob. We are grateful to Andre Luckow for the original development of BigJob. Part of this work also was performed as part of an ECSS in collaboration with Ron Levy and Emilio Gallicchio. We also thank Dave Wright (UCL) and Charlie Laughton (Nottingham) for providing useful input and feedback. This work is funded by NSF CHE-1125332 (Cyber-enabled Discovery and Innovation), HPCOPS NSF-OCI 0710874 award, NSF-ExTENCI (OCI-1007115) and NIH Grant Number P20RR016456 from the NIH 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 the Cybertools project (PI Jha) NSF/LEQSF (2007-10)-CyberRII-01, NSF EPSCoR Cooperative Agreement No. EPS-1003897 with additional support from the Louisiana Board of Regents. Computing resources used for this work were made possible via NSF TRAC awards TG-MCB100111 and TG-MCB090174 and LONI resources.

## References

[1] J. L. Banks, H. S. Beard, Y. Cao, A. E. Cho, W. Damm, R. Farid, A. K. Felts, T. A. Halgren, D. T. Mainz, J. R. Maple, R. Murphy, D. M. Philipp, M. P. Repasky, L. Y. Zhang, B. J. Berne, R. A. Friesner, E. Gallicchio, and R. M. Levy. Integrated Modeling Program, Applied Chemical Theory (IMPACT). *Journal of Computational Chemistry*, 26:1752–1780, 2005.

[2] Bigjob: Saga-based pilot-job implementation. `https://github.com/saga-project/BigJob`.

[3] T. Bishop. High-throughput / high-performance md studies of the nucleosome. `https://www.xsede.org/web/guest/ecss-projects`.

[4] D. Case, T. Darden, T. Cheatham, III, C. Simmerling, J. Wang, R. Duke, R. Luo, R. Walker, W. Zhang, K. Merz, B. Roberts, S. Hayik, A. Roitberg, G. Seabra, J. Swails, A. Gotz, I. Kolossvary, K. Wong, F. Paesani, J. Vanicek, R. Wolf, J. Liu, X. Wu, S. Brozell, T. Steinbrecher, H. Gohlke, Q. Cai, X. Ye, M.-J. Hsieh, G. Cui, D. Roe, D. Mathews, M. Seetin, R. Salomon-Ferrer, C. Sagui, V. Babin, T. Luchko, S. Guasrov, A. Kovalenko, and P. Kollman. Assisted model building with energy refinement (amber). `http://ambermd.org/`, April 2012.

[5] Dare gateways. `http://dare.cct.lsu.edu/`.

[6] Extended collaborative support. `https://www.xsede.org/ecss`.

[7] European Grid Initiative. `http://web.eu-egi.eu/`.

[8] E. Gallicchio, R. Levy, and M. Parashar. Asynchronous replica exchange for molecular simulations. *Journal of Computational Chemistry*, 29:788–794, 2008.

[9] Globus. `http://globus.org/`.

[10] IU. Indiana university pervasive technology institute: Quarry. `http://pti.iu.edu/hps/quarry/`.

[11] J. Kim, W. Huang, S. Maddineni, F. Aboul-ela, and S. Jha. Exploring the rna folding energy landscape using scalable distributed cyberinfrastructure. In *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, HPDC '10, pages 477–488, New York, NY, USA, 2010. ACM.

[12] J. Kim, S. Maddineni, and S. Jha. Characterizing deep sequencing analytics using bfast: towards a scalable distributed architecture for next-generation sequencing data. In *Proceedings of the second international workshop on Emerging computational methods for the life sciences*, ECMLS '11, pages 23–32, New York, NY, USA, 2011. ACM.

[13] 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*, CCGRID '10, pages 349–358, Washington, DC, USA, 2010. IEEE Computer Society.

[14] R. Levy. Distributed and loosely coupled parallel molecular simulations using the saga api. `https://www.xsede.org/web/guest/ecss-projects`.

[15] A. Luckow and S. Jha. Abstractions for loosely-coupled and ensemble-based simulations on azure. *Cloud Computing Technology and Science, IEEE International Conference on*, pages 550–556, 2010.

[16] A. Luckow, S. Jha, J. Kim, A. Merzky, and B. Schnor. Adaptive Replica-Exchange Simulations. *Royal Society Philosophical Transactions A*, pages 2595–2606, June 2009.

[17] 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.

[18] J. C. 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*, 26:1781–1802, 2005.

[19] redis. `http://redis.io/`.

[20] The SAGA Project. `http://www.saga-project.org`.

[21] A. Thota, A. Luckow, and S. Jha. Efficient large-scale replica-exchange simulations on production infrastructure. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 369(1949):3318–3335, 2011.

[22] XSEDE. `https://www.xsede.org/`.

[23] The intelligent transport layer - zeromq. `http://www.zeromq.org/`.