AIMES Final Technical Report

Daniel S. Katz

1 Project Information

DOE award number	DE-SC0008617
Name of the recipient	University of Chicago
Project title	An Integrated Middleware Framework to Enable Extreme Collaborative Science (AIMES)
Name of the project principal investigator	Daniel S. Katz
Date of report	January 31, 2017
Period covered by this report	September 1, 2014 — August 31, 2016

2 Overview

Many important advances in science and engineering are due to large-scale distributed computing. Notwithstanding this reliance, we are still learning how to design and deploy large-scale production Distributed Computing Infrastructures (DCI). This is evidenced by missing design principles for DCI, and an absence of generally acceptable and usable distributed computing abstractions. These gaps underlie the following observations:

- Distributed applications are characterized by limited functional sophistication, yet they are not simple to develop. "Heroic" effort is required to add or incorporate new functionality. This suggests a possible lack of reliable abstractions for distributed applications.
- Existing production DCIs such as OSG and XSEDE are designed for specific classes of applications. It is neither easy to port applications from one DCI to another, nor to expand the type of applications supported by each DCI. Furthermore, it is not easy to interoperate across production DCIs. This is due, at least in part, to a lack of infrastructure abstractions that represent DCI well enough to enable developers to build richer applications.
- The lack of an adequate set of abstractions for distributed applications and infrastructure, as well as design principles for DCI, results in: (i) applications that are brittle, (ii) an inability to reason about the spatiotemporal execution of distributed workloads, and answer even basic questions about their execution. For example: Which resources should a workload use? What time-to-solution should a workload expect? How do specific execution decisions influence the performance?

The AIMES project was conceived against this backdrop, following on the heels of a comprehensive survey of scientific distributed applications [1]. The survey established, arguably for the first time, the relationship between infrastructure and scientific distributed applications. It examined well known contributors to the complexity associated with infrastructure, such as inconsistent internal and external interfaces, and demonstrated the correlation with application brittleness. It discussed how infrastructure complexity reinforces the challenges inherent in developing distributed applications.

We have identified several additional contributing factors responsible for infrastructural complexity, for example, the absence of extensible and scalable abstractions for resource management. One consequence is that different applications manage their execution and distributed resource utilization in diverse if not irreconcilable ways, both in principle and practice. This results in applications being bound to specific platforms, thus acting as a further barrier to interoperability. Determining widely-usable abstractions for resource management is a non-trivial undertaking, made harder by the fact that DCIs are characterized by both spatial and temporal fluctuation.

An important driver of AIMES was to first identify such abstractions, and then to translate them into scalable and uniform distributed resource management approaches and capabilities. A relevant prerequisite was the need to "formalize and normalize" the set of decisions that are needed to execute a distributed application on multiple and diverse resources. A related issue was integrating information from the application and infrastructure, as the nature of execution decisions is intimately related to the type of information available.

In the absence of adequate abstractions, how can we design experiments to investigate or support different ways to execute distributed applications, while being extensible and scalable to real science applications? Furthermore, given the broad range of application properties, resource characteristics, and the type and reliability of resource information, how can we "experiment" with and "integrate" different application requirements, infrastructure types, and information?

AIMES laid the foundations to address the tripartite challenge of dynamic resource management, integrating information, and portable and interoperable distributed applications. Four abstractions were defined and implemented: **skeleton**, **resource bundle**, **pilot**, and **execution strategy**. Skeletons are synthetic applications with a spectrum of characteristics and tunable properties to mimic the behavior of real-life distributed applications [2]. Resource bundles represent an aggregate set of resources of fluctuating capacity that hide the heterogeneity and dynamism inherent in DCI resources. The pilot abstraction is well known in distributed computing but inconsistently used and understood. A sound theoretical basis was provided for the pilot abstraction [3, 4] and its applicability was extended also to high-performance and data-intensive computing [5, 6]. Execution strategies model the set of decisions that need to be made in order to execute a distributed application on diverse resources [7].

The four abstractions were implemented into software modules and then aggregated into the AIMES middleware. This middleware successfully integrates information across the application layer (skeletons) and resource layer (Bundles), derives a suitable execution strategy for the given skeleton and enacts its execution by means of pilots on one or more resources, depending on the application requirements, and resource availabilities and capabilities.

AIMES Results

AIMES has advanced the state of distributed computing in at least three important ways:

1. We have designed and implemented powerful and extensible abstractions for distributed applications and resources. Collectively, they provide the ability to execute applications on dynamically varying resources. However, exposing simplicity is not the same as hiding complexity. To that end, we support scalable execution of a range of applications on production DCI by integrating abstractions at multiple levels. Scale, generality, extensibility, reasoning, and repeatability are the basis for the claim that the complexity of DCI at large scale can be managed via abstractions. Collectively, the abstractions provide an integrated approach to resource-management in distributed, high-performance and data-intensive scenarios, which were previously considered distinct.

2. Building upon the ability to integrate information, we have proposed the abstraction of execution strategy. Execution strategies support end-to-end reasoning by capturing the decisions involved in the execution of different applications independent of the underlying resource utilization model. Furthermore, the AIMES infrastructure can take an execution strategy and implement it for a distributed application. This permits empirically determining a suitable, if not yet optimal, execution strategy for a given application using a given set of resources, and thus provides the ability to examine the impact of execution decisions on scalable execution as a function of different application and resource characteristics. We have shown that execution strategies can improve the qualitative and quantitative aspects of distributed execution [7], dissolving the artificial barriers between high-throughput and high-performance.

3. We have implemented an experimental "laboratory" that supports investigation of questions and along dimensions that were not originally conceived. The AIMES infrastructure supports the design of new experiments and analyzing tradeoffs of different decisions; the same software system also supports scalable science. For example, the AIMES infrastructure supports experimental workloads on five and more distributed resources, but the RADICAL-Pilot component also supports thousands of MPI jobs used for biophysical simulations [8].

These environment enabled by these advances can be favorably compared to an earlier landscape dominated by distributed applications that needed high-levels of customization and had limited scalability on the one hand, with DCIs that were characterized by "gluing it together" and the absence of a systems approaches on the other.

This report is organized as follows. In §3, we discuss progress on specific aims and accomplishments, and provide a brief overview of progress in Project Year 3 (PY3) and and the no-cost extension year (collectively referred to as PY3/4) toward the goals of the overall project. In §4, we provide detailed technical discussion along the three functional tracks. Although our focus in PY3/4 has been on prototyping and initial validation of abstractions, along with the integration of the three tracks, we will also discuss some of the "deeper" conceptual issues that have been exposed in our initial work in §5. Like most successful research, at least as many new questions have arisen as have been answered. Subsequent sections of the report address administrative, personnel, and financial aspects of the project.

3 Project Year 3/4 Accomplishments: Overview

Effort in AIMES Project Years 1 and 2 (PY1, PY2) primarily focused on initial development and integration: bringing together the applications, resource, and middleware tracks of the project to

deliver simple yet non-trivial integrated end-to-end capabilities. The integrative end-to-end capability provided the project with necessary validation of the conceptual approach and proposed abstractions, while providing useful experience in the design of interfaces and engineering interoperable systems and solutions. AIMES enhanced our understanding of extreme-scale distributed science via the execution of applications and the dynamic federation of resources in PY3/4. Specifically:

- In PY1, we designed, developed, and integrated preliminary implementations of abstractions for distributed applications and resources.
- In PY2, we demonstrated integrated capabilities and experimented with a range of execution strategies on dynamic federation of heterogeneous resources via pilot overlays on large-scale production DCI.
- The primary goals of PY3/4 were to make empirical progress towards both a quantitative and qualitative understanding of the limits and effectiveness of the AIMES approach, thus there was a need to design and implement experiments that would provide insight into the scalability and effectiveness of the approach. In PY3/4, we generalized the core capabilities to a wider range of resource and information types, applications, and infrastructure classes. A specific focus was on extending the four AIMES abstractions to the Open Science Grid (OSG), and thereby providing, a common conceptual basis for applications to reason about execution and resource management across different distributed resources.

We discuss progress towards these high-level PY3/4 goals by describing advances along the objectives listed in our original proposal.

3.1 Objective 1: Transparent Computation and Data

The flexible and dynamic utilization of resources is important for extreme-scale science, with multiple conditions that must be met in order to enable this. One necessary condition we focus on is transparent "computation and data." We define computation and data *transparency* as a property that is a conjunction of both resource interoperability and location agnosticism. More specifically, compute and data resources can be interoperably utilized, and compute and data elements can be retrieved/placed independent of their location.

Such transparency can best be achieved via the conceptualization and design of suitable abstractions. The AIMES project posits two initial abstractions—pilot and Bundle—working collaboratively and cooperatively to provide transparency from the details of the levels below. Thus, compute and data locations are transparent to the applications thanks to the pilot abstractions; in turn, the resource details are transparent to the pilot abstractions via the Bundle abstractions.

It is important to distinguish between a representation of resources from the information used to characterize their availability. In PY3/4, we extended resource representations from PY1/2 (supercomputer Bundles for CPU resources) to much more heterogeneous grid Bundles (both CPU and network resources) on OSG. This brings in a much richer collection of heterogeneous dynamic resources and resource aggregations based on similar CPU and memory characterizations. OSG is particularly challenging as it exposes a view of resources that is more transparent than the view exposed by supercomputers such as XSEDE, in that it can be difficult to track specific resources in OSG. Characterizing OSG resources and integrating them into Bundles was a further validation of the Bundle concept of providing transparent computation and data.

3.2 Objective 2: Multi-level Integration

In order to ensure that research ideas are systematically examined for suitability at extreme-scales, research and development must progress *in-phase*. This enables modular and iterative development, as well providing timely testing of assumptions. Research ideas should ideally be tested against real-world application scenarios and at scale; thus, there should be integration between applications and infrastructure. However, this opens up a difficult set of design and deployment decisions, as real-world applications and infrastructure are complex and difficult to develop/deploy against.

Our approach to end-to-end integration is mediated by the middleware framework. That is to say, at the upper-level, applications are integrated with the middleware framework, and the middleware framework's integration with the infrastructure layer is made easier and robust via the Bundle abstraction. The middleware in turn relies upon the application abstractions, which provide the ability to capture the essential properties of applications while not exposing their full complexity, and by being free of the burdens imposed by the complexity of the infrastructure. Application skeletons [9] and Synapse [10] (both of which which we discuss in §4.1) are examples of application abstractions.

In PY1, we made significant progress towards the integration of application skeletons with the pilot abstractions. We began the design of the integration of pilot abstractions with the Bundle framework, namely, the separation of functionality and interface aspects.

In PY2, we successfully finished initial integration at both levels. This was captured in a demonstration at SC'13 where we defined a skeleton for a Bag-of-Tasks (`dd`) and ran it on a pilot-based overlay on multiple heterogeneous resources (on FutureGrid), which were in turn based upon dynamically acquired information (as provided by the Bundles).

In PY3/4, we improved and tightened the integration of application skeletons into the AIMES middleware by allowing skeletons to be called directly from the middleware (vs. via a command line interface, as was the case previously) and returned the application structure output of the skeleton (the description of the tasks to be run and their dependencies) directly to the middleware (vs. to a file that the middleware could read in, as was the case previously.) Further, we also enabled the use of Synapse emulator via Skeletons, adding the possibility to set the compute quantify for a task as a number of FLOPS. When used as an end-to-end solution, the AIMES middleware is now able to distribute executions on all the HPC XSEDE resources, the XSEDE OSG Virtual Cluster, BlueWaters, and Titan. Workloads can be executed concurrently on one or more pilots instantiated on each of these machines. Finally, the AIMES middleware and the Swift workflow system were integrated via a REST interface, enabling the execution of scientific workflows described in the Swift scripting language.

3.3 Objective 3: Deployment, Testing, and Evaluation

Based on results in PY2, we completed and extended the deployment, testing, and evaluation of the AIMES middleware to all the HPC and OSG resources of XSEDE and to Titan. This involved the following modules: synthetic distributed applications emulators Synapse and Skeletons; the workload manager eManager; the information system Bundle; the pilot system RADICAL-Pilot; and the interface for interoperability across resources called RADICAL-SAGA.

We improved the integration of the AIMES middleware modules, publishing most of them on the PyPI (`https://pypi.python.org`) repository to facilitate deployment by means of pip, the *de facto* standard installation tool for Python modules. As part of this publishing process, we refactored the

documentation of all the modules, refining instructions and examples. All this contributed towards a production-ready version of the AIMES middleware, even if more work is needed to complete this process, especially for the eManager and Bundle modules.

Two fundamental advancements have been achieved by the AIMES middleware in PY3/4: support for the XSEDE OSG Virtual Cluster and support for Titan. As described in §4.3, this required development of specific new capabilities across the whole stack and a refinement of the procedures we use to deploy its python modules. On OSG, the integration between Skeleton and Synapse has enabled the description of workloads as skeletons and their profiling and emulations via Synapse. Synapse can now be downloaded at runtime on every OSG resource exposed by the OSG Virtual Cluster's broker, enabling the emulation of synthetic distributed applications on OSG. The same can be done on Titan, executing Synapse via the newly developed Open Run-Time Environment (ORTE) layer of RADICAL-Pilot.

Our progress on the deployment of the AIMES middleware has enabled its testing at a greater scale than during PY2. We performed about 400 experimental runs on OSG alone, executing workloads of up to 2,048 tasks on between 8 and 2,048 single-core pilots. With this work we developed a methodology to characterize the distribution of workload executions on OSG, measuring the distribution of their time to completion on the pool of resources currently made available to the XSEDE users. Approximately 1,000 experimental runs have been performed on Titan, executing between 24 and 16,000 tasks per run, on between 1 and 1,000 compute nodes. This has enabled the ongoing evaluation of alternative execution strategies for the ATLAS Monte Carlo workflow as well as simulation-analysis workflows for Molecular Dynamics.

Testing performed at this scale exposed the differences between XSEDE, OSG, and Titan when considering the execution of workflows with both homogeneous and heterogeneous tasks. This insight is greatly advancing our understanding of distributed executions, especially regarding the management of the non determinism and dynamism of resources. Specifically, the AIMES project has enabled distinguishing between the benefits of executing tasks requiring one or more cores, input files of different size, on infrastructures with fundamentally different overheads in terms of queue waiting time, duration of pilot availability, and statistical distribution of application execution time and data staging times.

The evaluation of the AIMES middleware has been particularly important in PY3/4. We profiled and emulated real-life workloads both from the domains of molecular dynamics and particle physics, measuring the performance gain and overheads imposed by the AIMES middleware on their execution. This work prompted several optimizations of our software stack but also a principled and experimental understanding of the benefits and limitations of distributed executions on different type of resources. We learned that fault-tolerance and failure management is necessary on OSG, while distribution across at least three HPC XSEDE resources allows us to overcome overheads typical of HPC resources. We also learned the limits of our middleware in terms of data staging, especially on OSG where more development is needed to execute data-intensive workflows. On Titan, we learned how to leverage the opportunities offered by back-filling but also the limitations of our current implementation of the ORTE communication and coordination layer.

3.4 Project Plan and Schedule Status

As seen in Table 1, we have either finished or made significant progress towards most of the stated PY3 aims. We discuss the details of our accomplishments in the next section.

Table 1: Project Plan for PY3. A ✓ represents a completed goal; an ⇑ represents a goal that was partially addressed; a □ represents a goal that was not addressed.

Objectives	Year 3 Aims	Notes
Objective 1: Transparent Computation and Data	□ Develop an understanding of the resource network layer and a strategy for its integration within the AIMES software toolkit. ✓ Develop characterization of heterogeneous CPU and network resources in multi-site Grid. ⇑ Develop technique to exploit Bundle resource availability and queueing time information on XSEDE for elastic MPI jobs. □ Ensure that skeletons work consistently with applications regarding data and networks.	• We characterized the capabilities of heterogeneous nodes on OSG from the perspective of peak CPU and memory, slot availability on those nodes, and workload patterns. We characterized the network bandwidth available from the OSG data storage node to all other OSG compute nodes. • We built a new scheduling framework called Elastic Job Bundling which exploited Bundle information to reduce waiting time for large jobs by breaking an MPI job into smaller sub-jobs each with smaller wait time and then combining the results. • While skeletons make use of data and network, network requirements were not explicitly inserted as part of the skeleton generation.
Objective 2: Multi-level Integration	✓ Develop AIMES software toolkit V2. ✓ Develop and experimentally measure advanced execution strategies for large-scale, real-life applications. ⇑ Develop a planner to devise and coordinate appropriate execution strategies for a given workload. ⇑ Continue the integration of real-time resource information within planning and scheduling algorithms. □ Develop Storage Bundle concept. ⇑ Explore widely distributed Bundles. ✓ Continue to use Skeletons to accurately represent applications.	• We furthered the implementation of multi-pilot, load-balancing, cross-infrastructure algorithms for both data and compute management. Users were able to leverage a growing number of execution strategies, depending on the characteristics of their given workload and of the targeted resources. • We investigated planning algorithms for the execution of distributed applications and the implementation of a planner module. We supported this investigation with several experiments that characterized both availability and capabilities of HPC and HTC resources. • We integrated Synapse and Skeletons to enable profiling and emulation of the compute and I/O loads of distributed scientific applications. • We implemented Bundle on OSG, and as future work will implement it on a Cloud (such as Amazon AWS).
Objective 3: Deployment, Testing, and Evaluation	□ Develop a taxonomy of the workloads that would benefit from the capabilities implemented within the AIMES software toolkit. ✓ Coordinating with user communities for the early adoption of the AIMES software stack. ⇑ Final testing of Compute and Storage Bundles across all platforms. ✓ Perform integrated testing using both skeletons and real applications from user communities on both sets of HPC and HTC resources.	• We supported multiple types of distributed workloads of uncoupled ensembles as well as loosely-coupled ensembles, such as replica-exchange. • We engaged existing communities further (e.g., high-energy physics and computational biology), but also fostered early adoption by new scientific communities (e.g., climate sciences). • We performed several experiments on both HPC and HTC resources, emulating molecular dynamics workflows as executed by collaborating user communities.

4 Technical Details of Accomplishments

As pointed out in §2, the accomplishments of the AIMES project in PY3/4 came from both development and research efforts. On the development side, a prototype of integration between skeletons, pilot-based middleware, and Bundles is in its final development stage. This prototype will be used by the AIMES project to investigate not only the engineering requirements of such an integrated architecture but also theoretical issues related to performance analysis, smart-scheduling, affinity modeling, and the impact of different parallel application paradigms on the underlying infrastructure layers.

From a research point of view, the AIMES project has started to work on multiple initiatives, each focused on a specific theoretical challenge. Altogether, these lines of research contribute to the creation of a consistent and unifying vision for the next generation of middleware to support extreme-scale research. The output of these activities consists of accepted and under review full-length papers, position papers, abstractions for modeling several aspects of resources and applications, and workshop presentations.

The project has progressed by investigating three main research areas: resource management, models for resource availability, and models for distributed applications. Specifically, the research about resource management investigates how to couple dynamically application requirements with resources availability; models for focuses on the aggregation and abstraction of different types of resources within a heterogenous set of infrastructures; while the research into application models studies the defining characteristics, properties, and behavior of different types of scientific workloads. Altogether, these three lines of research offer both a qualitative and quantitative analysis of the end-to-end process of executing a scientific workload on a set of resources when both the workload and the resources exhibit a dynamic behavior and an arbitrary degree of heterogeneity. Here we provide technical details of work to date along these three tracks.

4.1 Application Track (led by UChicago)

The applications track had three activities.

The first was to study distributed science and engineering applications, focusing on those of interest to DOE, to understand their current state and project where they might go in the future, and to understand the capabilities that their developers and users feel are needed in the future. This was primarily done in Y1, though we continued to remain in contact with such application communities during the rest of the project.

The second was to develop a representations of these applications that are useful to the AIMES project (and other projects working in the same space of developing cyberinfrastructure systems and tools). We have done this through our Application Skeleton tool.

The current Application Skeleton tool can create easy-to-access, easy-to-build, and easy-to-run bag-of-task, (iterative) map-reduce, (iterative) multi-stage workflow applications. The Application Skeletons tool uses a top-down approach to describe an application by stages, each of which comprises a set of tasks. The Application Skeleton tool allows users to specify the task type (serial or parallel), the number of processes of each task, runtime, read/write buffer size, computation and I/O interleaving option, input/output files, and associated file size. Dependencies between tasks in different stages are in the form of file production and consumption. Users can specify file-to-task mapping and cross-stage file referencing in the application description file. The Application Skeleton tool can generate synthetic applications that are compatible with tools including Swift,

Pegasus, and Bash shell.

For three representative workflow applications (Montage, BLAST, and CyberShake PostProcessing) we have studied, we have compared the real application performance with that of the skeleton application. The performance comparison shows overall error of -1.3%, 1.5%, and 2.4% for Montage, BLAST, and CyberShake PostProcessing, respectively. We also have shown that using Application Skeletons tool can simplify the process of system optimization design, implementation, and verification using four common techniques as examples: data caching, task scheduling, I/O tuning, and a resilience mechanism [9].

During PY3/4, we improved Skeletons through feature additions, bug fixes, code improvements, and through integration with the Synapse tool developed at Rutgers that allows work to be expressed in systemless units (e.g., operations rather than seconds). The feature additions were needed for integration with the rest of the AIMES project. Specifically, the skeleton was modified to allow it to be called via an API (in addition to the previously possible calls via the command line) and to output the application description as a JSON object (in addition to the previous output options of a shell script or Swift or Pegasus file.) An example of a bug fix is that use of a distribution function could return non-sensical value (e.g., task lengths as a normal distribution with a mean of 10 seconds and a standard deviation of 5 seconds would sometimes return negative task lengths, which are now bound to be zero or greater.) A sample code improvement is that when mapping input files to tasks, we had used a 'choose' function, which was not scalable when the numbers of files and tasks grew, so we replaced it with calls to the Python itertools package. A summary of the skeleton tool and the synapse tool was published in an FGCS paper in 2016 [2].

The third activity was to integrate application skeletons with the other components of the project, using a variety of systems. We worked with the other parts of the project, led by Rutgers, to perform detailed testing of the various integrated scenarios, as described in §3.3 and §4.3.

4.2 Resource Modeling Track (led by Minnesota)

In PY1/2 our efforts were centered around answering the following question: Given a large number of resource selection choices, how does the AIMES middleware help the user or application make better decisions on resource usage? Our concrete accomplishment toward this goal is the Bundle resource manager middleware, or Bundle for short.

Bundle has been designed to provide a resource abstraction layer. Architecturally, Bundle consists of the BundleManager and BundleAgent. The BundleManager is an entity that aggregates information collected by BundleAgents. BundleAgents are a set of entities that each monitors a resource platform. The Bundle can be used as a subsystem of AIMES or as standalone software. Based on the implementation of Bundle, we studied several research topics. Bundle was validated on supercomputer resources such as XSEDE and DOE Hopper during this time period.

In years PY3/4, the resource management track was extended to Grids such as OSG. This introduced far more heterogeneity, distribution, and dynamics. In OSG, resource availability is much more difficult to characterize due to the transparent nature of resources. The heterogeneity of resources also required methods to aggregate resources to characterize collections of resources often needed by applications.

4.2.1 Characterizing Networks on Grids/OSG

Networks on OSG are notoriously heterogeneous due to their wide-area distribution, lack of centralized control, and use of shared networks. We performed a detailed study of the available bandwidth between a central OSG node called the OSG stash storage system where data should be staged for OSG jobs, and each site in OSG. To accomplish this we used a combination of the iperf tool and our own probing infrastructure. Characterizing the expect bandwidth between a data source and compute node is an important ingredient in making scheduling decisions.

The results shows that (1) the average network bandwidth between OSG stash and different OSG sites range from a few Mbps to a Gbps; (2) the variation of network bandwidth within the same OSG site is bimodal, either very low or very high, indicating that the staging speed on some OSG sites is highly predictable while the opposite is true for others. When high predictability and performance guarantees are required, a scheduler must take this into account for data-intensive jobs. This result validates our assumption that there is a high degree of heterogeneity between the wide-area network connections in OSG, a multi-site Grid environment, and further quantifies the amount of heterogeneity. A sample of our results is shown in Figure 1.

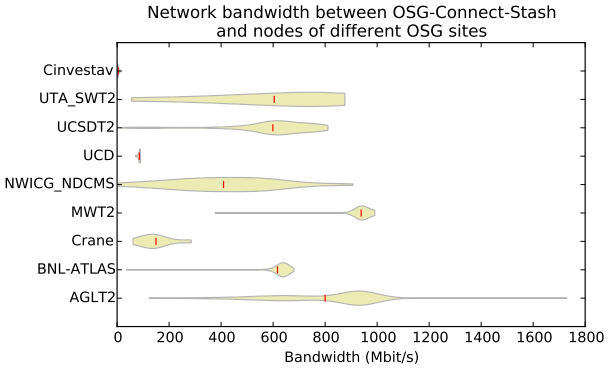

Figure 1: OSG Network Bandwidth Diversity.

4.2.2 Characterizing Heterogeneous Resources on Grids/OSG

Compute resources in OSG have a large variance in capabilities. In our work, we sought to characterize the computational characteristics of OSG nodes in terms of power and in terms of expected workload. To characterize OSG workload and resource availability, we continuously monitored opportunistic Condor slots cataloguing those with in-use or free status across OSG sites. We also continuously monitored the number of waiting and running jobs in OSG sites to further characterize the active workload. We analyzed this information to give a full picture of OSG workload variation.

Due to the federated nature of OSG, the compute capacities offered by OSG sites are time variant. The number of slots that are available to ordinary OSG users who rely on opportunistic slots is affected by how site owners tune the CPU time that are 'reserved' for their internal users. For example, if a site administrator ramps up and starts using some additional CPU hours per month then the available opportunistic slots will drop accordingly. Likewise, if a site administrator ramps down production for some period of time, then the number of opportunistic slots available may rapidly increase for that time period.

Our monitoring facility does not have special privilege to detect when site administrators change the internal reservations, but through historical data, we can tell when the opportunistic slots change. For example, if the number of slots associated with the opportunistic osg-flock condor pool of a certain site decreases while the workload measured by queueing and running jobs stays stable, it is highly likely that this site is experiencing some administrative capacity changing. Under this condition, users querying our data are recommended not to submit jobs targeting the specific site. Likewise, if we observe capacity increase for a certain site with capable nodes, our users are

recommended to direct job submissions to this specific site.

One of the challenges is the transparency of OSG nodes and correlating performance measures with actual node end-points. To solve this problem, we scanned with available nodes by their network MAC address (which uniquely identified the nodes) that were exposed to opportunistic Condor jobs. We recorded the hardware configurations in order to associate the jobs with the actual resources they ran on. So far, we have identified thousands of OSG nodes possessing unique hardware. We observed the trend that many physical nodes apply dynamic slot partitioning: each node is shown as a single slot with multi-cores and all of the remaining memory size. The single slot is then dynamically sliced into compute slots with varying sizes with respect to the specific resource requirements of the jobs which are scheduled onto that node. How users can leverage this system feature will be our future work.

Finally, we characterized the individual compute and memory characteristics of individual OSG nodes via probing jobs. We used the HINT benchmark which is a computer benchmarking tool developed at the Scalable Computing Laboratory (SCL) of Ames Laboratory. Unlike traditional benchmarks, HINT neither fixes the size of the problem nor the calculation time and instead uses a measure called QUIPS (QUality Improvement Per Second). This enables HINT to display the speed for a given machine specification and problem size. The results on the HINT benchmark were tabulated by the BundleAgent. We then clustered the compute nodes by these performance measurements which reflects both their CPU and memory capabilities. An example of clustering OSG compute resources based on similar computation power is shown in Figure 2.

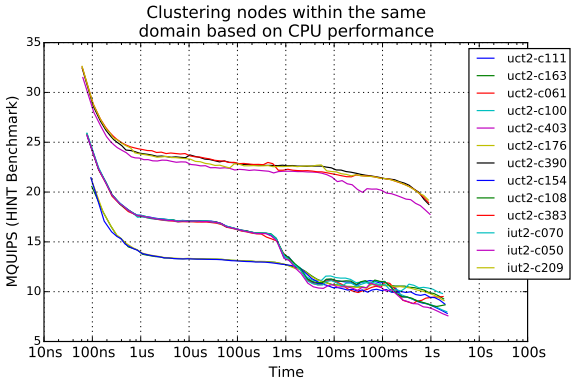

Figure 2: Clustering OSG Compute Resources.

The results of the HINT benchmark show that the compute nodes on each separate OSG site can be clearly clustered into several groups with identical performance within each group. Given a hardware id, Bundle can respond to a query for resources with a specific hardware configuration and HINT benchmark score. Moreover, the user can further query all of the nodes with identical performance. This can help the user choose a subset of nodes with uniform, predictable compute performance. Also, unsurprisingly, OSG is comprised with different generations of hardware, with the high-end servers comparable to individual nodes of a supercomputer to low-end servers with outdated configurations. This should be taken into account when submitting jobs with high performance or time sensitive requirements.

4.2.3 Enhancing Bundle Implementations across supercomputers (XSEDE) and Grids (OSG)

We extended Bundles to support a wide range of interfaces enabling much greater support for diverse platforms including OSG. These interfaces included MongoDB, JSON file and Bundle API upon which a caller can query information from a Bundle. We also added a notion of session to the Bundle interface. A Session is used to hold a BundleManager instance and is used as the root object for all other Bundle objects. Each Session has a uid, which could be used to reconnect

to it. The major consideration of introducing the concept of sessions is due to the distributed deployment of Bundle and WorkloadManager instances. A session records (1) resource list and credentials; (2) location of the database; (3) states maintained for different WorkloadManager instances. Using session, one deployment of Bundle can concurrently serve multiple applications, since each application is uniquely identified. An application doesn't need to maintain a connection with Bundle, it can use its session id to retrieve any previous configuration. The enhanced Bundle architecture is shown in Figure 3.

Another consideration is the speed for a query. Due to the sheer size of data, an on-demand query can be very time-consuming. Bundle periodically performs queries to generate a global state of resources and caches this as a JSON file. Not only does this support our historical resource characterization process, but it also allows a user to get a rapid response to a query with respect to current (and approximate) resource availability.

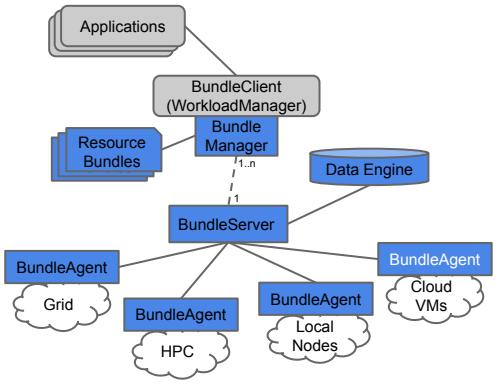

Figure 3: Exhanded Bundle Architecture.

4.2.4 Bring Elasticity to Parallel Applications

During our investigation of the job queueing time prediction problem, we observed that many jobs experience large queueing time. Extreme cases include a job that run for only ten minutes but waited for more than ten hours before being started. This degree of slowdown could seriously impact a user's research progress. If we could develop a method that helps shorten application response time, then it could potentially accelerate the amount of science performed by a user.

With this goal in mind, we proposed a new method in PY2 that allows one application to obtain continuous progress while using resources allocated for multiple jobs. The essential benefit of this approach is that an application does not require all needed resources to be allocated at once for one single job, which is the resource allocation paradigm for which the HPC cluster is designed. Instead, our method could enable an application to break its one integrated resource request into several smaller chunks, and link portions together. This idea is inspired by the concept of elasticity generally adopted by cloud computing.

The reason why this approach could result in faster turnaround time is that intuitively a smaller sized resource requests are more easily fulfilled than bigger ones. For example, smaller jobs asking for less number of processors or shorter run length have a better chance of being backfilled. For another, even for clusters encouraging large sized jobs by assigning to them higher priority, resource fragments could be better utilized by smaller jobs. It is likely that before a large resource request is fully satisfied, an application may finish by taking advantage of the accumulation of smaller jobs. This opportunistic method should guarantee that applications would not finish later than when this approach is not applied.

We are working on both the algorithm and mechanism of enabling this new approach. Algorithmically, our middleware needs to figure out when and how to break larger resource requests into smaller ones. This algorithmic problem is interesting due to unpredictability of resource availability and its dynamic changes.

From the mechanism's side, when an application is composed of tasks that can run independently (pleasingly parallel), simply distributing them among separate jobs can be an easy and feasible so-

lution. However things are more complicated for tightly coupled parallel applications like programs written in MPI that involve frequent inter-process communication. Our solution is when the number of usable processors is smaller than that needed (typically the number of physical processors is equal to the number of logical MPI processes), our method could let the MPI application run with certain degree of oversubscription. For example, an MPI application whose desired maximum parallelism is 64 processes can run on 32 physical processors with an oversubscription degree of 2 logical processes per physical processor. When another job become available, our mechanism will allow part of the running MPI program to migrate to the newly allocated processors such that degree of oversubscription decreases after process redistribution.We propose to use virtual machine migration to do this. In PY3/4, we have completed an evaluation of the elastic job implementation. Using real-world job traces we showed that EJB can (i) reduce target job's mean turnaround time by up to 48%, (ii) reduce system-wide mean job turnaround time by up to 27%, and (iii) reduce system fragmentation by up to 59%. We have also developed an implementation architecture that can realize this approach.

4.3 Middleware and Modeling Track (led by Rutgers)

The overarching goals of the middleware track led by Rutgers are to: (i) couple diverse application types, classes and patterns with a range of resources types by providing a common interfaces and by exposing common execution semantics, (ii) provide the advanced resource management capabilities, which in turn enable scalable execution of workloads, and (iii) provide a software toolkit that integrates compute, data, and network resources across multiple computing infrastructures en route to supporting computational science requirements at extreme scales. A prototype of such as software toolkits must support empirical examination of both structured decision making as well as investigation of "what-if" scenarios.

In PY3/4 we capitalized on the work done in PY2 and PY1, completing goals i—iii. We defined and formalized the abstraction of 'execution strategy' to model the sequence of decisions that have to be made to execute a workload on resources. Each decision of an execution strategy is based on evaluating how alternative choices satisfy one or more metrics, which in turn is based on a model of how each choice relates to a metric. For example, for the metric time-to-completion of a workload, the decision of how many and which resources to use relies on a model of each resource's compute performance. Decisions are made by users (e.g., via

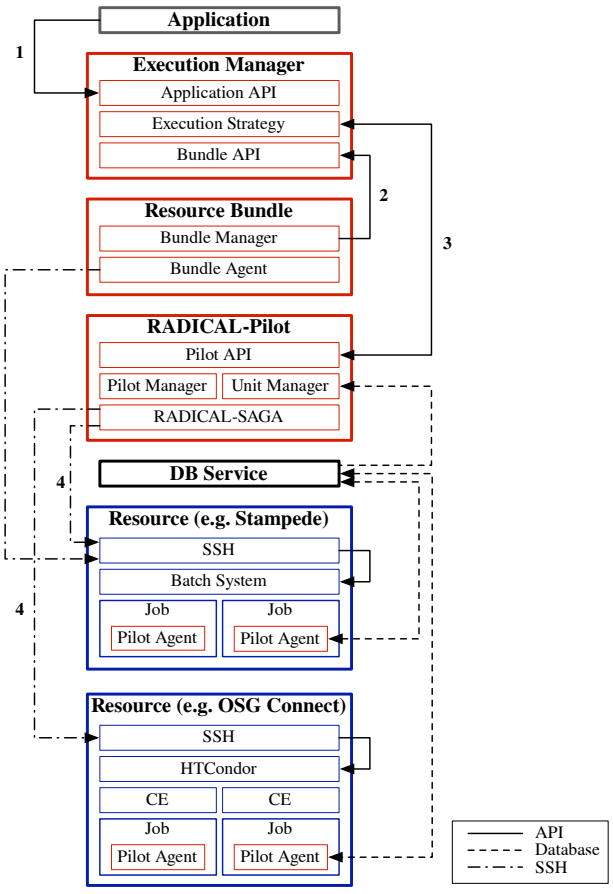

Figure 4: Architecture of the AIMES integrated middleware. Gray = application layer; red = AIMES middleware components; black = supporting services; Blue = resource layer.

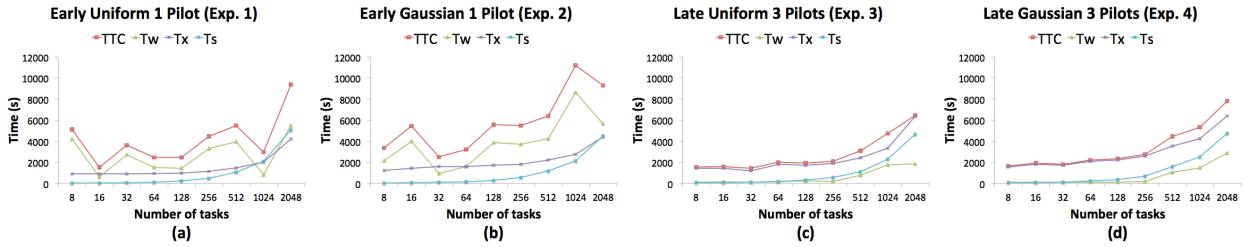

Figure 5: Time constituents of the overall time-to-completion (TTC) of a workload execution. T_w = pilot setup and queuing time; T_x = execution time; T_s = input/output files staging time. During execution (T_w), (T_x), and (T_s) overlap so $TTC < T_w + T_x + T_s$.

configuration files), programmers (e.g., via how the application is written), or algorithms (e.g., via runtime decisions).

In this context, we conducted experimental research to develop models of resources' behavior when performing distributed executions. This required the further development of the AIMES integrated middleware to support experimentation on diverse infrastructures and with diverse workloads, both synthetic and derived from actual use cases. Figure 4 shows the architecture of the AIMES middleware: The prototype of workload manager developed in PY2 was extended to derive and enact specific execution strategies (arrow 1 in Figure 4) using the information provided by the application layer and the Bundle mode (arrow 2 in Figure 4). RADICAL-Pilot and RADICAL-SAGA were used (arrows 3–4 in Figure 4) to enable the submission and management of pilots on both XSEDE and XSEDE OSG resources.

Our experiments characterized the distribution and concurrent execution of application tasks on multiple resources of XSEDE. We focussed on HPC clusters such as Stampede, Comet, Gordon, and SuperMIC, as well as the OSG Virtual Cluster.

4.3.1 XSEDE Characterization

We published the results of four experiments we performed with the AIMES middleware to characterize the execution of distributed applications on multiple HPC XSEDE machines [7]. Specifically, we focused on applications requiring between 8 and 2,048 cores, and with fixed 15-minute runtime or a (truncated) Gaussian distribution of runtimes between 1 and 30 minutes. These durations were chosen so to be consistent with about 35% of the jobs executed on XSEDE.

This study offered at least four important insights: (1) Pilots can be used to enable high-throughput executions on HPC machines, traditionally designed to support mainly single, long-lived MPI tasks; (2) the use of pilots is consistent with the policies of the target HPC machines, i.e., they do not 'game' HPC schedulers and scheduling policies; (3) At the investigated temporal and spatial scales, the time spent by a pilot in the queue of a single HPC machines dominates the overall time-to-completion (TTC) of the workload; (4) Late binding across at least three pilots, each submitted to a different HPC machine, dramatically reduces and normalizes the impact of queuing time on the workload TTC.

Figure 5 summarizes the results of our study, showing the value of the three main components of TTC for each experiment: (1) time setting up the execution including waiting for the pilot(s) to become active on the target resource(s) (T_w, green); (2) time executing all the application tasks on

the available pilot(s) (T_x, purple); and (3) time staging application data in and out (T_s, azure).

T_w is the TTC component with the most variation among experiments and the most contribution to TTC. The variation is more pronounced for early binding (between 600 and 8,600 seconds) less for late binding (between 99 and 2,800 seconds). In early binding, T_w has large variation vs. number of tasks, and between uniform and Gaussian distribution of task durations. In late binding, T_w has a smooth increase vs. number of tasks and almost no variation between uniform and Gaussian distributions of task durations.

Figure 5 shows the dominance of T_w contribution to TTC. In the early binding strategies (Figure 5 (a) and (b)), the red line (TTC) and the green line (T_w) have the same shape, with the variations in TTC determined by equivalent variations in T_w. T_w dominates also the TTC of the late binding strategies. Figure 5 (c) and (d) show how T_w accounts for almost all the TTC for runs with up 256 tasks, and most of the TTC for the remaining number of tasks.

The difference in performance across strategies is therefore due to the duration of T_w. On average, when using three pilots, the first pilot takes less time to become active than when using a single pilot. This is due to the large variability of T_w for the same job submitted multiple times to the same resource. Due to the large variation in T_w on each single resource, when using multiple resources, it is more likely to find at least one of the resources with a comparatively short T_w.

4.3.2 OSG Characterization

Currently, we are extending the work done with the XSEDE HPC resources to the OSG resources exposed via the XSEDE OSG Virtual Cluster. These OSG resources have distinctive characteristics that require tailored development and experimental investigation: (i) resources are hidden behind a broker so that the user does not have to directly connect to each resource scheduler; (ii) resources mainly support single-core executions; (iii) most of the resources do not guarantee a specified walltime, i.e., resources can become unavailable at any time during execution; (iv) each job should should run for no more than 12 hours; and (v) no limit is enforced on the number of concurrent jobs that can be submitted to the queue of the XSEDE OSG Virtual Cluster.

These characteristics introduce indeterminism in both the resource capabilities and availability, requiring the implementation of new systems to provision and manage resources and tasks execution. We enabled bulk submissions in RADICAL-SAGA and we extended the execution manager to handle task rescheduling and resource bookkeeping when one, more or all the resources suddenly become unavailable. Further, we refactored the RADICAL-Pilot task scheduler to handle high-rate scheduling across as many as 2,048 pilots. We partially implemented data staging—that requires the use of third-party systems to make files available to all the resources selected by the broker of the XSEDE OSG Virtual Cluster—and we have implemented a first prototype of a more general solution.

We focus our experiments on the dynamism and heterogeneity of OSG resources. We use Synapse [10] to emulate the execution of an AMBER simulation, as performed by the domain scientists of one of our Molecular Dynamic use cases. We execute between 8 and 2,048 emulations on between 8 and 2,048 pilots. According to the capabilities of the resources provided by the XSEDE OSG Virtual Cluster, each pilot has a single core and an indeterminate walltime. Each pilot is executed on a resource picked by the virtual cluster's broker, unknown in advance to the user. As a result, our emulations can be run on one or more resources with different capabilities and with a varying degree of concurrency, failure, and rescheduling.

Figure 6 shows some preliminary results. Figure 6a shows the multimodal and long tail distribution of the time taken by each emulation to be performed across the OSG resources. It should be noted that we have configured Synapse to always emulate the same amount of FLOPS. In this way, every difference in execution time can be explained by differences in the resource capabilities. The capabilities of the OSG resources offered via the XSEDE OSG Virtual Cluster are heterogeneous, possibly requiring the development of tailored scheduling heuristics in the AIMES middleware.

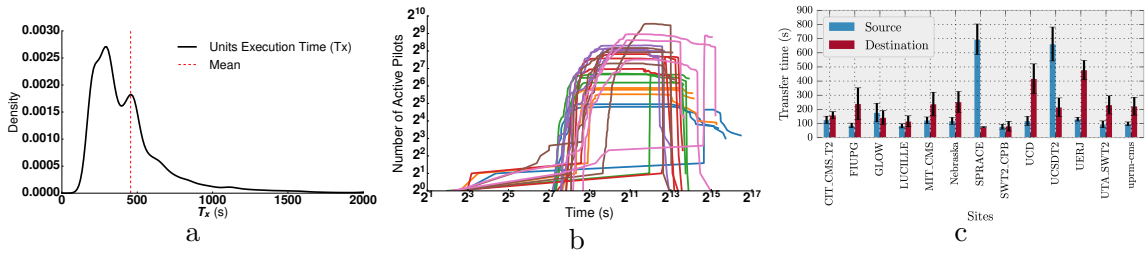

Figure 6: OSG has different types of heterogeneity and dynamism, including variation in number of resources, the availability of individual resources and the connectivity between them.

Figure 6b shows three examples of resource dynamism: (i) at different points in time, the XSEDE OSG Virtual Cluster offers different number and type of resources to execute the same workload; (ii) during runtime the number of resources available for execution varies (mostly between 2^2 and 2^8 cores); and (iii) during runtime the availability time of resources varies (mostly before 2^7 and after 2^{14} seconds.)

Figure 6c shows temporal and spatial heterogeneity for networking performance between different OSG resources. The "source" bars represents the time taken to transfer 1 GB of data from the given resource to the other twelve resources. The error bars represent the variation in time (i.e., different instances). The "destination" bars represent the time taken to move 1 GB to a given resource from the remaining 12 resources. In addition to the asymmetry between source and destination bars for any given resource, the fluctuations of the bars is typically different.

Dynamism and heterogeneity of resources require tailored resource and workload management capabilities, and specific heuristics for resource selection. We developed these capabilities for the AIMES middleware and our experiments showed how to collect the information to derive resource selection heuristics.

4.3.3 Swift Integration

As part of our research in design and development of middleware to enable extreme-scale computing, we explored the benefits of a "building-block" approach to engineering [11] the functional components of an integrated middleware. According to this approach, we engineered the AIMES middleware with independent, stand-alone modules that expose only well-defined interface. This facilitated the integration of our modules with existing tools, a particularly important requirement for any newly developed middleware when considering the body of work and tools already available.

We leveraged this opportunity by integrating the AIMES middleware with the Swift workflow system developed at Argonne and the University of Chicago. We developed an interface to enable the two systems to exchange task descriptions (Figure 7). The interface was implemented via a dedicated provider for Swift (Figure 7, AIMES provider) and a HTTP-based RESTful API. The use of REST helped to encapsulate the resource provisioning logic and to bridge any differences

between Java (Swift) and Python-based services (AIMES).

We designed three sets of experiments to compare distributed executions among Swift, AIMES, and their integration. These experiments allowed us to: (i) investigate alternative execution strategies to execute workloads of different sizes and types across multiple resources; (ii) compare the tradeoffs imposed by these execution strategies on TTC; (iii) outline how design features and configuration parameters enable execution strategies; and (iv) illustrate how the integration of AIMES and Swift supports profiling and emulation of real-life workflows on distributed and heterogeneous resources.

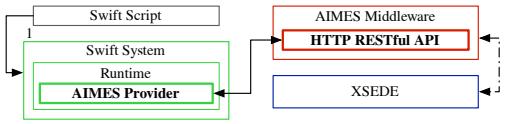

Figure 7: AIMES and Swift integrated architecture. Only the components developed for the integration are displayed.

Each experiment executed an increasingly large BoT or workflow on two to four XSEDE resources and NCSA's Blue Waters, depending on AIMES, Swift, or their integrated capabilities. We used task and pilot concurrently within and across resources, measuring realistic overheads and trade offs on production resources. In this way, our experiments were representative of the conditions under which users execute scientific workloads and workflows. We used Synapse to emulate a multi-stage molecular dynamics application [13] depicted in Figure 8.

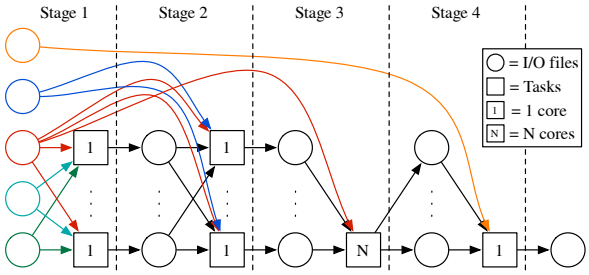

Figure 8: Molecular Dynamics workflow [12]. Stages 1 and 2 perform concurrent simulations and Stages 3 and 4 perform aggregated analysis.

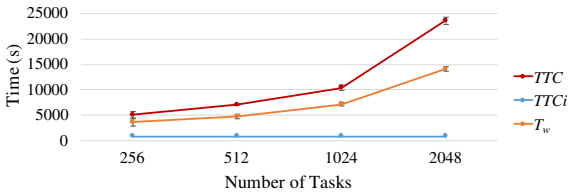

Figure 9: Average TTC and T_w for workflows (Figure 8) with 256, 1024, and 2048 simulations, executed via the AIMES and Swift integrated system on Stampede, Gordon, SuperMIC, Comet, and Blue Waters.

Figure 9 shows the results of the experiments performed with the integration among the two systems. Thanks to their joint capabilities, we were able to describe the workflow of Figure 8 as a Swift script, coordinate its execution via the Swift workflow engine, and leverage the late binding capabilities of RADICAL-Pilot to schedule tasks across multiple resources, including XSEDE and Blue Waters.

Figure 9 shows a progressive increase of both TTC and T_w averages. Our analysis shows that TTC increases mainly due to the time taken to stage input and output files between the user's workstation and the resource of the pilot on which each task is executed. While input and output files are on average few hundreds KB, their number increases with the scale of the workflow. The progressive increase of T_w is due to a corresponding increase in the pilots' average queue time. This likely depends on the size of the requested pilots: the larger the number of tasks of the workflow, the larger the size of each pilot on each resource.

Figure 9 also shows small error bars for both TTC and T_w. For TTC, this indicates that network latency, staging mechanisms, and the concurrency of task execution are consistent across executions, and that the execution strategy based on late-binding of tasks to pilots on multiple resources keeps

both the mean and variance of queue waiting time low.

5 What are the important questions that AIMES motivates us to ask?

Building upon advances arising from skeletons, Bundles and pilots, execution strategies allow qualitative reasoning about distributed workload execution that is resource and workload agnostic. There remains, however, a critical need for a quantitative basis for distinction between different possible decisions. For example:

- What are the constraints on the quantitative advantages arising from execution strategies? How are these dependent on workload properties? Or on resource availability? How does this translate into making "effective" scheduling decisions? How can we estimate different metrics or measures of performance?
- How sensitive is the planning and execution of complex workloads on the specifics of distributed infrastructure? How can different heterogeneous infrastructure be unified by a time-dependent capacity model that goes beyond point prediction models?
- How will we design the next DCI to meet the requirements of multiple science projects with predefined quality-of-service? Is it possible to derive models that allow us to "design infrastructure" for specific performance and requirements?

To answer these questions for current and future distributed computing systems requires new research along the following lines:

- From modeling to models: Models to answer the previously mentioned questions, if any, are currently restricted to specific application scenarios and infrastructure, and are not general purpose. Quantitative modeling of execution must include models of DCI resources, middleware, and applications. Planning and execution would be more precise with models of DCI than without, but such robust models do not exist yet. How do we build macroscopic models of application execution that are not unmanageably sensitive to microscopic heterogeneities? How would models that support design for specific performance generalize to design principles and architectures for next generation of distributed computing infrastructure? The answer to these questions cannot be derived from a single monolithic model, but require a hybrid set of models at multiple levels, different scales, and granularity. How are such models designed and composed? How can models support quantitative end-to-end reasoning?
- From sensing to actuation: We must transition from resource federation of known and predetermined resources to federation of resources that are discovered dynamically. How does the addition of resources that were not known at the time of initial planning change the course of execution? How do the kind of resources that the middleware "actuates" be influenced by application properties? These questions reiterate the need for quantitative models of applications, resources, and execution.
- Dynamism at extreme scale: Dynamism is going to become more pervasive, diverse, and significant as we move towards greater scale. For example, with better resource discovery

and new types of resources, there will be a larger number of resources resulting in greater variation in their temporal properties. Different classes of applications will result in more burstiness in application and data production. In other words, dynamism is set to become a dominant and first class property of both distributed applications and DCIs at larger scale. Inter alia, this will require better algorithms that are more responsive and better able to handle dynamism along the different dimensions. Last but not least, AIMES was designed upon the assumption that information could be dynamically integrated across levels. To what dynamic scales do these assumptions and approaches work? Answers to these questions require conceptual advances in the way we characterize dynamism in large-scale DCIs, as well as a radically different approach to responding to it.

Each of these research tracks and themes represents fundamental challenges as the community attempts to take distributed computing to the next scale. Each track encompasses a plethora of questions in turn. We hope to pursue these important questions as logical continuation of and within the powerful framework of AIMES.

6 Cost Status

During this project, we spent $137,233 at UChicago. Of this, about 75% was spent on personnel, specifically Zhao Zhang (a graduate student who developed the skeleton tool), Yadu Nand Babuji (a staff member who supported the skeleton tool after Zhang graduated and worked on integrated, project-wide testing). and Mike Wilde (the project lead for Swift and Yadu's supervisor, who also provided information about how to best use OSG for integrated testing.) Note Katz's salary was 100% covered by NSF while he was at the Foundation, though he spent 10–15% of his time on this project, so the fraction of the total project resources that was personnel was actually much larger than the fraction covered under the DOE funds.

About 15% was spent on travel for project meetings and conferences. The remaining 10% was spent on supplies, services, and other miscellaneous costs, including tuition remission for Zhang.

The project was fully spent out.

7 Administrative and Organizational Reporting

7.1 Publications

1. Shantenu Jha, Andre Merzky, Matteo Turilli, "Towards End-to-End Integrated Modeling of Extreme Scale Systems". Workshop on Modeling & Simulation of Exascale Systems & Applications, 2013. (Ref. [14])

2. Shantenu Jha, Andre Merzky, Matteo Turilli, Daniel S. Katz and Jon Weissman, "Towards Distributed Exascale Computing: An AIMES Perspective". Workshop on Modeling & Simulation of Exascale Systems & Applications, 2014. (Ref. [15])

3. Andre Luckow, Mark Santcroos, Ole Weidner, Ashley Zebrowski, Shantenu Jha, "Pilot-Data: An Abstraction for Distributed Data". Journal Parallel and Distributed Computing, October 2014. doi: 10.1016/j.jpdc.2014.09.009 (Ref. [5])

4. Zhao Zhang and Daniel S. Katz, "Using Application Skeletons to Improve eScience Infrastructure". Proceedings of the IEEE 10th International Conference on e-Science, 2014. doi: 10.1109/eScience.2014.9 (Ref. [9])

5. Daniel S. Katz, Andre Merzky, Zhao. Zhang, Shantenu Jha, "Application Skeletons: Construction and Use in eScience". Future Generation Computing Systems, v.59, pp. 114–124, 2016. doi: 10.1016/j.future.2015.10.001 (Ref. [2])

6. Feng Liu, Jon B. Weissman, "Elastic job bundling: an adaptive resource request strategy for large-scale parallel applications". SC '15 Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, November 2015. (Ref. [16])

7. Matteo Turilli, Francis Liu, Zhao Zhang, Andre Merzky, Micheal Wilde, Jon Weissman, Daniel S. Katz, Shantenu Jha, "Integrating Abstractions to Enhance the Execution of Distributed Applications". Proceedings of the 30th IEEE International Parallel & Distributed Processing Symposium (IPDPS), 2016. doi: 10.1109/IPDPS.2016.64 (Ref. [7])

8. Shantenu Jha, Daniel S. Katz, Andre Luckow, Omer Rana, Yogesh Simmhan, Neil Chue Hong, "Introducing Distributed Dynamic Data-intensive (D3) Science: Understanding Applications and Infrastructure". Journal of Concurrency and Computation: Practice and Experience (CCPE), in press, 2016. (Ref. [17])

Several other publication are under review.

1. Matteo Turilli, Mark Santcroos, Shantenu Jha, "A Comprehensive Perspective on Pilot-Jobs", 2015. `http://arxiv.org/abs/1508.04180`

2. Matteo Turilli, Yadu Nand Babuji, Andre Merzky, Ming Tai Ha, Michael Wilde, Daniel S. Katz, Shantenu Jha, "Analysis of Distributed Execution of Workloads", 2016. `http://arxiv.org/pdf/1605.09513v1.pdf`

3. Shantenu Jha and Matteo Turilli, "A Building Blocks Approach towards Domain Specific Workflow Systems?", 2016. `https://arxiv.org/abs/1609.03484`

Peer-reviewed Software

1. Zhao Zhang, Daniel S. Katz, Andre Merzky, Matteo Turilli, Shantenu Jha, Yadu Nand Babuji, "Application Skeleton: Generating Synthetic Applications for Infrastructure Research". Journal of Open Source Software, 2016. doi: 10.21105/joss.00017 (Ref. [18])

White Papers

1. Shantenu Jha, Daniel S. Katz, Matteo Turilli, and Jon Weissman, "From Abstractions to MODELS: MOdels for Distributed and Extremely Large-scale Science". Zenodo, 2016. doi:10.5281/zenodo.44236 (Ref. [19])

Student work towards degrees:

1. Zhao Zhang, Doctor of Philosophy in Computer Science (June 2014), University of Chicago: "Enabling Efficient Parallel Scripting on Large-Scale Computers".

7.2 Technologies/Techniques

- Skeleton: https://github.com/applicationskeleton/Skeleton
  Version 1.2 has been archived as https://doi.org/10.5281/zenodo.13750
- Synapse: https://github.com/radical-cybertools/radical.synapse
- Emaneger: https://github.com/radical-cybertools/aimes.emgr
- Bundle: https://github.com/radical-cybertools/aimes.bundle
- RADICAL-Pilot: https://github.com/radical-cybertools/radical.pilot
- RADICAL-SAGA: https://github.com/radical-cybertools/saga-python

Experiments and experimental data:

- Experiments on XSEDE HPC resources: https://bitbucket.org/radical-project/aimes.experiments
- Experiments on XSEDE OSG Virtual Cluster: https://github.com/radical-experiments/AIMES-Experience
- Experiments on AIMES/Swift integration: https://github.com/radical-experiments/AIMES-Swift
- Experiments on XSEDE and XDMoD: https://github.com/radical-experiments/AIMES-XDMoD
- Experiments on Synapse: https://github.com/radical-experiments/radical.synapse

References

[1] Jha S, Cole M, Katz DS, Parashar M, Rana O, Weissman J. Distributed computing practice for large-scale science and engineering applications. Concurrency and Computation: Practice and Experience. 2013;25(11):1559–1585. https://doi.org/10.1002/cpe.2897.

[2] Katz DS, Merzky A, Zhang Z, Jha S. Application skeletons: Construction and use in eScience. Future Generation Computer Systems. 2015;https://doi.org/10.1016/j.future.2015.10.001.

[3] Luckow A, Santcroos M, Merzky A, Weidner O, Mantha P, Jha S. P*: A model of pilot-abstractions. IEEE 8th International Conference on e-Science. 2012;p. 1–10. https://doi.org/10.1109/eScience.2012.6404423.

[4] Turilli M, Santcroos M, Jha S. A Comprehensive Perspective on Pilot-Jobs; 2016. (under review)
http://arxiv.org/abs/1508.04180.

[5] Luckow A, Santcroos M, Zebrowski A, Jha S. Pilot-Data: An Abstraction for Distributed Data. Journal Parallel and Distributed Computing. 2014 October;
https://doi.org/10.1016/j.jpdc.2014.09.009.

[6] Luckow A, Paraskevakos I, Jha S. Pilot-Abstraction: A Valid Abstraction for Data-Intensive Application on HPC, Hadoop and Cloud Infrastructures?; 2015. http://arxiv.org/pdf/1501.05041v1.pdf.

[7] Turilli M, Liu FF, Zhang Z, Merzky A, Wilde M, Weissman J, et al. Integrating Abstractions to Enhance the Execution of Distributed Applications. In: Proceedings of 30th IEEE International Parallel and Distributed Processing Symposium (IPDPS); 2016. p. 953–962. https://doi.org/10.1109/IPDPS.2016.64.

[8] Radak BK, Romanus M, Lee TS, Chen H, Huang M, Treikalis A, et al. Characterization of the Three-Dimensional Free Energy Manifold for the Uracil Ribonucleoside from Asynchronous Replica Exchange Simulations. Journal of Chemical Theory and Computation. 2015;11(2):373–377. Available from: https://doi.org/10.1021/ct500776j.

[9] Zhang Z, Katz DS. Using Application Skeletons to Improve eScience Infrastructure. In: Proceedings of the 2014 IEEE 10th International Conference on e-Science - Volume 01. E-SCIENCE '14. Washington, DC, USA: IEEE Computer Society; 2014. p. 111–118. Available from: https://doi.org/10.1109/eScience.2014.9.

[10] Merzky A, Jha S. Synapse: Synthetic Application Profiler and Emulator. In: 2016 IEEE International Parallel and Distributed Processing Symposium Workshops, IPDPS Workshops 2016, Chicago, IL, USA, May 23-27, 2016; 2016. p. 1259–1268. Available from: https://doi.org/10.1109/IPDPSW.2016.168.

[11] Jha S, Turilli M. A Building Blocks Approach towards Domain Specific Workflow Systems?; 2016. (under review) http://arxiv.org/abs/1609.03484.

[12] Balasubramanian V, Trekalis A, Weidner O, Jha S. Ensemble toolkit: Scalable and flexible execution of ensembles of tasks. Accepted for ICPP 2016 (Philadelphia), available as arXiv:160200678v2. 2016;.

[13] Balasubramanian V, Bethune I, Shkurti A, Breitmoser E, Hruska E, Clementi C, et al. ExTASY: Scalable and Flexible Coupling of MD Simulations and Advanced Sampling Techniques. In: Accepted for IEEE International Conference on eScience; 2016. https://arxiv.org/abs/1606.00093.

[14] Jha S, Merzky A, Turilli M, Katz DS, Weissman J. Distributed Exascale Computing: An AIMES Perspective; 2013. http://hpc.pnl.gov/modsim/2013/.

[15] Jha S, Merzky A, Turilli M. Modeling Distributed Extreme-Scale Applications and Systems; 2014. Http://hpc.pnl.gov/modsim/2014/.

[16] Liu F, Weissman JB. Elastic Job Bundling: An Adaptive Resource Request Strategy for Large-scale Parallel Applications. In: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. SC '15. New York, NY, USA: ACM; 2015. p. 33:1–33:12. Available from: http://doi.acm.org/10.1145/2807591.2807610.

[17] Jha S, Katz DS, Hong NC, Luckow A, Rana O, Simmhan Y. Introducing Distributed Dynamic Data-intensive (D3) Science: Understanding Applications and Infrastructure. Concurrency and Computation: Practice and Experience. 2016;Available from: https://arxiv.org/pdf/1609.03647.pdf.

[18] Zhang Z, Katz DS, Merzky A, Turilli M, Jha S, Nand Y. Application Skeleton: Generating Synthetic Applications for Infrastructure Research. The Journal of Open Source Software. 2016 May;1(1). Available from: https://doi.org/10.21105/joss.00017.

[19] Jha S, Katz DS, Turilli M, Weissman J. From Abstractions to MODELS: MOdels for Distributed and Extremely Large-scale Science; 2015. Available from https://doi.org/10.5281/zenodo.44236/.