Integrating Abstractions to Enhance the Execution of Distributed Applications

Matteo Turilli
RADICAL Laboratory, ECE
Rutgers University
New Brunswick, NJ, USA

Feng (Francis) Liu
Computer Science and Engineering Department
University of Minnesota
Minneapolis, MN, USA

Zhao Zhang
AMPLab
University of California
Berkeley, CA, USA

Andre Merzky
RADICAL Laboratory, ECE
Rutgers University
New Brunswick, NJ, USA

Michael Wilde
Computational Institute
University of Chicago &
Argonne National Laboratory
Chicago, IL, USA

Jon Weissman
Computer Science and Engineering Department
University of Minnesota
Minneapolis, MN, USA

Daniel S. Katz
Computational Institute
University of Chicago &
Argonne National Laboratory
Chicago, IL, USA

Shantenu Jha[*]
RADICAL Laboratory, ECE
Rutgers University
New Brunswick, NJ, USA

ABSTRACT

The number of distributed applications that can utilize multiple distributed resources in a simple and scalable manner is still limited. This is a representation of the state-of-the-practice of large-scale distributed computing as well a product of the lack of abstractions for distributed applications and infrastructure. This paper presents three new abstractions and integrates them within a pilot-based middleware framework. This integrated middleware supports the execution of large-scale distributed applications on multiple, production-grade, and heterogeneous resources and enables extensive experimentation on the execution process itself. Thousands of experiments have been performed on XSEDE and NERSC production resources. These experiments offer valuable and actionable insights on how distributed applications should be coupled with multiple heterogeneous resources. This paper provides conceptual understanding and practical guidance into the improvement of the current generation of distributed application, as well as the design of the next-generation of distributed infrastructures by: (1) providing a virtual laboratory for a structured, semi-empirical study of executing dynamic applications on production-grade dynamic resources, and (2) understanding the impact of heterogeneity, scale and degree of distribution on the execution of distributed applications.

[*]Corresponding author

Categories and Subject Descriptors

C.2.4 [**Computer Systems Organization**]: Distributed Systems—*distributed applications*; C.4 [**Computer Systems Organization**]: Performance of Systems

General Terms

Experimentation, Theory

Keywords

abstractions, middleware, execution strategies, distributed systems

1. INTRODUCTION

The need for the collective utilization of multiple, distributed and heterogeneous resources is common to large-scale science projects [1, 2] as well as to the long-tail of science [3]. While impressive strides have been made in the use of individual HPC resources, the ability to utilize them for collective high-performance and distributed computing (HPDC) [4] is at best limited. [5]

Currently, the most relevant challenges are developing distributed applications as well as designing large-scale distributed computing infrastructure (DCI) to support them. A few simple classes of distributed applications are supported (e.g. embarrassingly parallel,) yet even these applications are not always easy to develop or execute across a DCI. This is due, at least in part, to a lack of abstractions that generalize and uniformly represent DCI well enough that application developers can build more sophisticated and portable applications. As a consequence of being dependent upon local and point solutions, distributed application are often brittle. It is thus no surprise that the ability to reason about the execution of distributed applications on DCI is limited. For example: If an application has a certain characteristic, what type of infrastructure should it use? What time-to-completion should be expected if an application had the collective power of XSEDE [6] available to it? How do specific execution decisions influence the performance? Given a specific infrastructure available, how should an application adapt to best use it?

While the ability to reason about alternative execution approaches and performance trade-offs among different execution possibilities is a desired goal, there is first a need to improve basic execution capabilities across multiple distributed resources. Enhancements in execution capabilities need to be quantitative (e.g., number of heterogeneous resources that can be utilized and of applications that can be executed) as well as qualitative (e.g., resource abstractions and unifying models of resource management). Progress has been made in addressing the heterogeneity of the resource access layer, but the lack of well established abstractions and approaches to resource management bind applications to specific platforms, acting as a barrier to interoperability.

In this paper, we systematically examine the technical issues of the above limitations. We address them by means of a powerful and minimally complete set of abstractions. These abstractions span multiple levels: application (skeletons), resources and resource level information (bundles), and dynamic resource management (pilots). With these abstractions in place, and motivated by improving the scale and state-of-the-practice of distributed applications execution, we formalize the set of decisions needed for such an execution in the form of a fourth abstraction: Execution Strategy.

We implement these abstractions to build a general purpose and extensible middleware, so as to investigate experimentally the characteristics of the interoperable execution of distributed applications. Insight into the consequences of different execution strategies, the range of application and resource characteristics investigated, and the scale of the experiments permit us to claim the abstractions as valid instruments of intuition, experimentation, and effective execution of distributed applications.

Experiments using large-scale, production grade, distributed, and heterogeneous resources (up to five different concurrent machines) were performed over the course of almost a year. Our experiments have been repeated more than 20,000 times for different distributed applications; the total number of tasks executed is more than 10 million. The range and scale of these experiments enable us to measure advantages in specific execution strategies and quantify their impact as a function of scale, for both applications and resources. This is an important and non-trivial advance towards the ability to reason about and to discern the consequence of distributed application execution decisions. The ability to analyze and determine optimal execution strategies will ultimately yield quantitative models of distributed execution [7].

This work also establishes that simple execution strategies and scalability are not necessarily mutually exclusive. Engineered implementations of well-defined conceptual models (abstractions) can support both semi-empirical and hypothesis-driven investigation at the scale and at levels of sophistication needed to capture the requirements of existing large-scale, distributed applications [8]. This work takes an important step in moving beyond the traditional limitations of reasoning about distributed execution. It also provides empirical evidence of the ability to design and architect distributed infrastructures for specific performance and resource properties. Put together, this work advances the state-of-the-practice of distributed application execution, extending its impact from the formal realm into the very practice of large-scale distributed computing for science and engineering [9].

In Section 2, we provide a brief summary of precursory and related work. Section 3 discusses the abstractions we have defined and integrated. The architecture and resultant capabilities of integrating the abstractions is presented in Section 4, along with a detailed discussion of experimental design. Section 4 also discusses the results of the core experiments of this paper and an analysis of the results. Section 5 discusses the implications on the state-of-the-practice, implications for the future of distributed application execution and its possible challenges.

2. RELATED WORK

This work was conceived following the comprehensive survey of distributed applications [9]. The survey established, arguably for the first time, the relationship between infrastructure and scientific distributed applications. It examined the issue of inconsistent internal and external interfaces and how they lead to application brittleness.

The challenge of distributed application execution on heterogeneous resources is a well-known and extensively researched problem. The I-Way [10], Legion [11], and Globus [12, 13] projects all integrated existing tools to use multiple resources to run distributed scientific applications. However, their successes also exposed some of their core limitations. Mandatory deployment of homogeneous middleware on all resources and the complexity of porting user-facing distributed applications greatly limited adoption and usability. Success stories such as described by Allen et al. [14] clearly show the complex process of porting a distributed application to a framework in order to use distributed resources.

The abstractions and tools described in this paper avoid traditional limitations by: (1) addressing heterogeneity with interoperability rather than requiring homogeneous middleware on each target resource; (2) avoiding the assumption that the resource and application layer are static across the execution process; and (3) by minimizing the porting and execution overheads of end user applications. We believe the execution strategy, skeleton, and bundle abstractions are novel, though related work has been performed.

Bokhari [15] and Fernandez-Baca [16] theoretically proved the NP completeness of matching/scheduling components on distributed systems. Braun et al. [17], Chen and Deelman [18], and Malawski et al. [19] present execution strategies modeled as heuristics based on empirical experimentation. Workflow systems like Kepler [20], Swift [21], and Taverna [22] all implement execution management but in the form of a single point solution, specifically tailored to their execution models. Our execution strategy abstraction instead isolates, generalizes, and makes explicit the representation and mechanics of controlling the coupling of a given set of tasks over diverse resources.

Work from Foster and Stevens [23], Meyer et al. [24], Logan et al. [25] resemble some of the features of skeletons but are limited only to parallel applications or directed acyclic graphs (DAGs). Bundles leverages some of the work done in information collection [26], resource discovery [27, 28], and resource characterization as it relates to queue time prediction [29, 30, 31, 32, 33] and its difficulties [34].

3. DISTRIBUTED SYSTEM ABSTRACTIONS

A fundamental attribute of DCI is the spatiotemporal fluctuation of the available capacity of resources. Application execution therefore takes place on a dynamically varying resource landscape. The coupling between the application and the resources will have at least one time varying component, and where the application workload is changing, possibly both. As decisions influencing execution are dependent upon changing state information, the coupling between the application layer and the infrastructure layer must move beyond static interoperability. In fact, execution decisions are intimately related to the type of information used, and for distributed applications this information has to be drawn both from the application level as well as from the infrastructure level.

The primary objective of integrated middleware is to bring together application-level and infrastructure-level information. This is non-trivial; not only is the type and reliability of resource information difficult to normalize, but the application requirements are also diverse. Given the broad range of application characteristics and resource types, we want to abstract a core set of distributed application properties and capture how they are related to resource layer properties, and vice versa. Abstractions that are usable for a wide range of application characteristics and resource types also need to be extensible and able to be integrated together. We are faced with the classic design tension of cohesion (narrowly defined abstractions) versus coupling (integration).

Although the full complexity of working in large-scale production environments cannot be reduced to the sum of the complexities of the constituent components, simplifying those components is an important first step. We use four abstractions to support the dynamic execution of distributed applications on diverse resources. Our work provides a minimally complete set of abstractions and more: it provides a laboratory for testing ideas “in the wild”, i.e., in the full complexity of **production** distributed environments.

3.1 Application Abstractions

Most distributed applications that we have observed are of two types. The first is Many-Task Computing (MTC) [35], where applications are composed of tasks, which themselves are executables. The tasks can be thought of as having a simple structure from the outside: they read files, compute, and write files, though they can be quite complex internally. They can also be either sequential or parallel. These MTC applications fall into a small number of types, specifically bag-of-task, (iterative) map-reduce, and (iterative) multistage workflow. Interaction between tasks is usually of the form of files produced by one task and consumed by another. The tasks can be distributed across resources, and there is a framework that is responsible for launching the tasks and moving the files as needed to allow the work to be done.

The second type is applications composed of distributed elements that interact in a more complex manner, such as by exchanging messages while running, possibly as services. The elements of these applications can have persistent state, while the MTC tasks do not; their outputs are based solely on their inputs, and they are basically idempotent.

The majority of distributed science and engineering applications are MTC, while in business, there is much more of a mix of the types. Because we are concerned with science and engineering application, we currently focus on MTC applications. We generalize bag-of-task, (iterative) map-reduce, and (iterative) multistage workflow applications into (iterative) multistage workflow applications, since bag-of-task applications are basically single-stage applications and map-reduce applications are basically two-stage applications.

We use a top-down approach to abstract the application: an application is composed by a number of stages (which can be iterated in groups), and each stage has a number of tasks. An application is described by specifying the number of stages and the number of tasks, input and output file and task mapping, task length, and file size inside each stage. Task lengths and file sizes can be statistical distributions or a polynomial functions of other parameters. For example, input file size can be a normal distribution, task length can be a linear function of input file size, and output size can be a binomial function of task runtime.

We call this type of abstract application a ‘Skeleton Application’ [36], and have built an open source tool (Application Skeleton [37]) that can create skeletons. The tool is implemented as a parser. It reads in a configuration file that specifies a skeleton application, and produces three groups of outputs: (1) **Preparation Scripts**: the preparation scripts are run to produce the input/output directories and input files for the skeleton application. (2) **Executables**: executables are the actual tasks of each application stage. We assume different stages use different executables. (3) **Application**: the overall skeleton application can be implemented in one of four formats: shell commands that can be executed in sequential order on a single machine, a Pegasus DAG [38] or a Swift script [21] that can be executed on a local machine or in a distributed environment, or a JSON structure that must be used by middleware that is designed to read it, in our case, the integrated middleware described in Section 4.

The application skeleton tool itself can be called by a user from the command line, or through an API. The work in this paper uses the skeleton API to call the skeleton code. To execute a skeleton application, the preparation scripts are run to create the initial input data files, then the skeleton application itself is run (the DAG with Pegasus, the Swift script with Swift, the Shell script with Bash, or the JSON structure with other software that can use it). The executables produced by the skeleton tool as the tasks for each stage copy the input files from the file system to RAM, sleep for some amount of time (specified as the runtime), and copy the output files from RAM to the file system.

We have previously [36] tested the performance accuracy of the skeleton applications. We profiled three representative distributed applications—Montage [39], BLAST [40], CyberShake-postprocessing [41]—to understand their computation and I/O behavior and then derived parameters required to specify the skeletons. We showed that the application skeleton tool produced skeleton applications that correctly captured important distributed properties of real applications but were much simpler to define and use. Performance of the real application vs the skeleton applications showed overall errors of -1.3%, 1.5%, and 2.4% for Montage, BLAST, and CyberShake PostProcessing, respectively. At the stage level, fourteen out of fifteen stages had errors of less than 3%, ten had errors less than 2%, and four had errors of less than 1%.

3.2 Resource Abstractions

Very few scientific applications run on dedicated resources owned by the end user. As a consequence, most users of scientific applications have to deal with the challenges of sharing heterogeneous resources that have dynamic availability and performance. This introduces complexity for both the application writer and end user. We mitigate this complexity with a new resource abstraction.

Resource allocation for scientific applications can be either static or dynamic. Allocation is static when users select resources based on knowledge of capacity, performance, policy, and cost. Often, this decision is made on an ad hoc basis. Allocation is dynamic when users monitor resource status during application execution and adjust resources as needed. Typically, this is beyond user ability and resource capability. Both static and dynamic resource allocation require accurate resource characterization but we have observed that despite its importance, systematic approaches to address the need for such a characterization are lacking.

We have developed an application-centered resource abstraction layer exposed by a uniform interface. This layer functions as the bridge between applications and compute infrastructures providing accurate resource characterizations. In this way, efficient resource selection and usage by scientific applications is facilitated. We call this layer the 'Bundle Abstraction', or simply the 'Bundle'.

The bundle abstraction has been implemented as a *resource bundle*. A resource bundle may be thought of as an instance of a bundle representing some portion of system resources. A resource bundle may contain an arbitrary number of resource categories (e.g., compute, storage) but it does not 'own' the resources. In this way, a resource may be shared across multiple bundles and users can be provided with a convenient handle for performing aggregated operations such as querying and monitoring.

A resource bundle has two components: *resource representation* and *resource interface*. The resource representation characterizes heterogeneous resources with a large degree of uniformity, thus hiding complexity. Currently, the resource bundle models resources across three basic categories: compute (including memory), network, and storage. Resource measures that are meaningful across multiple platforms are identified in each category. For example, the property 'setup time' of a compute resource means queue wait time on a HPC cluster or virtual machine startup latency in a cloud [42]. Similarly, processors can refer to either actual cores or virtual machines.

The resource interface exposes deeper characterizations of the resource by measuring how it has been used in the past and provides universal interfaces that hide the unnecessary complexities within each resource. Three types of resource interfaces are exposed: querying, monitoring, and discovery. The query interface uses end-to-end measurements to represent and organize resource information. For example, the query interface can be used to inquire how long it would take to transfer a file from one location to a resource and vice versa.

The resource interface supports queries both in an on-demand mode (i.e., right now) and in a predictive mode using a data-driven approach. Our predictive capability is based on a characterization of the resource in terms of predicted workload and utilization as opposed to accurate queue waiting time prediction, which is extremely hard to do accurately [34].

The resource bundle monitoring interface allows callers to inquire about resource state and to chose system events for which to receive notification via an asynchronous callback mechanism. For example, performance variation within a loosely-coupled cluster can be monitored so that when the average performance has dropped below a certain threshold for a certain period, subscribers of such an event will be notified. The callback mechanism may trigger subsequent scheduling decisions such as adding more resources to the application.

The resource discovery interface, which is future work, will allow the user to request resources based on abstract requirements so that a tailored resource bundle can be dynamically created. A simple language for specifying resource requirements that will allow compute, storage, and network requirements to be specified is being developed. This concept has been shown to be successful for cloud storage through Tiera [43], where resource capacities and resource policies are specified in a compact notation. In this way, there will be two ways to create a resource bundle: by explicitly adding resources to it, and by defining resource requirements and allowing the system to discover and assemble an appropriate set of resources to be contained within it.

3.3 Dynamic Resource Abstractions

The pilot abstraction [44] generalizes the common concept of a resource placeholder, as a pilot holds a portion of the resources of a system so that the user can utilize it to execute compute tasks. A pilot is submitted to the scheduler of the target system and, once active, it allows for the direct submission of compute tasks by the user. In this way, the user executes compute tasks within the time and space boundaries set by the system scheduler for the pilot while avoiding the scheduler overhead for each compute task. Usually, the benefits of an implementation of a pilot system are measured in terms of (single core) task throughput but, within our work, the pilot abstraction has been leveraged more broadly to achieve heterogeneous resource aggregation and to investigate and measure optimal distributed application execution.

Implementations of the pilot abstraction have proven very successful at supporting computationally intense scientific research, especially when requiring the execution of large scale, single/low cores tasks. For example, the ATLAS project [45] leverages a pilot system integrated within the Production and Distributed Analysis system (PanDA) to execute the single-task portion of the 5 million jobs processed by the project every week [46]. Analogously, the middleware deployed by the Open Science Grid (OSG) [47] project uses a pilot system named 'Glidein' to provide more than 700 million CPU hours a year, mostly to applications requiring high-throughput of single-core tasks [48]. Furthermore, many leading user-facing tools and frameworks implement the pilot abstraction to support the execution of user-specified workflows. The Swift runtime system implements pilots in a subsystem named 'Coasters' [49] while the FireWorks and Pegasus workflow systems use, respectively, subsystems named ‘Rockets’ [50] and ‘Corral’ [38].

Generally, the available pilot implementations suffer from two main limitations: (1) they are either coupled to a resource-specific middleware or part of a vertical application framework; and (2) they do not fully implement the richness of the pilot abstraction as defined, for example, in Ref [44]. In order to avoid these limitations a stand-alone pilot system called ‘RADICAL pilot’ was implemented. RADICAL pilot does not need to be deployed and managed by the target resources, exposes a well-defined programming interface to user-facing tools, and thanks to RADICAL SAGA [51] (the reference implementation of the SAGA OGF standard [52]) allows for enacting pilots and executing compute and data tasks on diverse resources.

RADICAL pilot is designed to be both a production grade tool to execute scientific applications and a full-fledged experimental toolkit. An explicit state model and a set of timers and introspection tools can be used to record the performance of each state transition and the state properties of each RADICAL pilot component. To the best of the authors’ knowledge, these capabilities are missing in the pilot systems currently available but they are fundamental for enabling users to tailor effectively and efficiently the executions of distributed applications to their use cases.

The Pilot abstraction and its pilot system implementations help to highlight the general issue of coupling the application requirements to suitable target resources. This issue needs to be addressed whenever a pool of physical or logical resources have to be acquired to execute a distributed application, independently from the use of a pilot system. For example, jobs are used to acquire resources on a HPC or grid system, while virtual machines are used in a cloud infrastructure context. The *Execution Strategy* abstraction was defined and then implemented to address this application requirements/resource capabilities coupling challenge.

3.4 Distributed Execution Abstractions

The coupling process of distributed applications and resources involves several decisions to be made in order to target the correct amount and type of resources for the required period of time. For example, when targeting multi-user HPC resources, the user has to choose which queue to use, how many cores to request, for how long such cores will be needed, and where the data should be read from and written to. In the presence of multiple resources, the user may want to choose whether to target one or more resources, whether to use the resource with the lightest load or the largest amount of cores, memory, disk space, or bandwidth available. Broadly speaking, these choices depend upon the characteristics of the resources, on the requirements of the application, and on how the former can satisfy the latter. We refer to the set of all the possible choices associated with the execution of a given distributed application on a set of target resources as the ‘Execution Strategy’. This is traditionally determined based on user knowledge and experience, often through unstated heuristics.

An execution strategy can be thought of as a tree where each type of choice is a vertex and each edge is a dependence relation among types of choice. In the simplest case, there is only one value for each type of choice that allows for the application to be executed. For example, given a single resource, with a single queue, an application with a predefined number of tasks of known duration, and single-level scheduling (i.e., no pilots) there will be only one execution strategy available to the user. The number of values available for each type of choice increases when two or more resources can be used, when the duration for the tasks of the application is less precisely known, or when multi-level scheduling is available allowing for different binding times. In this case, the user will have the opportunity to target a different resource or to select different values for the walltime of each task. Our goal is to offer an alternative view: an explicit, scientific method of determining the execution strategy to enhance the user control over the optimization process and tailoring of her application execution.

Note that every execution strategy available for a given application is guaranteed to allow for its execution. The selection among alternative strategies is relevant only when considering multiple types of metrics. For example, execution strategies may differ in terms of time-to-completion (TTC), maximum throughput, energy consumption, affinity to specific resources, or allocation consumption. The relevance of each metric depends on qualitative requirements of the user and their quantification on the type of target resource(s) and given application. TTC is arguably one of the most relevant metrics; part of our work was improving distributed application execution by experimenting whether, how, and why alternative execution strategies lead to different TTC for the same application.

The execution strategy abstraction has been implemented into a software module called ‘Execution Manager’. This module interfaces with the implementation of all the described abstractions. It is therefore specialized to execute skeleton applications by means of pilots instantiated on resources chosen leveraging bundles. Nonetheless, the execution strategy abstraction is not intrinsically dependent on the pilot, skeleton, or bundles abstraction as the type and values of its decision points are not predetermined.

The strategy derivation process has five steps: (1) information is gathered about an application via the Skeleton API and about available resources via the Bundle API; (2) the application requirements and the spatiotemporal properties of the resources needed to execute it are derived from the skeleton description; (3) a suitable set of target resources is chosen by evaluating information provided by the bundle module; (4) a set of pilots is described and then instantiated on the chosen resources; and (5) the strategy is enacted by executing the application on the instantiated pilots.

The choices made in steps 2 and 3 depend on whether an optimization metric is given to the execution manager. For example, given a distributed application composed of independent tasks and the TTC metric, the execution manager will select a set of resources that allow for maximal execution concurrency, to minimize the execution time. Similar decisions will be made depending on the amount of data that needs to be staged, the bandwidth available between resources, the set of data dependences, etc. Note that this type of optimization leverages heuristics based on semi-empirical data. For example, knowing whether a bag of 2,048 single-core, loosely-coupled tasks, with a heterogeneous number of cores, duration, and input/output data should be executed on a single large pilot or on three smaller pilots instantiated on different resources, depends on algorithmic considerations as much as on empirical data about the behavior of pilots and resources under these conditions. This is why improving the practice of distributed computing execution requires both theoretical and empirical insights.

4. INTEGRATING DISTRIBUTED COMPUTING ABSTRACTIONS IN MIDDLEWARE

The four abstractions and their implementations presented in Section 3 help to address at least three of the problems that beset the practice of executing distributed applications: interoperability, scalability, and adaptivity. Interoperability here refers to abstracting localized differences in the resource and application layer to consistent and unified representations. Interoperability supports scalability, the possibility to execute an increasingly large number of loosely coupled tasks of a distributed application across multiple, heterogeneous resources. Adaptability consists of tailoring the execution process of specific distributed applications to the acquisition and utilization of diverse resources. As such, it is one of the core contribution of this paper to the practice of executing distributed applications.

Skeleton and bundle, which implement skeleton applications and resource bundles respectively, support interoperability by describing application properties and providing current and consistent status and behavior information about the resources. Pilot, implemented as RADICAL pilot, promotes scalability by aggregating heterogeneous resources across different administrative domains. Finally, execution strategy, implemented in an execution manager, enables adaptability by tailoring the execution of the distributed application to the capabilities of each target resource.

Skeleton applications, resource bundles, RADICAL pilot, and the execution manager have been implemented as Python modules. Each module exposes a well-defined API to promote and facilitate integration and, together, these four modules constitute a prototype that we call the AIMES (Abstractions and Integrated Middleware for Extreme-Scales) middleware (see Figure 1.) The AIMES middleware is extensible; different modules can be integrated when new functionality is needed. For example, as seen in Section 3, RADICAL SAGA has been integrated to provide low-level resource interoperability, and new application modules are being developed to support the execution of diverse applications and workflows. The AIMES middleware is also self-contained; its capabilities depend exclusively upon its modules and not on further components that need to be deployed into the application or resource layers.

As discussed in Section 1, improving the practice of executing distributed applications requires technological solutions fostered by a qualitative and quantitative understanding of the distributed execution process. For this reason, the AIMES middleware has been developed both to support the execution of large scale scientific workloads by enabling interoperability, scalability, and adaptivity, but also as an experimental platform to explore and measure the underlying issue of the dynamic coupling of application requirements to resource capabilities. A set of experiments has been conducted to isolate the parameters whose values change depending on (1) scale; (2) concurrency; and (3) distribution of resources, pilots, and application tasks.

Scale is both temporal and spatial, e.g., the amount of time and resources needed to execute applications. The same applies to pilots and resources when considering their quantity, size, and duration. Concurrency indicates tempo-

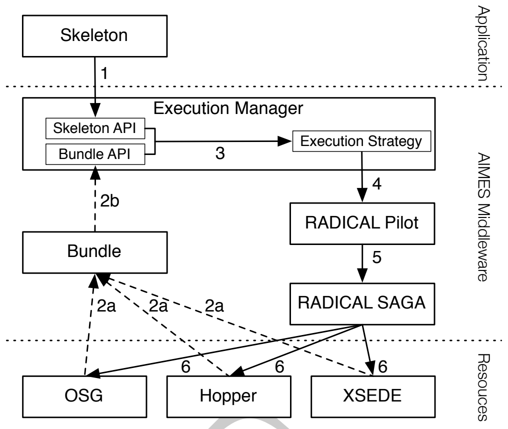

Figure 1: High-level architecture of the AIMES middleware. (1) provides distributed application description; (2) provides resource properties and state; (3) derives execution strategy; (4-6) enacts execution strategy.

ral overlapping and is a function of the type of tasks that need to be executed, the quantity of resources that are available at a given point of time, and the size, number and duration of the available pilots. Distribution[1] identifies the type of variation of the values of a parameter, for example, the uniform or Gaussian distribution of the durations of a set of tasks or the distribution of the size of a set of pilots.

The evolution of the values of scale, concurrency, and distribution over time is a measurable representation of the dynamical behavior of applications, resources and, more importantly, of their coupling. The experiments presented in this section measure specific characteristics of this behavior in order to advance our understanding of distributed applications and their execution process. As shown by the analysis of experimental results, this understanding is directly transferable to the user in terms of practical indications on how to optimize the time to completion of distributed applications.

4.1 Current Capabilities

The AIMES middleware currently exposes capabilities at the application, resource, and execution management level. A user of the middleware can use the skeleton module to concisely describe a distributed application composed of many executable tasks by specifying their number, their duration, their intercommunication requirements, their data dependences, and their grouping into stages. Using the bundle module, the user can also specify a set of resources that should be considered. This information consists of not only the capacity of a resource and some of its configuration properties, but also the availability of such resources in terms of utilization, state, and composition of their queues, and type of jobs already scheduled for execution. Finally, the user can set the parameters that need to be evaluated by the execution manager to derive an execution strategy.

Internally, the AIMES middleware manages the execution of the given application skeleton by acquiring its characteristics through the skeleton API (Figure 1, step 1). The same is

[1]This refers to a statistical distribution and not a distribution of resources.

done with the information about the available resources via the bundle API (Figure 1, step 2). An execution strategy is then incrementally derived by the execution manager (Figure 1, step 3). In the context of the experiments presented in this section (see Table 1), the derivation typically begins by setting the degree of execution concurrency and the percentage of the available resources that should be considered for execution, and then progresses until all the following parameters are set: (1) what resource to target for the execution; (2) what type of pilot container to use (typically a 'job' for HPC-like resources); (3) number, size, and duration of the pilots to instantiate; (4) what scheduler to use to place the skeleton tasks on the instantiated pilots; and (5) what type of binding to use for the tasks, early or late.

The values assigned to each decision point of an execution strategy are defined by evaluating the characteristics of the application, of the resources, and of the choices that have been already made for that strategy. For example, a resource is selected depending on the number of available cores, the length of its queue, its load, and the network bandwidth available between the resource and the machine from where the AIMES middleware is being instantiated. Analogously, the number, size, and duration of the pilots are derived by the space and time requirements of the given application. Finally, the pilot container and the type of scheduler are chosen depending on the set of target resources, and on the selected task/pilot binding. Under these conditions, multiple effective execution strategies are derived and an optimization metric is used to select the most appropriate one. The decision process required to set each parameter of the execution strategy can be extended to include arbitrarily complex heuristics and algorithms.

Once the execution strategy has been derived, the given application is executed (Figure 1, steps 4, 5, and 6). Figure 2 demonstrates the execution of a skeleton application with fifty tasks on five resources. The bottom third illustrates the pilot activity, i.e., how many tasks have been executed on each pilot, distinguished via color. The middle third of the diagram shows the progression of each task (units) through all of its valid states, with each state transition represented by a vertical line and the time spent by that task in that state represented by the horizontal line. Finally, the top third shows the state progression of each pilot used to execute the given tasks.

The figure offers a concise yet complete explanation of all the stages of the execution of a skeleton application. Pilots advance through some preparatory states and are sent to remote resource's queue (`PENDING ACTIVE`, top third of the diagram.) Once they become active, tasks transition from the `UNSCHEDULED` state to transferring their input files to the remote resource (`PEND. INPUT` and `TRANSF. INPUT`,) then to being scheduled and executed (middle third of the diagram). When the tasks start to execute, they acquire a core from those made available by the active pilots (bottom third of the diagram.) Each task executes between 1,600 and 8,900 seconds, then transfers its output (middle third of the diagram,) and finally enters the `DONE` state. Once all the units have been executed, all scheduled pilots are canceled so not to waste resources.

4.2 Experimental Setup and Design

We used the AIMES middleware to perform experiments exploring several properties of distributed applica-

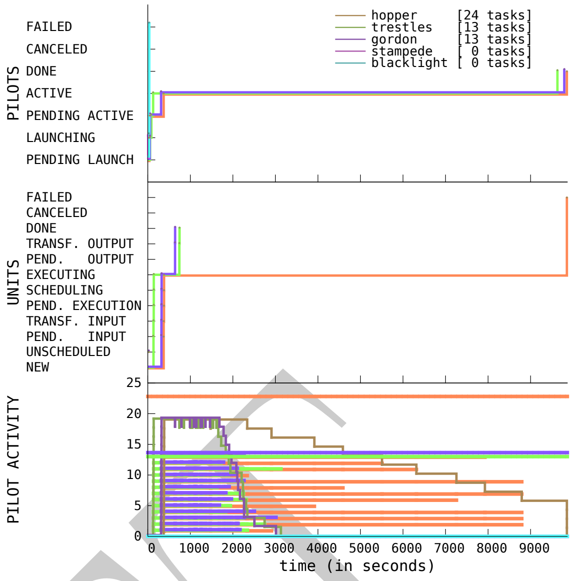

Figure 2: Fifty tasks late bound to five pilots submitted to XSEDE and NERSC resources. Tasks are backfilled and executed on the first three pilots that became active on Hopper, Trestles, and Gordon.

tions. Here we describe the design and results of four experiments measuring the impact of two execution strategies on the overall time to completion (TTC) of two types of skeleton (see Table 1). Both types of skeletons have nine applications, each with between 8 and 2,048 single-core tasks. In one type of skeleton, each task of each application takes 15 minutes to execute, while in the other, the task length follows a truncated Gaussian distribution (mean: 15 min.; stdev: 5 min.; lower bound: 1 min.; upper bound: 30 min.) The execution strategy decisions, measured for their impact on the TTC of each skeleton application, were: (1) early or late binding of tasks to pilots; (2) the type of scheduler used to place tasks on pilots; (3) the number of pilots; (4) their size; and (5) their walltime.

The four experiments consist of 36 unique combinations of skeleton application and execution strategy properties. Execution of each combination (i.e., a run) is repeated as many times as required by the relative error of the measurements, with execution order varied to avoid correlation among combinations. The experimental runs are repeated at irregular intervals so to avoid dependences based on short-term resource load patterns. The order in which pilots are submitted to the resources is also varied to account for differences in submission time across resources.

Table 1 shows some subtle differences among experiments. Experiment 1 and 2 bind all the tasks to a single pilot while 3 and 4 use up to three pilots. This is due to how early binding works with multiple pilots. In an experiment with early binding, tasks are bound to the pilots before they become active, without therefore taking into account pilot availability, load, or location. As such, the TTC of the experiment is determined by the last pilot becoming active and all its tasks being executed. In this scenario, the TTC of an experiment using early binding on multiple pilots would be equivalent to an experiment executing all of the tasks on a single pilot

Experiment	Skeleton Application		Execution Strategy				
ID	**#Tasks**	**Task Duration (min)**	**Binding**	**Scheduler**	**#Pilots**	**Pilot Size**	**Pilot Walltime**
1	$2^n, n = [3, 11]$	15	Early	Direct	1	$\#Tasks$	$T_x + T_s + T_{rp}$
2	$2^n, n = [3, 11]$	$1 - 30$ (trunc. Gaussian)					
3	$2^n, n = [3, 11]$	15	Late	Backfill	$1 - 3$	$\frac{\#Tasks}{\#Pilots}$	$\frac{T_x+T_s+T_{rp}}{\#Pilots}$
4	$2^n, n = [3, 11]$	$1 - 30$ (trunc. Gaussian)					

Table 1: Skeleton applications and execution strategies used for the experiments. Each application task runs on a single core. T_x = estimated workflow execution time; T_s = estimated total data staging time; T_{rp} = AIMES middleware overhead.

assuming that for all the experimental runs, the single pilot is instantiated on the same target resource.

Another difference is that experiments 3 and 4 use three pilots with 1/3 of the walltime and 1/3 of the cores of the single pilot used in experiments 1 and 2. This is justified by considering two issues: overuse of resources, and the difference between direct and backfill scheduling. Using three pilots, each with enough cores to execute all the given tasks, would mean asking for three times the resources needed to run the given distributed application. Furthermore, even accepting the inefficiency of such an overuse, all the tasks of the application would be scheduled on the first pilot that becomes available. This is a corner case of experiments 1 and 2, namely, all tasks are executed in a single pilot because the other two pilots do not become available in time. The only difference is that for experiments 3 and 4, the total TTC in such a corner case would be roughly three times longer than one in which a pilot three times larger had been used. Such a difference can be accounted for analytically, without need for an empirical validation.

4.3 Results and Analysis

The experiments were performed over the course of almost a year, repeated more than 20,000 times for a total of 10 million tasks executed. The collected experimental data were analyzed to identify the dominant factors that contribute to the TTC of the distributed application. Figure 3 shows the three main components of TTC for the strategies listed in Table 1: (1) the time spent to setup the execution of the experiment and by the pilot(s) while waiting to become active on the target resource(s) (T_w); (2) the time taken to execute all the tasks of the application on the available pilot(s) (T_x); and (3) the time spent to stage in and out the input and output data of the application (T_s).

T_s depends on the distributed application considered and is restricted to a small percentage of the overall TTC. Larger amounts of data could easily make T_s dominant and a set of dedicated experiments would be required to investigate the differences among strategies using decision points about, for example, compute/data affinity, amount of network bandwidth available between the origin of the data and the target resource(s), or the number and location of data replicas. This is the subject of future work.

T_x also depends on the application considered. For these experiments, the duration of the tasks are either 15 minutes or a truncated Gaussian distribution between 1 and 30 minutes. These values were selected because they are consistent with the duration of several scientific applications [53] that could benefit from distributed execution. This also minimized the relatively high but well-characterized overheads of the RADICAL pilot prototype, especially for applications with 1024 and 2048 tasks.

Finally, T_w depends mostly on the queuing time of the resource, which is determined by the state of the resource when the pilot job is submitted and on the specific configuration of the resource scheduler. As such, T_w is outside the control of the user and of the AIMES middleware.

It should be noted that the measurements of T_s, T_x, and T_w are not only aggregates of multiple elements (e.g. the overheads of RADICAL pilot, the performance of the resource scheduler and data subsystem, or the network latency) but also a function of the overlap among the states of each task and pilot. For example, T_x depends on the duration of each task plus the RADICAL pilot and resource overheads but, as tasks are executed with a certain degree of concurrency, the overall T_x of the application will also be a function of the overlap among task durations.

Figure 3 shows at least two relevant differences across experiments. First, T_x varies over uniform and Gaussian distributions and over early and late binding. Not surprisingly, the variation over distributions is mainly due to the different duration of the tasks (15 minutes for the uniform distribution, between 1 and 30 minutes for the truncated Gaussian) along with differing concurrency due to these different lengths. The variation of T_x over types of binding is explained by the number and size of their pilots. The strategies adopting early binding can count on a single pilot as large as the sum of the three of the strategies using late binding. As a consequence, the tasks are executed with higher concurrency for early binding strategies than for the late binding strategies. Second, T_w varies over early and late binding. This is interesting because it shows how distributing the execution across multiple target resources allows for consistently reducing the average time spent waiting for at least one pilot to become active.

Figure 4 (a) and (b) show that the advantage of late over early binding depends almost exclusively on the differences in T_w, i.e. pilot queuing time (note how the shapes of the four lines in (b) closely resemble those of the overall TTC for each experiment in (a)). The large error bars of Figure 4 (c) shows the variability of T_w for the same job submitted multiple times to the same resource with early binding. An analogous degree of variability can be observed for Gaussian distributions in Figure 4 (a) and the differences at 16 and 1024 tasks is a strong indication of how widely TTC for early binding varies. Surprisingly, this large variability is already overcome by targeting three resources (pilots) as shown in Figure 4 (d). The interplay of the differences in queuing time among the resources and backfill scheduling of the tasks on the free cores available on each active pilot eliminates the dominance of queue time in the TTC. Accordingly, Figure 4 (d) shows reduced error bars and a consistently better TTC

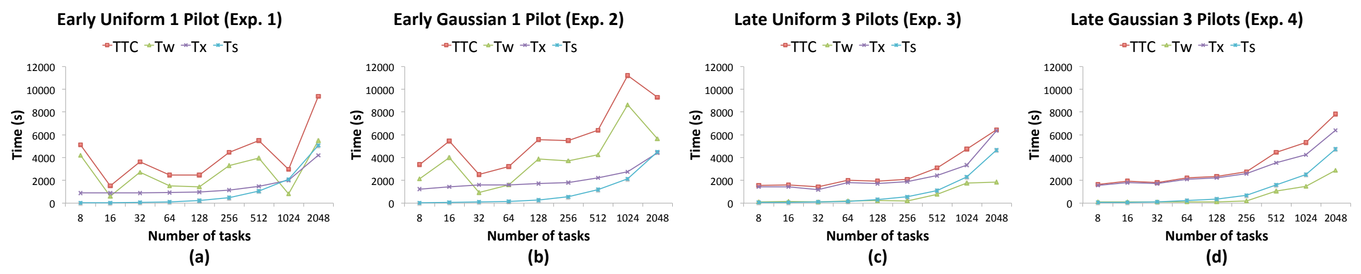

Figure 3: TTC and its time constituents presented for each experiment in Table 1 as a function of the distributed application size. T_w = pilot setup and queuing time; T_x = execution time; T_s = input/output files staging time. During execution (T_w), (T_x), and (T_s) overlap so TTC $< T_w + T_x + T_s$.

across all the distributed application sizes.

It should be noted that early binding would still be desirable for distributed applications with a duration of T_x long enough to make the worse case scenario of T_w irrelevant. In this case, early binding would offer better TTC because of the larger size of its single pilot and therefore the greater level of concurrent execution it would support for the application tasks. Both space and time efficiency would be maintained as all the pilot cores would be utilized and no walltime would be used on the target resource beyond the duration of T_x.

The analysis of the experimental data leads to four main contributions to the practice of executing distributed applications. First, the relevance of the abstraction of distributed execution strategy is confirmed by showing the correlation between its decision points and the optimization of distributed applications TTC, **independently** of the number of tasks and the distribution of their durations (compare Figure 3 (a) and (c), (b) and (d)). A targeted set of experiments will be used to explore the correlation between TTC, late binding, and varying data requirements of the distributed application.

Second, the explicit control of the coupling between application requirements and resource capabilities by means of an execution manager showed the potential for better performance when executing distributed application, both in terms of scale and TTC (compare Figure 4 (c) and (d)). The coupling process is usually enacted implicitly, on the basis of informal heuristics developed by the users via trial and error. Unavoidably, such an approach limits the scalability of distributed applications and the optimal utilization of available resources. An execution manager gives the user the opportunity to control and explicitly optimize the coordination of the execution process of distributed applications.

Third, distributing the execution of multiple tasks with heterogeneous or uniform duration and size by means of multiple pilots instantiated across diverse resources is an effective way to improve TTC (compare Figure 4 (c) and (d)). While this benefit has been advocated and measured before, these experiments show how such a concrete benefit can be obtained by means of a lightweight middleware, executed on the users' systems, with minimal or no requirements on the software stack installed on each target resource.

Fourth, abstracting and unifying the representation of resource capabilities alongside enabling interoperability and adaptivity of the execution process is an effective way to manage overheads outside the direct control of the user. Specifically, notoriously unpredictable queuing time [34] on HPC-like resources (Figure 4 (a) and (b)) was effectively normalized for different size and types of distributed applications (Figure 4 (c) and (d)) without circumventing the resource scheduler or 'playing' its fairness policies. Users can leverage immediately this result obtaining a sizable improvement of the TTC of their distributed applications.

5. DISCUSSION AND CONCLUSIONS

Existing production DCI such as OSG and XSEDE are designed for specific classes of applications. It is neither easy to move applications from one DCI to another nor to expand the type of applications supported by each DCI. This suggests a possible lack of abstractions and design principles for DCI, and results in distributed applications that are fragile. Not surprisingly, the practice of large-scale science and engineering on DCI is often characterized by 'heroic' activity, which is not scalable or sustainable. The contribution of this paper is to show that what had been 'heroic' can be converted to routine practice. Specifically, via persistent experimentation on production DCI sustained over almost a year at unprecedented scale, we illustrate that the abstractions proposed in this paper, once implemented and integrated, have the following impacts: (1) they enable improved distributed execution; (2) they serve as a starting point to support better, even if somewhat limited, reasoning about execution on DCI; and (3) they enable the (elementary) design of DCI.

To benchmark the progress that has been made, we note that in 2005-9, a 'heroic' but ultimately unachievable attempt was made to use multiple distributed TeraGrid resources to execute O(1000) parallel and concurrent tasks [8]. The complexity and challenges of scale prevented a well designed 'distributed' algorithm from being successful. This previous attempt can now be achieved, at least in scale and number of tasks. This is due in part to more scalable infrastructure, but also due to effectiveness of the abstractions presented here. It is not just the case that we can now do in 2015 what we could not do a decade earlier, but we now know how to do it in an extensible and general-purpose manner.

Improving the practice of distributed execution has a tangible impact on scalable applications, novel algorithms, and current and future production DCI. For example, enhanced distributed execution based on a mix of job sizes has been used to improve resource utilization on DOE supercomputers [54]. For future benefits, imagine an architect of a production DCI trying to solve: Given a fixed number of cores and workloads up to size N, over how many independent

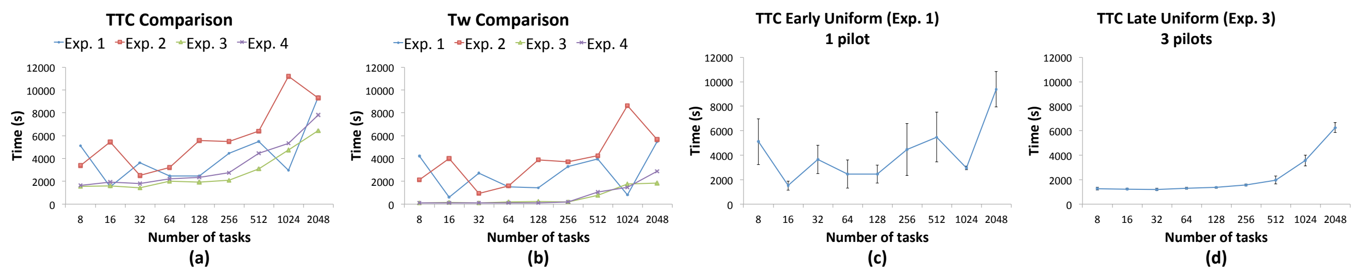

Figure 4: Comparison among the TTCs of each experiment (see Table 1). Discontinuity of TTC in experiment 1 and 2 and smooth progression of TTC in experiment 3 and 4 (a) are due to differences in T_w (b). Relative errors in (c) and (d) confirms the difference in variance of T_w.

distributed (physically or logically) resources should these cores be spread so as to engineer a DCI with a well defined response time? Figure 4 (d) provides preliminary insight to the imaginary architect: by supporting late-binding on up to three distinct resources, the processing time of a distributed application comprised of up to 256 tasks can be kept constant, and can be well bounded for applications with up to 2048 tasks (as evidenced by the plateau followed by linear increase but with small error bars).

The experimental results presented in Section 4 hold for distributed applications with spatial and temporal heterogeneity, which provides empirical confidence that large-scale DCI can be architected and designed to specification and purpose, as opposed to simply putting heterogeneous resources together to construct DCI, as it is currently done.

We believe these are the first reported experiments that present 'collective' properties of multiple distributed resources. This was possible because the AIMES middleware is composed by integrating abstractions that represent well-defined functionality and clean interfaces. In this way, the middleware that supports investigation of the research challenges underpinning distributed execution also supports production-grade experiments. In general, the AIMES middleware provides a laboratory for testing many existing and future ideas; this serves as further justification of the significant effort that has been spent in building the laboratory infrastructure, that is, implementing and integrating the abstractions to work on heterogeneous resources.

We are incrementally but consistently improving the AIMES virtual laboratory: We are in the process of integrating Swift [21] into our system at the same level as the current application skeletons to allow testing of greater range of applications. The execution manager currently uses heuristic techniques, and we will explore other options for its decision making, including using a formal rule engine and artificial intelligence techniques.

In order to validate claims of generality, enhance ability to work with different types of heterogeneity at greater scale, and improve both our understanding and capabilities on production infrastructure, we are extending our work in three distinct directions. First, we are incorporating networks into our work. Specifically, we will add network monitoring information to the resource bundle, and then use this information in the execution strategies. To study this, we will also work with applications where the data sizes are more varied and have more impact than in our current experiments. Second, we are also starting to use Open Science Grid, which presents a very different usage model (HTC vs. HPC), capabilities and interface. This involves changes to the resource bundle and execution manager to enable unified decisions across different types of systems. Third, we are designing experiments with distributed applications that have size and duration probability distributions similar to the actual load on XSEDE. The ability to mimic XSEDE's load will help us to understand some of the challenges in designing execution management services. In fact, we are already running experiments for distributed applications comprised of non-uniform task sizes in addition to temporal variations.

A set of challenges remain to be addressed to guide the future-of-the-practice. Execution strategies will have to evolve to manage significantly greater levels of spatiotemporal heterogeneity that will characterize the resources of the near future. This will require not only qualitative abstractions to evolve but also quantitative models of execution for distributed application and of DCI resources federation to be developed. Together, these advances will enable predictive modeling of distributed execution and must be implemented in middleware available to applications, tools, systems, and services so as to prevent each one having to 'roll its own'.

6. CONTRIBUTIONS AND ACKNOWLEDGMENTS

MT, SJ, DSK, JW planned this paper, and it was written primarily by MT with contributions from SJ, DSK, FL and JW. Experiments core to this paper were designed, executed and analyzed by MT. SJ and AM helped design experiments; FL, AM and ZZ contributed to the implementation of the infrastructure for the experiments. DSK introduced the idea of application skeletons, which were designed at high level by DSK, MW, and ZZ. ZZ led the detailed design, implementation, and testing, with participation from DSK and MW. FL and JW designed the concept of resource bundles.

This work is primarily funded by DOE ASCR under grants DE-FG02-12ER26115, de-sc0008617, and de-sc0008651, and NSF CAREER under ACI-1253644 (SJ). This work has also been made possible thanks to computer resources provided by XRAC award TG-MCB090174 and resources provided by NERSC. We thank Mark Santcroos and Yadu Nand Babuji for useful contributions to RADICAL Pilot and skeletons respectively. Work by Katz was supported by the National Science Foundation while working at the Foundation. Any opinion, finding, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the NSF.

7. REFERENCES

[1] Large Synoptic Survey Telescope http://www.lsst.org/lsst/.

[2] The Square Kilometer Array https://www.skatelescope.org/.

[3] T. Hey, S. Tansley, and K. Tolle, Eds., *The Fourth Paradigm: Data-Intensive Scientific Discovery.* USA: Microsoft Research, 2009.

[4] "Best papers of hpdc (1992-2012)," http://www.hpdc.org/best.php/.

[5] D. S. Katz, D. Hart, C. Jordan, A. Majumdar, J. Navarro, W. Smith, J. Towns, and N. W.-D. V. Welch and, "Cyberinfrastructure usage modalities on the teragrid," in *2011 High-Performance Grid and Cloud Computing Workshop*, 2011, pp. 927–934.

[6] "XSEDE: Extreme Science and Engineering Discovery Environment," https://www.xsede.org/, 2012.

[7] Adofly Hoise et al, Report of the Workshop on Modeling and Simulation of Systems and Applications, University of Washington, Seattle. http://hpc.pnl.gov/modsim/2014/Presentations/ModSim_2014_Report_FINAL.pdf.

[8] H. S. C. Martin, S. Jha, S. Howorka, and P. V. Coveney, "Determination of free energy profiles for the translocation of polynucleotides through alpha-hemolysin nanopores using non-equilibrium molecular dynamics simulations," *Journal of Chemical Theory and Computation*, vol. 5, no. 8, pp. 2135–2148, 2009, see Supplementary Material at: http://pubs.acs.org/doi/suppl/10.1021/ct9000894/suppl_file/ct9000894_si_003.pdf. [Online]. Available: http://dx.doi.org/10.1021/ct9000894

[9] S. Jha, M. Cole, D. S. Katz, M. Parashar, O. Rana, and J. Weissman, "Distributed computing practice for large-scale science and engineering applications," *Concurrency and Computation: Practice and Experience*, vol. 25, no. 11, pp. 1559–1585, 2013. [Online]. Available: http://dx.doi.org/10.1002/cpe.2897

[10] I. Foster, J. Geisler, B. Nickless, W. Smith, and S. Tuecke, "Software infrastructure for the I-WAY high performance distributed computing experiment," in *Proc. 5th IEEE Symposium on High Performance Distributed Computing*, 1996, pp. 562–571.

[11] A. S. Grimshaw and W. A. Wulf, "The legion vision of a worldwide virtual computer," *Comm. of the ACM*, vol. 40, no. 1, January 1997.

[12] I. Foster, C. Kesselman, and S. Tuecke, "The anatomy of the grid - enabling scalable virtual organizations," *International Journal of Supercomputer Applications*, vol. 15, p. 2001, 2001.

[13] I. Foster, C. Kesselman, J. M. Nick, and S. Tuecke, "The physiology of the grid: An open grid services architecture for distributed systems integration," 2002.

[14] G. Allen, I. Foster, N. T. Karonis, M. Ripeanu, E. Seidel, and B. Toonen, "Supporting efficient execution in heterogeneous distributed computing environments with cactus and globus," in *Supercomputing, ACM/IEEE 2001 Conference.* IEEE, 2001, pp. 52–52.

[15] S. H. Bokhari, "A shortest tree algorithm for optimal assignments across space and time in a distributed processor system," *IEEE Trans. Softw. Eng.*, vol. 7, no. 6, pp. 583–589, Nov. 1981. [Online]. Available: http://dx.doi.org/10.1109/TSE.1981.226469

[16] D. Fernandez-Baca, "Allocating modules to processors in a distributed system," *IEEE Trans. Softw. Eng.*, vol. 15, no. 11, pp. 1427–1436, Nov. 1989. [Online]. Available: http://dx.doi.org/10.1109/32.41334

[17] T. D. Braun, H. J. Siegel, N. Beck, L. Boloni, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys, and B. Yao, "A taxonomy for describing matching and scheduling heuristics for mixed-machine heterogeneous computing systems," in *Reliable Distributed Systems, 1998. Proceedings. Seventeenth IEEE Symposium on.* IEEE, 1998, pp. 330–335.

[18] W. Chen and E. Deelman, "Partitioning and scheduling workflows across multiple sites with storage constraints," in *Parallel Processing and Applied Mathematics*, ser. Lecture Notes in Computer Science, R. Wyrzykowski, J. Dongarra, K. Karczewski, and J. Waśniewski, Eds. Springer Berlin Heidelberg, 2012, vol. 7204, pp. 11–20. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-31500-8_2

[19] M. Malawski, G. Juve, E. Deelman, and J. Nabrzyski, "Algorithms for cost- and deadline-constrained provisioning for scientific workflow ensembles in iaas clouds," *Future Generation Computer Systems*, vol. 48, pp. 1 – 18, 2015, special Section: Business and Industry Specific Cloud. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167739X15000059

[20] I. Altintas, C. Berkley, E. Jaeger, M. Jones, B. Ludascher, and S. Mock, "Kepler: an extensible system for design and execution of scientific workflows," in *Scientific and Statistical Database Management, 2004. Proceedings. 16th International Conference on.* IEEE, 2004, pp. 423–424.

[21] M. Wilde, M. Hategan, J. M. Wozniak, B. Clifford, D. S. Katz, and I. Foster, "Swift: A language for distributed parallel scripting," *Par. Comp.*, pp. 633–652, September 2011.

[22] T. Oinn, M. Addis, J. Ferris, D. Marvin, M. Senger, M. Greenwood, T. Carver, K. Glover, M. R. Pocock, A. Wipat *et al.*, "Taverna: a tool for the composition and enactment of bioinformatics workflows," *Bioinformatics*, vol. 20, no. 17, pp. 3045–3054, 2004.

[23] I. T. Foster and R. L. Stevens, "Parallel programming with algorithmic motifs," in *Proceedings of the 1990 International Conference on Parallel Processing, Urbana-Champaign, IL, USA, August 1990. Volume 2: Software*, 1990, pp. 26–34.

[24] L. Meyer, M. Mattoso, M. Wilde, and I. Foster, "WGL - a workflow generator language and utility," http://dx.doi.org/10.6084/m9.figshare.793815.

[25] J. Logan, S. Klasky, H. Abbasi, Q. Liu, G. Ostrouchov, M. Parashar, N. Podhorszki, Y. Tian, and M. Wolf, "Understanding I/O Performance Using I/O Skeletal Applications," in *Euro-Par 2012 Parallel Processing*, ser. Lecture Notes in Computer Science, C. Kaklamanis, T. Papatheodorou, and P. G. Spirakis, Eds. Springer Berlin Heidelberg, 2012, vol. 7484, pp. 77–88. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-32820-6_10

[26] M. Cardosa and A. Chandra, "Resource bundles: Using aggregation for statistical wide-area resource discovery and allocation," in *Distributed Computing Systems, 2008. ICDCS'08. The 28th International Conference on.* IEEE, 2008, pp. 760–768.

[27] C. Liu and I. Foster, "A constraint language approach to grid resource selection," 2003, unpublished manuscript, available from http://people.cs.uchicago.edu/~chliu/grads/doc/sc03-liu.pdf.

[28] D. Oppenheimer, J. Albrecht, D. Patterson, and A. Vahdat, "Design and implementation tradeoffs for wide-area resource discovery," in *High Performance Distributed Computing, 2005. HPDC-14. Proceedings. 14th IEEE International Symposium on*, July 2005, pp. 113–124.

[29] D. Nurmi, J. Brevik, and R. Wolski, "Qbets: queue bounds estimation from time series," in *Job Scheduling Strategies for Parallel Processing.* Springer, 2008, pp. 76–101.

[30] A. B. Downey, "Predicting queue times on space-sharing parallel computers," in *Parallel Processing Symposium, 1997. Proceedings., 11th International.* IEEE, 1997, pp. 209–218.

[31] R. Wolski, "Experiences with predicting resource performance on-line in computational grid settings," *ACM SIGMETRICS Performance Evaluation Review*, vol. 30, no. 4, pp. 41–49, 2003.

[32] W. Smith, V. Taylor, and I. Foster, "Using run-time predictions to estimate queue wait times and improve scheduler performance," in *Job Scheduling Strategies for Parallel Processing.* Springer, 1999, pp. 202–219.

[33] H. Li, D. Groep, J. Templon, and L. Wolters, "Predicting job start times on clusters," in *Cluster Computing and the Grid, 2004. CCGrid 2004. IEEE International Symposium on.* IEEE, 2004, pp. 301–308.

[34] D. Tsafrir, Y. Etsion, and D. G. Feitelson, "Backfilling using system-generated predictions rather than user runtime estimates," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 18, no. 6, pp. 789–803, 2007.

[35] I. Raicu, I. Foster, and Y. Zhao, "Many-task computing for grids and supercomputers," in *Proc. of Many-Task Comp. on Grids and Supercomputers, 2008*, 2008.

[36] Z. Zhang and D. S. Katz, "Using application skeletons to improve eScience infrastructure,," in *Proceedings of 10th IEEE International Conference on eScience*, 2014.

[37] D. S. Katz, A. Merzky, M. Turilli, M. Wilde, and Z. Zhang, "Application Skeleton v1.2," *GitHub*, Jan 2015, dOI: 10.5281/zenodo.13750. [Online]. Available: http://dx.doi.org/10.5281/zenodo.13750

[38] E. Deelman, G. Singh, M.-H. Su, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahl, G. B. Berriman, J. Good, A. Laity, J. C. Jacob, and D. S. Katz, "Pegasus: A framework for mapping complex scientific workflows onto distributed systems," *Scientific Programming*, vol. 13, no. 3, pp. 219–237, 2005.

[39] J. C. Jacob, D. S. Katz, G. B. Berriman, J. C. Good, A. C. Laity, E. Deelman, C. Kesselman, G. Singh, M.-H. Su, T. A. Prince, and R. Williams, "Montage: a grid portal and software toolkit for science-grade astronomical image mosaicking," *Intl. J. of Comp. Sci. and Eng.*, vol. 4, no. 2, pp. 73–87, 2009.

[40] D. R. Mathog, "Parallel BLAST on split databases," *Bioinformatics*, vol. 19, no. 14, pp. 1865–1866, 2003. [Online]. Available: http://bioinformatics.oxfordjournals.org/content/19/14/1865.abstract

[41] P. Maechling, E. Deelman, L. Zhao, R. Graves, G. Mehta, N. Gupta, J. Mehringer, C. Kesselman, S. Callaghan, D. Okaya, H. Francoeur, V. Gupta, Y. Cui, K. Vahi, T. Jordan, and E. Field, "SCEC CyberShake workflows – automating probabilistic seismic hazard analysis calculations," in *Workflows for e-Science*, I. J. Taylor, E. Deelman, D. B. Gannon, and M. Shields, Eds. Springer, 2007, pp. 143–163.

[42] Y. Simmhan and L. Ramakrishnan, "Comparison of resource platform selection approaches for scientific workflows," in *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing.* ACM, 2010, pp. 445–450.

[43] A. Raghavan, A. Chandra, and J. Weissman, "Tiera: towards flexible multi-tiered cloud storage instances," in *Proceedings of the 15th International Middleware Conference.* ACM, 2014, pp. 1–12.

[44] A. Luckow, M. Santcroos, A. Merzky, O. Weidner, P. Mantha, and S. Jha, "P*: A model of pilot-abstractions," *2012 IEEE 8th International Conference on E-Science*, pp. 1–10, 2012, http://doi.org/10.1109/eScience.2012.6404423.

[45] G. Aad *et al.*, "The atlas experiment at the cern large hadron collider," *JINST*, vol. 3, p. S08003, 2008.

[46] T. Maeno, K. De, A. Klimentov, P. Nilsson, D. Oleynik, S. Panitkin, A. Petrosyan, J. Schovancova, A. Vaniachine, T. Wenaus, and D. Yu, "Evolution of the atlas panda workload management system for exascale computational science," *J. Phys.: Conf. Ser.: Journal of Physics: Conference Series*, vol. 513, no. 3, 2014.

[47] R. Pordes *et al.*, "The open science grid," *Journal of Physics: Conference Series*, vol. 78, no. 1, p. 012057, 2007.

[48] D. S. Katz, S. Jha, M. Parashar, O. Rana, and J. B. Weissman, "Survey and analysis of production distributed computing infrastructures," arXiv, Tech. Rep. 1208.2649, 2012, http://arxiv.org/abs/1208.2649. [Online]. Available: http://arxiv.org/abs/1208.2649

[49] M. Hategan, J. Wozniak, and K. Maheshwari, "Coasters: uniform resource provisioning and access for clouds and grids," in *4th IEEE/ACM International Conference on Utility and Cloud Computing*, 2011.

[50] "Fireworks workflow software," http://pythonhosted.org/FireWorks, Apr 2015. [Online]. Available: http://pythonhosted.org/FireWorks

[51] A. Merzky, O. Weidner, and S. Jha, "SAGA: A standardized access layer to heterogeneous distributed computing infrastructure," *Software-X*, dOI: 10.1016/j.softx.2015.03.001. [Online]. Available: http://dx.doi.org/10.1016/j.softx.2015.03.001

[52] T. Goodale, S. Jha, H. Kaiser, T. Kielmann, P. Kleijer, A. Merzky, J. Shalf, and C. Smith, "A Simple API for Grid Applications (SAGA)," Open Grid Forum, OGF

Recommendation, GFD.90, 2007. [Online]. Available: http://ogf.org/documents/GFD.90.pdf

[53] B. K. Radak, M. Romanus, T.-S. Lee, H. Chen, M. Huang, A. Treikalis, V. Balasubramanian, S. Jha, and D. M. York, "Characterization of the Three-Dimensional Free Energy Manifold for the Uracil Ribonucleoside from Asynchronous Replica Exchange Simulations," *Journal of Chemical Theory and Computation*, vol. 11, no. 2, pp. 373–377, 2015. [Online]. Available: http://dx.doi.org/10.1021/ct500776j

[54] A. Klimentov, P. Buncic, K. De, S. Jha, T. Maeno, R. Mount, P. Nilsson, D. Oleynik, S. Panitkin, A. Petrosyan, R. Porter, K. Read, A. Vaniachine, J. Wells, and T.Wenaus, "Next Generation Workload Management System for Big Data on Heterogeneous Distributed Computing," in *16th International Workshop on Advanced Computing and Analysis Techniques in Physics Research (ACAT)*, September 2014, Keynote and Paper, https://indico.cern.ch/event/258092/page/17.