VGrADS: Enabling e-Science Workflows on Grids and Clouds with Fault Tolerance

Lavanya Ramakrishnan
Indiana University, Bloomington
laramakr@cs.indiana.edu

Daniel Nurmi
University of California, Santa Barbara
nurmi@cs.ucsb.edu

Anirban Mandal
Renaissance Computing Institute
anirban@renci.org

Charles Koelbel
Rice University
chk@cs.rice.edu

Dennis Gannon
Microsoft Research
degannon@microsoft.com

T. Mark Huang
University of Houston
tihuang@tlc2.uh.edu

Yang-Suk Kee
Oracle US Inc.
yang.seok.ki@oracle.com

Graziano Obertelli
University of California, Santa Barbara
graziano@cs.ucsb.edu

Kiran Thyagaraja
Rice University
kiran@rice.edu

Rich Wolski
University of California, Santa Barbara
rich@cs.ucsb.edu

Asim YarKhan
University of Tennessee, Knoxville
yarkhan@cs.utk.edu

Dmitrii Zagorodnov
University of California, Santa Barbara
dmitrii@cs.ucsb.edu

ABSTRACT

Today's scientific workflows use distributed heterogeneous resources through diverse grid and cloud interfaces that are often hard to program. In addition, especially for time-sensitive critical applications, predictable quality of service is necessary across these distributed resources. VGrADS' virtual grid execution system (vgES) provides an uniform qualitative resource abstraction over grid and cloud systems. We apply vgES for scheduling a set of deadline sensitive weather forecasting workflows. Specifically, this paper reports on our experiences with (1) virtualized reservations for batch-queue systems, (2) coordinated usage of TeraGrid (batch queue), Amazon EC2 (cloud), our own clusters (batch queue) and Eucalyptus (cloud) resources, and (3) fault tolerance through automated task replication. The combined effect of these techniques was to enable a new workflow planning method to balance performance, reliability and cost considerations. The results point toward improved resource selection and execution management support for a variety of e-Science applications over grids and cloud systems.

1. INTRODUCTION

The dream of providing on-demand computing resources for heavy users has been discussed for over a decade under various names: grid computing [15], utility computing [36], cloud computing [28], and others. When the Virtual Grid Application Development Software (VGrADS) [25, 32, 51] project began in 2003, the state of the art was a style of grid computing that allowed High Performance Computing (HPC) applications to run on distributed clusters, but only with careful tending. Applications needed low-level programming for managing heterogeneous resources, scheduling of computation and data movement, fault tolerance, and performance tuning. VGrADS set out to raise the level of programming for grid applications.

More recently, the rise of cloud computing has resulted in similar concerns regarding application management and programming models for scientific workflows. The cloud computing model provides on-demand immediate access to virtual machines on a pay-as-you-go model. Cloud computing has evolved in support of Web 2.0 applications [3] to manage dynamic load. Cloud computing for scientific applications has certain trade-offs. Clouds provide immediate guaranteed access to resources and customizable software environments through virtual machines that overcome some of the challenges with current shared batch queue systems. However, parallel scientific codes often experience a significant performance impact. Thus, we need programmatic methods for applications to study the suitability of different resource environments and make appropriate resource decisions.

We have made substantial progress in two broad areas of simplifying programming applications on grid and cloud systems :

- We separated application development from resource management by providing a uniform *virtual grid* abstraction atop widely differing resources.
- We provided tools to efficiently bridge the abstraction gap, including methods for scheduling, resource management, distributed launch, and fault tolerance.

Resource procurement and planning interfaces and strategies are especially critical for time-sensitive applications. Mesoscale meterology workflows in the Linked Environments for Atmospheric Discovery (LEAD) [10] project are examples of such deadline-sensitive workflows. These workflows are initialized by streaming sensor data and consist of a number of preprocessing steps that feed data to a computation-intensive weather model. These workflows

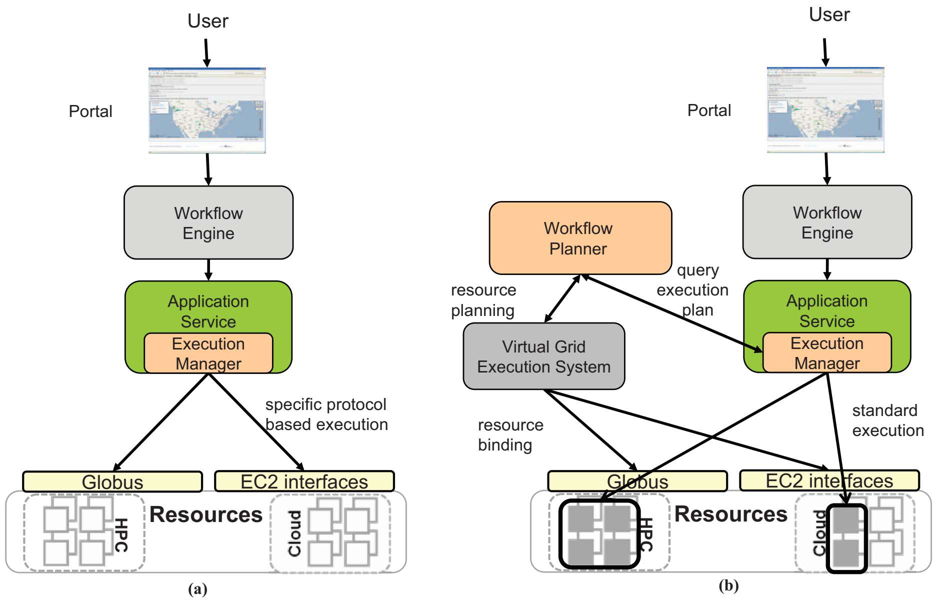


Figure 1: Comparison of the LEAD-VGrADS collaboration system with cyberinfrastructure production deployments.

require timely and coordinated access to computation and data during workflow execution.

This paper brings together many of the results from the VGrADS project demonstrating the effectiveness of virtual grids for scheduling LEAD workflows. In the process, it demonstrates a seamless merging of cloud and HPC resources in service of a scientific application. It also applies advanced scheduling techniques for both performance improvement and fault tolerance in a realistic context.

LEAD has been run as a distributed application since its inception, but VGrADS methods have opened new capabilities for resource management and adaptation in its execution. We present the motivation for the LEAD application and explain the effect of VGrADS on workflow scheduling in Section 2. We discuss VGrADS system components and the execution flow in Section 3. This describes the virtual grid abstraction, including its unified management of TeraGrid clusters and Amazon EC2 and Eucalyptus cloud sites, and our support for fault-tolerant scheduling workflows. We detail our experiments in Section 4 that demonstrate the feasibility of the virtual grid abstraction's new capabilities for e-Science [46] workflows such as LEAD and show that the overhead of the system is low. Finally, in Sections 5 and 6 we discuss related work and present our conclusions.

2. OVERVIEW

We use the LEAD cyberinfrastructure [10] as a testbed for VGrADS ideas. It is a modern cyberinfrastructure, with many features — notably the use of distributed resources for workflow execution — common to other e-Science domains [46]. In addition, LEAD workflows are time sensitive making timely resource procurement crucial. All this makes it an ideal choice to test the VGrADS infrastructure. The lessons learned here are more widely applicable to other e-Science applications and workflows [37].

2.1 LEAD Workflow Example

LEAD users access large-scale data, forecast models, and analysis and visualization tools for mesoscale meteorology through a portal interface. Using the portal interface, they compose, launch and monitor *workflows*, each of which consists of a set of tasks that produce and/or consume data. The need for data to be produced before it is consumed creates dependencies between the tasks. Moreover, many LEAD experiments require *workflow sets*. Because of the many parameters in weather models and uncertainty in the input, computing an exact result is often impossible. In these cases the user must often run the same model many times with different initial parameters to manage the accuracy of the result. Workflow planning in LEAD must support scheduling workflow sets with two characteristics: a deadline D when all work must be completed, and a fraction F of workflows which must complete by that deadline. The LEAD portal uses a workflow engine to manage the task dependencies and execution of a workflow. That engine in turn invokes the corresponding application service that has knowledge about the application's configuration and data.

For example, imagine a LEAD experiment to forecast severe storms. A workflow in this experiment might take as inputs streaming sensor data to be pre-processed and then used to launch an ensemble of weather models. Based on the application's requirements, the experiment might need to finish within three hours (i.e., $D = 3$) to produce timely forecasts, and require 15 of 20 models to complete (i.e., $F = 3/4$) to have acceptable confidence. It would be the workflow engine and application services's job to schedule the tasks so that those constraints are met.

2.2 Virtual Grid Abstraction

Figure 1 shows a simplified view of LEAD cyberinfrastructure, both with and without VGrADS. Both versions support the same user interface to construct and submit workflows. However, the *virtual grid* abstraction (described in detail in Section 3) enables a more sophisticated and effective scheduling of the workflow sets using the virtual grid abstraction. Originally (Figure 1(a)), the application services interacted directly with distributed sites using specific interfaces. This led to ad-hoc resource scheduling decisions distributed across different components. In turn, this made workflow planning and real-time adaptation extremely challenging, particularly with respect to performance and reliability variability. With VGrADS (Figure 1(b)), the *Virtual Grid Execution System (vgES)* provides a uniform interface for provisioning, querying, and controlling the resources. Similar resource abstractions have been proposed earlier over grid systems [16, 18]. However, vgES has the

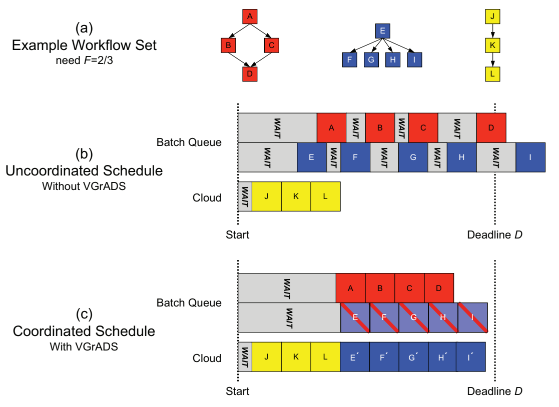


Figure 2: Example scheduling of workflows.

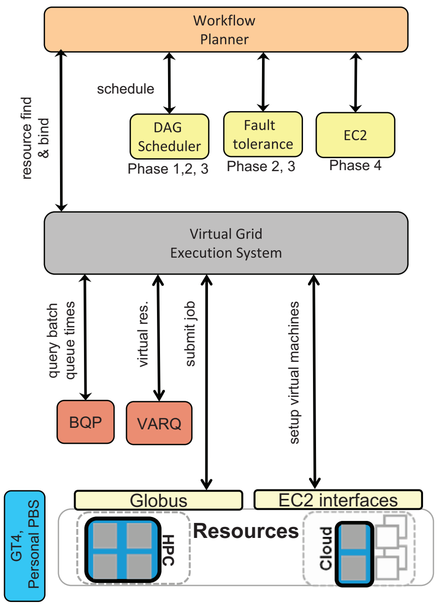


Figure 3: Interaction of system components for resource procurement and planning

ability to harness resources across grid and cloud sites as required by the application. The resources are annotated with QoS properties that aid workflow planning. Thus, the standardized interface enables a multi-phase workflow orchestration pipeline to balance performance, reliability and cost considerations for a set of workflows while being agnostic to specific policies and site implementations. In particular, it allows coordinating resources across the components in a workflow set, and managing fault tolerance for the overall workflow set. It also makes combining disparate resource types, including batch-queued resources (such as TeraGrid clusters) and cloud computing installations (such as EC2 and Eucalyptus), relatively seamless. Our implementation uses the LEAD cyberinfrastructure components to demonstrate the use of the virtual grid abstraction, but other scientific workflow systems will interact similarly with possibly different policies in resource procurement and workflow planning.

2.3 Coordinated Scheduling: An Example

We use an example to illustrate the usefulness of coordinated scheduling across different resource types - grids and clouds. Figure 2a shows a typical example of three small workflows that must be mapped onto a resource set consisting of a batch-queue cluster (which can handle two tasks concurrently) and a cloud resource. Each of the tasks in the workflows represents a scientific code. We label each task with a letter (e.g., the first workflow has tasks A, B, C and D). The graph shows the dependencies between the tasks (e.g., tasks B and C depend on task A). Figure 2b shows a Gantt chart (i.e., task mapping onto resources with time) when scheduling the tasks individually on available resources. Scheduling the tasks individually incurs unpredictable queue waits that cause the deadline to be missed. In this case, each job in a batch queue system incurs an overhead in terms of queue wait time. Similarly, the startup time for virtual machines in cloud systems imposes an initial overhead. This example mirrors current day practice in systems such as LEAD, before the advent of VGrADS.

The qualitative resource abstractions in VGrADS provide the ability to aggregate resources from disparate sources and under different policies. For example, batch queue resources, cloud resources, and resources available through offline agreements such as advanced reservations can be queried and used with a single interface. Thus, by using VGrADS to "reserve" a resource for the entire workflow (Figure 2c), we can avoid those delays. The new schedule meets the deadline even with higher initial wait times (reflecting requests for longer time slots). Moreover, the global view of the computation and the virtual grid allows the scheduler to use slack in the schedule to replicate the second workflow in Figure 2a for fault tolerance. This is represented by the new tasks E', F', G', H', and I'. Even if the copy of this workflow on the batch system (i.e. the set of crossed-out tasks) fails, the other copy can complete.

Although Figure 2 is a simplified motivating example, Section 4 will show a similar scheduling improvement on an actual LEAD run. Thus we see that the VGrADS interface provides a single interface to query and manage resources enabling better scheduling of workflows to meet its constraints. Next, we detail the system components of the VGrADS system supporting LEAD workflows.

3. SYSTEM COMPONENTS

Figure 3 shows the interaction of various system components. The first step in the workflow planning process is resource procurement. The planner triggers the activities of vgES system and returns a set of slots or a "Gantt Chart" that represents the resources assigned for this request. This Gantt Chart is then shared between the workflow planner, the DAG scheduler and the fault tolerance component to develop the schedule. The schedule for the workflows is represented as task mappings on each of the slots. The interaction of the virtual grid execution system with the grid and cloud sites are described in greater detail in Section 3.1. While resources are being bound by vgES, the workflow planner uses a four stage planning process to determine the schedule for the LEAD workflow sets to meet the deadline and accuracy constraint. The workflow planner interacts with a DAG scheduler, EC2 planner and fault tolerance sub-components to trade-off various system parameters - performance, reliability and cost. Once the workflow planner determines the workflow plan and the resources are bound, the execution system monitors and manages the workflows. We detail the execution system changes required to support workflow set orchestration in Section 3.3.

Thus, the integrated system is comprised of several interacting components that together provide a resource collection abstraction that we term virtual grids. This abstraction provides methodologies for efficiently utilizing resources through intelligent workflow orchestration, task scheduling, and fault tolerance mechaisms.

3.1 Virtual Grid Execution System

The Virtual Grid Execution System (vgES) [23, 25] acquires resources and ensures their availability during time intervals expressed by the user. vgES can be configured to draw from dedicated resources, batch queue managed resources, and cloud resources accessed through the Amazon Elastic Compute Cloud (Amazon EC2) [2] or Eucalyptus [33] . Internally, vgES maintains a database populated with static information about the entire collection of HPC and cloud resources available, and translates high level requests for resource collections into the most appropriate set of real resources at any given time. Note that, since the provisioning and availability chacteristics of the real resources often change over time, the same abstract request made at different times may result in a different sets of real resources being aquired that still meet the abstract request constraints.

When vgES recieves a request to construct a virtual grid, it performs a database search to find the best collection of resources to provision, and returns a virtual grid abstract data structure. The structure contains meta-data describing performance characteristics of the resources, timing information on when the resources will be available, and locality information about the proximity of the resources to one another. This information is organized as a tree that can be searched to inspect resource characteristics. In addition, vgES provides the view of resources in the form of a Gantt chart that describes how the resources will be provisioned over time.

After a request has been accepted and a virtual grid has been created, vgES finally performs the task of actually provisioning the resources using several "drivers" for the different types of resources (described in detail below) When vgES acquires access to the resources, we install our own, independent version of a Portable Batch Scheduler (PBS) compatible resource manager (Torque) and Globus interfaces [24] as shown in Figure 3. Having our own resource manager running inside each provisioned set of resources gives us both a uniform interface and the ability to reuse a set of resources for several tasks, rather than repeatedly making provisioning requests. Next, we describe how resource procurement is enabled at grid and cloud sites.

3.1.1 Service overlay over grid systems

Currently, most HPC sites use some form of batch scheduling software (e.g., Torque [47], Maui [27]) to manage the mapping of user workload to dedicated compute and storage resources. Typically, sites configure their systems to implement a "space-sharing" policy, where application jobs request resources and the batch queue software provides exclusive access to the requested resources when enough capacity becomes available. When there are more resource requests (jobs) than there are unoccupied resources available, some jobs must wait in a queue. The amount of time that jobs wait in queue on extant HPC sites has been shown to be both substantial (relative to the execution time of the jobs) and highly variable. This uncertaintly makes effective scheduling very difficult, especially for applications with time constraints.

To provide abstract slots for scheduling, even in the presence of batch queue delays, vgES uses an abstraction for making statistical advanced reservations called Virtual Advance Reservations for Queues (VARQ) [31]. VARQ is in turn built on another technology we developed, the Queue Bounds Estimation from Time Series (QBETS) [30] service, which provides statistical predictions of batch queue delays.

Queue Bounds Estimation. Previous studies [9, 43] of predicting batch queue turnaround times have focused on the modeling and prediction of batch queue wait time using point value statistics, such as the mean wait time, as the predictor for future job wait times. While these statistics provide good aggregate predictors, they tend to be poor predictors for individual job wait times due to the highly variable nature of the data (often spanning five to seven orders of magnitude when wait times are measured in seconds [6, 9, 43]). Instead, we predict *upper bounds* on those wait times. For many scientific workflows, including LEAD, the ability to predict the *longest time* that a job will wait in queue is enough to make informed scheduling decisions.

Our methodology for predicting bounds on individual job wait times, QBETS [30], uses historical job traces from over 25 supercomputer centers around the world which send trace updates, in real time, as jobs are processed. We perform one trace-based simulation per machine/queue tuple on this trace applying an automatic job clustering algorithm, an on-line change point detection heuristic, and finally an empirical quantile prediction methodology that results in our final bound prediction. Functionally, our method provides us the ability to service prediction requests in the form of two useful functions. The first function gives a user who is about to submit a job running on N processors for wall-clock time W (N,W) to a machine/queue combination (M,Q) an upper bound queue wait time prediction *bound* with a prediction success probability of *prob*.

$$bound = \text{GETBOUND}(M,Q,N,W,prob)$$

The result of this function is analogous to the statement, *A job* (N,W) *submitted to batch queue* (M,Q) *has a probability* $(prob)$ *of waiting in queue no longer than* $(bound)$. While this prediction is useful in many settings, for deadline-driven applications (e.g. LEAD), we would rather specify the time and predict the probability that the job will start before then. To do this, we can invert GETBOUND() and provide the second useful prediction routine:

$$prob = \text{GETPROB}(M,Q,N,W,bound)$$

Although GETPROB() accurately predicts the probability that a job will start before *bound*, it cannot say exactly when in that time interval the job will execute. That is, the job could start immediately or one second before the deadline. While this suffices for scheduling individual jobs, it is problematic when we are scheduling multiple jobs with dependencies on one another. We need a way to ensure that dependences are enforced, not just statistical estimates of that likelihood. To provide that capability, we build on QBETS with virtual advance reservations.

Virtual Advance Reservation. Virtual Advance Reservations for Queues (VARQ) [31] allows us to make probabilistic advance reservations with no modification to the underlying site software or policies. Our studies of time series data gathered for QBETS showed us that, although the queue wait time experienced by jobs is highly variable, the upper bound predictions produced by QBETS are far more stable. Often, the predictions remain stable for days or weeks changing only when a drastic event occurs (e.g. a deadline for a large user). Thus, we posit that we can legitimately make predictions for future job submissions assuming that large, prediction-changing events are infrequent. Based on this assumption, we have devised a methodology that allows us to search the time between *now* and a specified future job start deadline to find the best time to submit a job to meet the deadline. VARQ computes a *probability trajectory*, in 30 second intervals, by repeated calls to the QBETS GETPROB() routine. At each interval, we know that the job could start immediately after being submitted, and so the execution time of the job W must be adjusted to cover the time to the deadline, plus the job runtime itself. We denote this adjusted job runtime W_{adj}. VARQ uses GETPROB() to obtain a probability prediction, decrementing *bound* and W_{adj} by 30 seconds at each step until the deadline is reached. At the end of this process, VARQ uses the computed probability trajectory and a specified minimum success probability to find the latest point in time where a job can be submitted so that it will start before the specified start time, holding the resources for the entire duration of the job after the deadline arrives. In a previous work [31] we showed that VARQ can successfully make probabilistic advance reservations on large scale production super-computers today. Using VARQ, vgES can time submission to the queue to maximize the probability of resource arrival for virtual grid use.

3.1.2 *Cloud system overlay*

Over the past few years, a number of (mostly commercial) entities have started selling the excess capacity of their resources through an interface that allows a user to start and control virtual machine images that are capable of running their applications. These "cloud computing" systems can provide substantial computing resources to scientists and other users. While the end product of many cloud systems is similar to what a user expects to get from a batch scheduled HPC resource, the way in which the resources are provisioned is very different, and has unique associated costs and benefits. For example, with Amazon's EC2 [2], which is the leading cloud resource provider, a user submits a request for a number of virtual machine instances (based on a common template), and expects to either have those resources provisioned "instantly" (within a few minutes) or for the request to fail. In batch queue systems, the expectation is that a submitted valid job will always run "eventually". This difference adds a bit of complexity to a user who would like to make advance allocation requests using both types of resources in concert. In addition, another difference that impacts the co-scheduling of batch scheduled and cloud resources stems from the fact that batch scheduled resources typically incur a cost that is virtual in nature (users of such systems are commonly given a budget of node/CPU hours that they "spend" when they use resources), where public cloud offerings such as Amazon EC2 charge users based on resource usage using real currency. Balancing these different charging models during a resource scheduling phase must be taken into account and can be complex, depending on the objectives of the user.

In vgES, we have built a "cloud resource" driver that has a similar interface to the previously discussed QBETS/VARQ batch queue driver. Since Amazon EC2 is currently the best specified and most common interface to cloud resources, the driver is built to control virtual machine reservations through the EC2 API. Using this driver, vgES may provision resources from EC2 in order to meet deadlines if the current set of batch queue resources are unlikely to satisfy a user request.

To further augment the number of resources available to users, we have built an open source cloud computing system called Eucalyptus [33] that can be run on any set of machines that supports commonly available open-source software (Linux, Xen, etc). Eucalyptus exports an EC2-compatible interface and therefore the vgES driver can interact with any Eucalyptus installation just as it interacts with Amazon EC2. Thus, using Eucalyptus, we can transform any collection of machines into an EC2-compatible compute cloud that can be seamlessly managed by vgES. Studies into the suitability of cloud computing systems for HPC application workloads [12, 35] indicate that certain applications perform well in existing public clouds while others require more specialized performance/feature enhancements before they become viable HPC resource providers. While the public clouds are generally designed to provide good performance for a wide variety of applications, we can leverage the fact that Eucalyptus can run on local resources to provide those enhancements. The result for us is a specialized cloud computing installation that can utilize high performance networks, access to local parallel file systems, and other resources that are not typically offered by public cloud computing systems. We feel that this is the direction that private clouds and clouds for e-Science use will need to take in the future. We further argue that these experiments suggest that such clouds can be a viable platform for scientific computation.

3.2 Workflow Planner

vgES interacts with different resource types and unifies resources available with certain properties to the workflow planner. For example, scientific workflows such as LEAD often have assigned allocations through TeraGrid and other supercomputing centers as well as other local cluster sites. In addition to these assigned resources, users might have access to other resources that might have higher price points and are not used regularly. LEAD can use Amazon EC2 resources that are priced differently from the TeraGrid resources. The trade-off cost versus benefit of using higher priced resources must be represented in the workflow planning process. Using vgES, we study the impact of scheduling LEAD workflow sets on a mix of TeraGrid, local grid and cloud sites and Amazon EC2 resources. This four phase workflow planning approach is detailed in this section.

The workflow planning approach we describe here specifically targets deadline-sensitive workflows such as LEAD. However the different phases of the workflow planning approach can be applied to other workflows which might be less time-sensitive.

3.2.1 *Resource Procurement*

Each resource site is queried for a certain number of processors for the time duration from now till the deadline. Our workflow orchestration has two goals: to meet the specified deadline and schedule the maximum number of workflows in the given time such as to increase the probability that at least the minimum required workflows complete by the deadline. Thus, we pick an aggressive resource request policy (with no cost constraints) querying all sites for the maximum duration. Once a minimal set of resources are available, the workflow planner instructs vgES to "bind" the resources (i.e., appropriate mechanisms are used to start the procurement of the resources on grid and cloud sites). Resource availability might be delayed due to batch queue wait times or the virtual machine startup overhead. Once resources become available vgES sets up appropriate application software. The multi-phase planning process is initiated in parallel.

3.2.2 *Phase 1: Minimal Scheduling*

The goal of this stage of the pipeline is to schedule the minimum fraction of workflows required on regular resource sites (i.e., we do not use Amazon EC2 in this phase). We implement a simple probabilistic DAG scheduler. The DAG scheduler traverses the DAG bottom-up and assigns deadlines for the tasks given a workflow deadline. Subsequently the tasks are sorted by deadline for the scheduling phase. Each task T has a duration *d* and must be scheduled no earlier than *earliestStartTime* and must finish no later than *latestFinishTime*. The DAG scheduler finds a task mapping on each of the slots returned by vgES for a particular task. Subsequently all task mappings that meet the task deadline are considered for selection and the best success probability mapping is selected. Data transfer times are accounted for when finding a task mapping that can meet a deadline. For any task in a workflow, the probability that it will succeed depends on the resource on which it is scheduled as well as the probability of its parent tasks finishing. When two tasks are scheduled on independent resource slots their probabilities are independent and the probability of a task is the joint probability of its parent and itself. However, in a slot abstraction, if a Task T and its parents are scheduled on the same resource slot then the Task T has the same probability of finishing as its weakest parent. The process is repeated for all tasks in the workflow. If the minimum fraction of workflows cannot be scheduled at this stage, the planner exits with an error. The probability of a workflow completing is the minimum of the success probability of all tail nodes.

3.2.3 *Phase 2: Fault Tolerance Trade-off*

In the trade-off stage we compare scheduling additional workflows with increasing the fault-tolerance of one or more tasks of the scheduled minimum fraction of workflows. We compare the success probability of the minimum fraction of workflows completing as the criteria for picking the scheduling at this stage. Probabilities of tasks completing are computed using the failure probability of the resources and the probabilities of its parent tasks. We maintain a queue of tasks from the scheduled workflows that are sorted by their probability of completion. We compare:

1. a schedule from scheduling additional workflow from the workflow set, and
2. a schedule where a task T_i from the sorted queue of tasks of the scheduled workflow from Phase 1 is replicated one or more times on available resources.

The workflow planner interacts with a fault tolerance component to determine if a task should implement replication in the second case. For this implementation, the fault tolerance component implements replication based fault-tolerance techniques to increase the probability of success for each workflow task. Given the current mapping of tasks on a Gantt chart of available resource slots with corresponding reliability characteristics, each workflow task is replicated on additional slots to increase the probability of success of a task to the desired success probability. This in turn increases the workflow success rate and hence potentially increases the fraction of workflows that need to finish by a given deadline.

The fault-tolerance techniques determine the mapping of the replicated tasks on the available slots and return the mapping to the planner. During this mapping process, we use simple techniques based on joint-probabilities derived from success probabilities of tasks on slots. For each task, a window of replication is determined by the planner, which constrains the replicated tasks to start-after and end-before particular time based on task dependencies. The fault tolerance mapping process tries to fit a task on the available slots in that replication window based on the expected performance characteristics of the task (number of cpus required and expected execution times derived from performance models). If the task fits on a slot, the success probability of the task increases. When the success probability reaches the desired level during this replication process, the mapping of the replicated tasks is returned to the planner. It should be noted that the fault tolerance techniques may not be able to reach the desired success probability by replication in all cases. In such cases, the workflow planner is responsible for changing system parameters (e.g. less compact Gantt chart, additional resources) and iteratively using the fault tolerance techniques.

The schedule that yields the higher success probability from these two approaches is selected and this step is repeated till no additional workflows can be scheduled or all tasks in the original M workflows have been checked to see if fault tolerance can be applied.

3.2.4 *Phase 3: Additional Scheduling*

The goal of this phase is to use the available space on the slots for any other scheduling. If any workflows in the set have not been scheduled in the earlier step, an attempt is made to schedule those. If any tasks have not been checked for fault tolerance in the earlier step, an attempt is made to replicate those tasks to increase its success probability.

3.2.5 *Phase 4: EC2 Scheduling*

Finally, the planner uses Amazon EC2 as a backup resource to determine if additional replication on this higher-priced resource can increase the success probability of individual tasks. Amazon EC2 is the prevailing example of cloud systems. The cloud model lends itself well to adaptive load conditions albeit at a slightly higher cost model. This makes it an ideal choice for using as a overflow resource site for LEAD workflows to meet application requirements.

This orchestration approach enables us to study (a) the trade-offs between replicating certain tasks in the workflow, (b) the implications of using Amazon EC2 as a overflow resource for scientific workflows. We use specific policies (e.g., Amazon EC2 as overflow resource) in this implementation but the multi-phase workflow planning is configurable for other use cases and policy choices [40].

3.3 Execution Manager

Workflow engines execute workflows based on DAG dependencies. However when considering workflow sets the execution plan might include execution dependencies on other workflow's tasks that might be scheduled to run on the same resources. Thus we need execution level support for workflow set execution in slots that respects the order of the DAG and other workflow tasks scheduled on the same slot.

We implemented a simple slot exection ordering mechanism in the Execution Manager (shown in Figure 1b). This ordering mechanism submits jobs to the slot execution batch queue system using the schedule. When the execution manager receives a job it checks to see if all tasks that are scheduled on the slot before this task have been submitted. If not all tasks scheduled before this have been submitted the task is saved in a pending queue for later execution. Events in the system such as job submission and job completion trigger a thread that checks the elements in the pending queue to see if a task might now be ready to run. This ordering mechanism is sufficient to sequence task execution on slots as per the generated schedule. However this ordering mechanism is not completely resistant to failures. If an earlier task fails to arrive, a task will be stuck in the pending queue till it is rectified. Thus, the ordering mechanism depends on an external mechanism such as the monitoring system to diagnose errors and rectify it.

4. EVALUATION

The integrated system demonstrates a powerful resource abstraction over grid and cloud systems that can be used by workflow planning mechanisms for predictable service guarantees. This allows even deadline-sensitive applications such as LEAD to run on non-dedicated machines. However, it is important to understand the different parameters of the system and the tradeoffs with existing systems. We present results that (a) demonstrate acceptable overhead of vgES in executing LEAD workflows on a distributed system (Section 4.1), (b) compare our approach to traditional approaches using historical data from batch queue systems (Section 4.2), and (c) explore via simulation the parameters that impact the fault tolerance (Section 4.3).

Workflows. We submit eight LEAD workflows which are ready to run in five minutes from the start of the experiment. The data set we use for the each of the LEAD workflows is a small regional weather

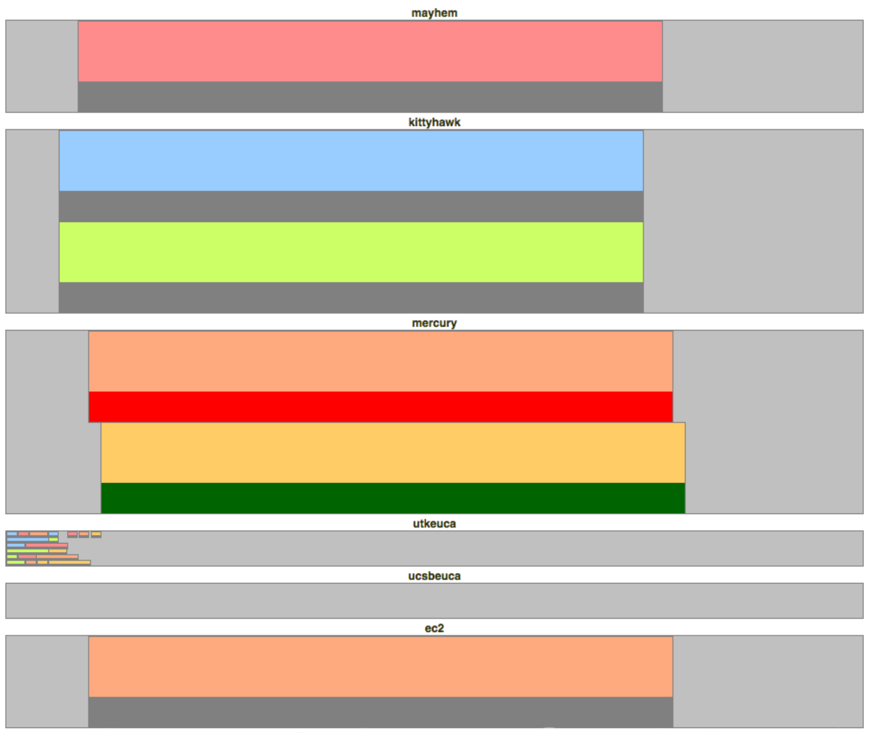


Figure 4: Scheduling an actual LEAD workflow set using VGrADS.

forecast and takes about 90 minutes to complete. The first few steps of the workflow take a few minutes on single processors and the weather forecasting model (WRF [29]) takes over an hour and fifteen minutes on 16 processors. The constraint on the workflow set is that at least one must complete by a deadline of 2 hours. Each of the experiments was repeated between 10 to 15 times.

Testbed. The experiments in Sections 4.1 and 4.2 use the same testbed containing batch and cloud resources. The software consists of an Apache ODE workflow engine, the workflow planner service, the vgES code base and associated databases. The distributed infrastructure consists of batch queue systems run at RENCI / University of North Carolina Chapel Hill (kittyhawk), University of California Santa Barbara (mayhem), and NCSA TeraGrid (mercury); Eucalyptus-based cloud resources at University of Houston (uheuca), University of Tennessee Knoxville (utkeuca)), and University of California Santa Barbara (ucsbeuca); and Amazon EC2 (ec2). For this set of experiments we obtained advanced reservations on the mercury and kittyhawk clusters.

Section 4.3 uses a different set-up. That testbed consists of a virtual machine where our entire software stack is hosted. As we describe in Section 4.3.1, the virtual machine allows us to vary system parameters (e.g. reliability) to study their impact on fault tolerance..

Visual representation. All the experiments we describe began as demonstrations at the SC08 conference. To give a feel for the experiments, Figure 4 shows a screen shot from one run. (Full playbacks of several cases can be found on the web [48].) This run scheduled 5 of the 8 workflows distributed across the six resources. The slot on each resource is represented by a background grey rectangle. Each colored rectangle represents one task, and tasks from the same workflow have the same color. Horizontal length represents (scheduled) time, and vertical height represents number of processors (up to 32). The color bands at the bottoms of tasks represent the task state (e.g. green for running, gray for completed). Here, the scheduler mapped many single-processor tasks from all the workflows to utkeuca (a Eucalyptus cloud). It assigned the WRF models to the larger batch queue clusters (mayhem, kittyhawk, and mercury). One model was replicated (on mercury and EC2) for fault tolerance. As it happened, one replica (on mercury, marked with the red band) did fail but the run met the overall constraints.

4.1 Infrastructure timing metrics

In this section we study the performance of our integrated system. We present the event timeline of the system and compare the resource binding parameters.

4.1.1 *Resource Binding*

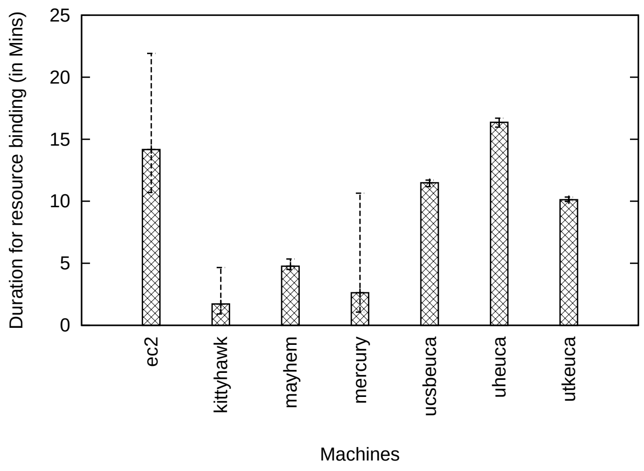


Figure 5: Binding time

Figure 5 shows the average time required for binding each of the sites over nine to thirteen runs. The error bars show the minimum and the maximum values seen in the experiments. The batch systems (kittyhawk, mayhem and mercury) take less time to set up than the cloud sites in our setup since we have reservations in place at the batch sites (kittyhawk and mercury) or the system has lower loads (mayhem) and hence there is no wait time. The Eucalyptus clouds (uheuca, ucsbeuca, utkeuca) and Amazon EC2 grant access to resources immediately but have a set-up overhead since the virtual machines need to be booted with the image. EC2 and uheuca take longer since they boot 16 nodes at each site. There is some variation in the bind time at Amazon EC2, kittyhawk and mercury that is the result of runtime characteristics such as the load on the machine. In this graph, we see that the overheads from cloud computing are slightly higher than from batch systems. However the overhead on all sites is less than 25 minutes, which is acceptable for workflows such as LEAD which run for a number of hours.

4.1.2 Event timeline

Figures 6 and 7 show a snapshot of an experiment timeline. These experiments were repeated a number of times and are repeatable as shown in our archive [48]. Figure 6 shows the timeline of the planning phase of the orchestration. Each line on the graph represents the duration of a particular event where the ends of the line signify the start and end time of the event. In the first step, the planner queries vgES. Once a set of slots is determined to be sufficient, the binding process starts on all the sites in parallel. While the binding is in progress, the planner queries bandwidth across all pairs of sites and launches the phased workflow planning. The four phases take only a few seconds and complete well before the resource procurement is complete. The resource procurement duration varies by site, but resources are ready within 20 minutes. Once the resources are ready, the workflows begin execution (Figure 7). In this snapshot workflow1 failed and hence finished early. All other workflows completed by the deadline of 2 hours as expected. These figures demonstrate the effectiveness of our approach in scheduling workflow sets across distributed resource sites.

We also compared the start and end times of execution with those predicted by the scheduler. The scheduler underestimated both by 13 to 22 minutes. The cause of this is that the slots returned by vgES do not consider the overheads associated with resource procurement and set-up. A simple work-around might be to assume a maximum overhead to the start time of the slots. But in the long term, we need need better prediction mechanisms returned by vgES interfaces, as well as support for dynamic and staggered execution to maximize slot utilization.

4.2 Deadline scheduling

Table 1 compares our scheduler with other strategies for deadline scheduling of workflows on a TeraGrid node. We use historical logs from the batch systems to find average wait times for jobs similar to the tasks in our workflow. In all cases, the queue variability is significant and strategies that do not appear to meet the deadline *might* work at some times. For all the cases, "resource binding" is the delay to begin computing, whether it is due to waiting in a queue or overhead for virtual machine startup. The best case time is running with an advanced reservation, thus avoiding any wait time. However, this requires requesting the reservation far in advance, which is not always practical. Without reservations, current systems like LEAD, insert tasks into a batch queue only when all their predecessors are complete (as Figure 2b showed). This incurs repeated waits in the queue, which the table shows to be more than the actual execution time and causes the workflow to miss the deadline.

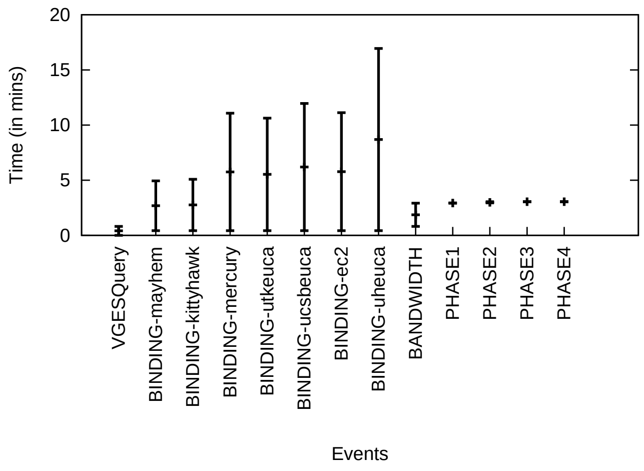


Figure 6: Time line of the planning phase of a run

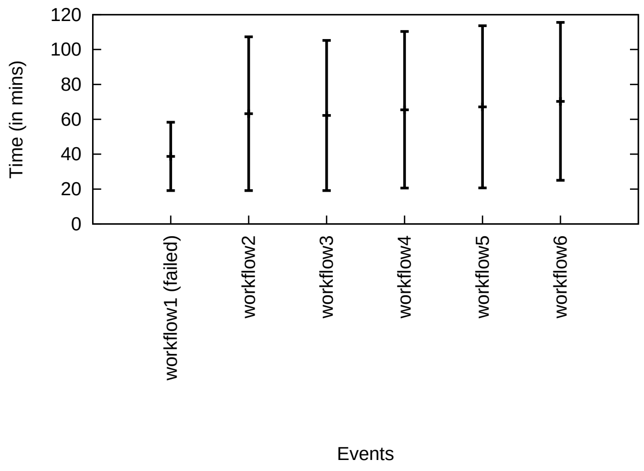


Figure 7: Time line of workflow execution

Another strategy would be to submit a large job for an entire workflow – corresponding to a virtual grid slot – which would run one or more workflows. As the table shows, even submitting a single workflow in this way may not meet the deadline. In fairness, we should note that, according to our data, this strategy would succeed about half the time based on the distribution of queue wait times. Requesting separate slots for six workflows (the number scheduled by our orchestration approach) is worse, and running all eight requires nearly 6 hours to complete. The vgES system queries multiple sites and the workflow planner then schedules the workflows across these distributed sites. By opportunistically procuring resources at multiple sites, vgES is able to complete execution of six workflows by the deadline inspite of the overheads.

4.3 Fault tolerance exploration

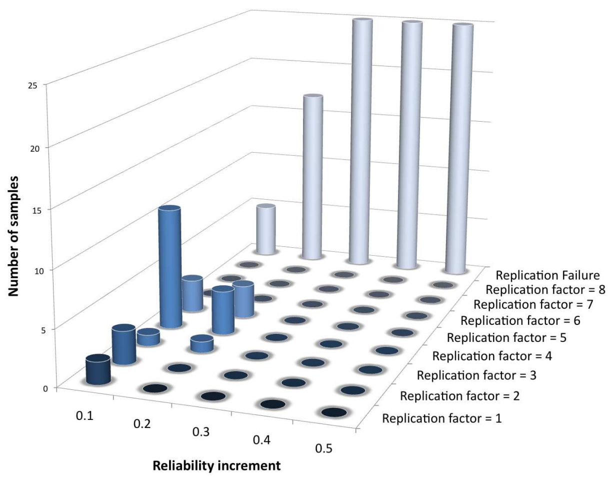
The workflow planner interacts with the fault-tolerance component to determine if and where a task needs to be replicated to increase the fault-tolerance of a given workflow. In this section, we evaluate these fault-tolerance techniques under varying resource (slot) reliability conditions.

4.3.1 Experimental Setup

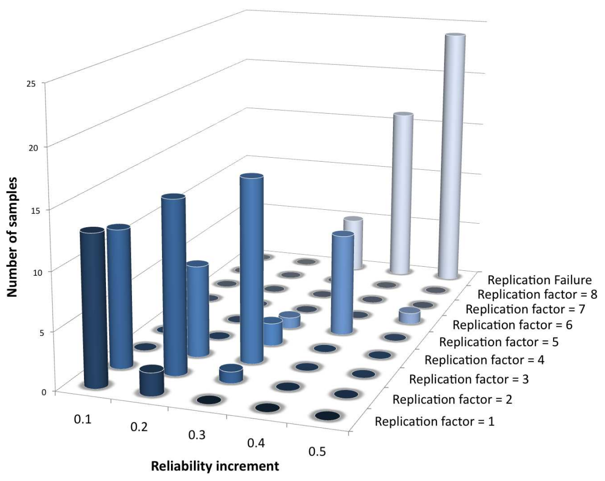
Type	**Resource Set**	**Planning** (s)	**Resource Binding** (s)	**Execution Time** (s)	**Total time** (s)
Batch queue execution time with advanced reservation	mercury (1 workflow)	*	-	5,224	5,224 (1.45 hrs)
Batch queue execution time in current systems (1 workflow)	mercury	-	*5,711*	5,224	10,935 (3.04hrs)
Batch queue single slot execution time (1 workflow)	mercury	-	*2,200*	5,224	7,424 (2.06 hrs)
Batch queue single slot execution time (6 workflows)	mercury	-	*2,830*	5,224	8,054 (2.24 hrs)
Batch queue single slot execution time (8 workflows)	mercury	-	*14,530*	5,224	19,754 (5.49 hrs)
vgES	Mix of batch and cloud resources	183 *	1,017	5,693 (from execution)	6,893 (1.93 hrs)

* TeraGrid users must request advanced reservations at least 48 hours before start of run.

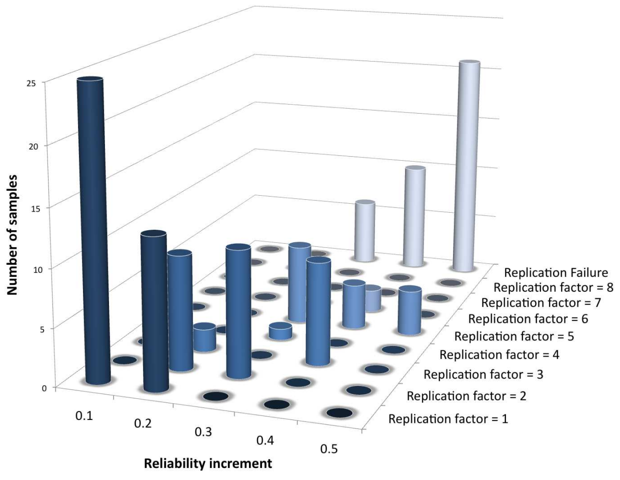
Table 1: Comparison of planning, resource availability overheads and execution time using various methodoligies to schedule workflows/sets. The numbers in italics are projections from historical data for similar sized jobs. The execution times (except the vgES run) are part of our performance model



(a) (slotAcqProb=0.4, slotUpProb=0.4)



(b) (slotAcqProb=0.6, slotUpProb=0.6)



(c) (slotAcqProb=0.8, slotUpProb=0.8)

Figure 8: Distribution of replication factors for slots with *low reliability* (a), *medium reliability* (b), *high reliability* (c)

For each experiment, we generate a Gantt chart for a given time window for replication. The Gantt chart is randomly populated with slots with random numbers of available processors, picked from a given set of possibilities. The slots have (a) a base "Slot Up" probability ('slotUpProb') with a random distribution of around 0.1 (slotUpProb $\pm$ rand() $\times$ 0.1), which determines the probability with which the slot would be available once the slot has been acquired, and (b) a fixed "Slot Acquire" probability ('slotAcqProb'), which determines the probability of acquiring a slot from available resources at a site. The combination of 'slotAcqProb' and 'slotUpProb' determines the probability of task completion on that slot. We execute an initial mapping process (using the DAG scheduler) of a LEAD workflow onto the Gantt chart slots. We then invoke the fault-tolerance component to potentially replicate a workflow task in order to increase the current success probability of the task by an amount denoted by the 'Reliability increment'. In other words, 'Reliability increment' is the difference between current and desired success probability for the given task. We vary 'Reliability increment' from 0.1 to 0.5 in steps of 0.1. If the increment value results in a desired success probability of more than 1, we peg it to the difference between 1 and current success probability. The fault-tolerance component either returns a set of slots to replicate if the desired reliability is obtained, or reports failure. The "replication factor" is the number of added slots required to reach the desired success probability for the given task. We run 25 trials for each increment, 'slotUpProb' and 'slotAcqProb' combination. We define the "replication success rate" as the ratio of number of replication success outcomes to the total number of trials for a case. Low replication factors and high replication success rates are desirable.

4.3.2 Results

Figures 8(a) -8(c) show the distributions of replication factors for different reliability increments for different combinations of 'slotAcqProb' and 'slotUpProb'. Figure 8(a) shows low-reliability resources, implying low values of 'slotAcqProb' and 'slotUpProb' (0.4). We observe that beyond a increment value of 0.2, replication fails for all Gantt charts. For lower increments, replication factors range between 1 to 5. The number of replication failures increases with increasing increments. Figure 8(b) shows medium-reliability slots ('slotAcqProb' = 0.6 and 'slotUpProb' = 0.6). We observe that there are fewer replication failures and lower median replication factors than in the low reliability case. Most cases up to an increment of 0.3 have a replication factor of 4 or less. Figure 8(c) shows the case where the slots have good reliabilities ('slotAcqProb' and base 'slotUpProb' values of 0.8). Here, we observe that there are even fewer replication failures and lower median replication factors than the medium reliability case. The majority of cases up to a reliability increment of 0.4 have a replication factor of 3 or less. We have run other scenarios with different increments and slot reliabilities and the results are similar.

We can infer from these observations that increasing reliability increments increases the median replication factor and the number of cases of replication failures, implying that replication techniques would be effective for moderate reliability increments. We can also infer that higher reliability of slots results in smaller median replication factors and fewer replication failures.

Figure 9 shows the replication success rate for different reliability increments for 'slotAcqProb'=0.7 and different values of base 'slotUpProb'. We observe that replication success rate decreases as reliability increment increases. Also, the success rate falls more rapidly for lower 'slotUpProb' values. We have made similar observations for other combinations of 'slotAcqProb' and 'slotUpProb'. The rate of decrease of replication success rate increases with de-

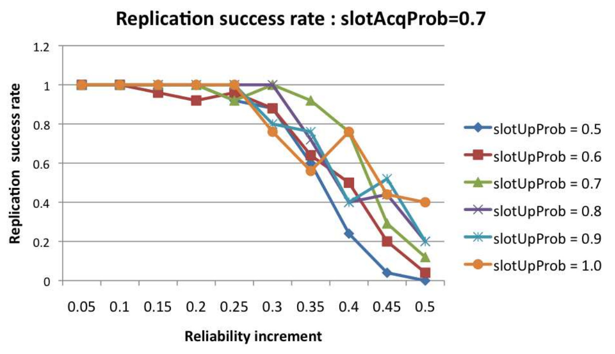


Figure 9: Replication success rate for different slot reliabilities

creasing reliabilities of slots. We believe that increasing the number of trials would smooth out some of the curves. From these observations, we infer that lower reliability increments and higher slot reliabilities are desirable in order to increase the replication success rate.

5. RELATED WORK

This work focuses on various aspects of cyberinfrastructure: execution management, resource provisioning, workflow planning and fault tolerance. However, there is no prior work that integrates grid and cloud sites through a single provisioning mechanism, nor any enabling execution of workflow sets with deadline and accuracy constraints. We provide a summary of related work that have common features with different aspects of our research.

Execution systems. Grid and cloud tools such as Globus [18], Apache Hadoop [19], and Virtual Workspaces/Nimbus [22] provide mechanisms to manage execution in distributed and clustered environments. However, application- and QoS-based resource selection is not possible with these tools. vgES provides a single interface to query and manage execution across distributed sites based on QoS properties of the resources. The idea of containers or glidein has been used before in different projects to manage provisioned resources [17, 38, 45]. Similarly, workflow engines (e.g., Taverna [34], Kepler [1], Pegasus [8]) provide execution-level support for managing workflows on distributed resources [46]. Our approach uses similar support for each workflow but also manages multiple workflows.

Resource Provisioning. The problem of provisioning resources in HPC systems has been well studied and a variety of techniques have been investigated within the context of HPC and Grid computing [14]. Most systems in operation today rely on batch queueing software [27, 47] to implement a space-sharing policy, where applications are given exclusive access to the resources during execution. This so called "best-effort" strategy is sufficient for applications that do not have strict deadlines, but results in hard-to-predict application execution times due to highly variable queue wait times. Several works have attempted to solve this problem by developing methodologies for predicting wait times. Downey [9] develops a strategy for modelling remaining job execution times in order to predict when similar jobs will have access to freed resources. Smith et al. [43] presents a template-based approach to categorize and predict job wait times based on the use of the mean wait times experienced by each category of jobs. These works provide good predictions for collections of jobs, but report large error ranges when used to predict individual job wait times. Nurmi et al. [6, 30] provide an approach to batch queue wait time prediction that results in probabilistic upper bound predictions that have been shown to produce accurate bound predictions for individual jobs.

For users who require a more concrete constraint on application execution completion times, some research has explored the use of advance reservations in HPC job schedulers. However, several researchers have shown that the introduction of advance reservations into HPC scheduling systems can result in significant decrease in site utilization [20, 44]. For this reason, most sites do not provide a general advance reservation facility to the user community.

Workflow planning. Workflow planning techniques today are focused on scheduling individual DAGs and do not consider the relationship between DAGs and constraints associated with scheduling a set of workflows [5, 32, 51]. Various DAG scheduling algorithms have been proposed for grid environments for optimizing various parameters [26, 42, 50]. However, they have neither attempted to orchestrate multiple workflows as required in our application nor balance performance and reliability.

Fault Tolerance. There are several techniques for fault-tolerance [11] of single parallel SPMD applications running on HPC systems, which include FT-MPI [13] and LA-MPI [4]. Reed et al. [41] identify reliability challenges for large-scale HPC systems and provide adaptive techniques for failure detection (using performance contracts) and runtime adaptation for MPI applications. The fault-tolerance methods discussed in this paper leverage those single-site techniques to distributed grid/cloud systems.

One of the prevalent methods for fault-tolerance on computational grid systems is simple retry, as in Condor-G [17] and Gridsolve [49]. The application is resubmitted on a resource in case of a failure. In Condor DAGMan [7] and Pegasus [5], in case of a failure, the remaining portion of the workflow (the rescue DAG) is re-executed. Kandaswamy et al. [21] evaluates migration based techniques for fault-tolerance of workflow tasks on grids. Ramakrishnan et al. [39] uses performability to capture degraded performance from varying resource reliability for scheduling and fault tolerance of workflows. In this paper, we use replication based fault-tolerance techniques based on resource (slot) reliability and application performance models.

6. CONCLUSION

Programming grid and cloud systems for e-Science workflows and managing QoS in these environments is challenging. VGrADS' virtual grid abstraction simplifies these tasks, unifying workflow execution over batch queue systems (with and without advanced reservations) and cloud computing sites (including Amazon EC2 and Eucalyptus). This paper details the vgES implementation of virtual grids and their use in fault tolerant workflow planning of workflow sets with time and accuracy constraints. Our experiments show the efficiency of the implementation and the effectiveness of the overall approach. Taken together, this provides an enabling technology for executing deadline-driven, fault-tolerant workflows. The integrated cyberinfrastructure from the LEAD and VGrADS system components provides a strong foundation for next-generation dynamic and adaptive environments for scientific workflows.

7. ACKNOWLEDGEMENTS

Research in the VGrADS project was supported by the National Science Foundation (NSF) through grant #0331645. Access to the LEAD infrastructure (both hardware and software) was supported by NSF through grant #0331594. Access to TeraGrid nodes was supported by NSF under award #0122296. The authors would also like to thank the LEAD and VGrADS teams for feedback and supporting infrastructure. The authors would like to dedicate this paper to the memory of Ken Kennedy, who led the VGrADS project from its inception to his untimely death.

8. REFERENCES

[1] I. Altintas, C. Berkley, E. Jaeger, M. Jones, B. Ludscher, and S. Mock. Kepler: An Extensible System for Design and Execution of Scientific Workflows, 2004.

[2] Amazon Elastic Compute Cloud (Amazon EC2). `http://aws.amazon.com/ec2/`.

[3] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. Above the Clouds: A Berkeley View of Cloud Computing. Technical Report UCB/EECS-2009-28, EECS Department, University of California, Berkeley, Feb 2009.

[4] R. T. Aulwes, D. J. Daniel, N. N. Desai, R. L. Graham, L. D. Risinger, M. A. Taylor, T. S. Woodall, and M. W. Sukalski. Architecture of la-mpi, a network-fault-tolerant mpi. *Parallel and Distributed Processing Symposium, International*, 1:15b, 2004.

[5] J. Blythe, S. Jain, E. Deelman, Y. Gil, K. Vahi, A. Mandal, and K. Kennedy. Task scheduling strategies for workflow-based applications in grids. In *IEEE International Symposium on Cluster Computing and the Grid (CCGrid 2005)*. IEEE Press, 2005.

[6] J. Brevik, D. Nurmi, and R. Wolski. Predicting bounds on queuing delay for batch-scheduled parallel machines. In *Proceedings of PPoPP 2006*, March 2006.

[7] Condor Team. Dagman metascheduler – `http://www.cs.wisc.edu/condor/dagman`.

[8] E. Deelman, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, K. Blackburn, A. Lazzarini, A. Arbree, R. Cavanaugh, and S. Koranda. Mapping abstract complex workflows onto grid environments. *Journal of Grid Computing*, 1(1):25–39, 2003.

[9] A. Downey. Predicting queue times on space-sharing parallel computers. In *Proceedings of the 11th International Parallel Processing Symposium*, April 1997.

[10] K. K. Droegemeier, D. Gannon, D. Reed, B. Plale, J. Alameda, T. Baltzer, K. Brewster, R. Clark, B. Domenico, S. Graves, E. Joseph, D. Murray, R. Ramachandran, M. Ramamurthy, L. Ramakrishnan, J. A. Rushing, D. Weber, R. Wilhelmson, A. Wilson, M. Xue, and S. Yalda. Service-Oriented Environments for Dynamically Interacting with Mesoscale Weather. *Computing in Science and Engg.*, 7(6):12–29, 2005.

[11] E. N. M. Elnozahy, L. Alvisi, Y.-M. Wang, and D. B. Johnson. A survey of rollback-recovery protocols in message-passing systems. *ACM Comput. Surv.*, 34(3):375–408, 2002.

[12] C. Evangelinos and C. Hill. Cloud Computing for Parallel Scientific HPC Applications: Feasibility of running Coupled Atmosphere-Ocean Climate Models on Amazon EC2. *ratio*, 2(2.40):2–34, 2008.

[13] G. E. Fagg, E. Gabriel, G. Bosilca, T. Angskun, Z. Chen, J. Pjesivac-Grbovic, K. London, and J. Dongarra. Extending the mpi specification for process fault tolerance on high performance computing systems. In *Proceedings of the International Supercomputer Conference (ICS) 2004*. Primeur, 2004.

[14] D. G. Feitelson and L. Rudolph. *Parallel Job Scheduling: Issues and Approaches*, pages 1–18. Springer-Verlag, 1995.

[15] I. Foster and C. Kesselman. *The Grid2*. Morgan Kauffmann Publishers, Inc., 2003.

[16] J. Frey, T. Tannenbaum, M. Livny, I. Foster, and S. Tuecke. Condor-g: A computation management agent for multi-institutional grids. *10th IEEE International Symposium on High Performance Distributed Computing (HPDC-10 '01)*, 00:0055, 2001.

[17] J. Frey, T. Tannenbaum, M. Livny, I. Foster, and S. Tuecke. Condor-g: A computation management agent for multi-institutional grids. *Cluster Computing*, 5(3):237–246, 2002.

[18] Globus. `http://www.globus.org/`.

[19] Hadoop. `http://hadoop.apache.org/core`.

[20] F. Heine, M. Hovestadt, O. Kao, and A. Streit. On the impact of reservations from the grid on planning-based resource management. In *International Workshop on Grid Computing Security and*

Resource Management (GSRM) at ICCS, pages 155–162, Atlanta, USA, 2005. Springer.

[21] G. Kandaswamy, A. Mandal, and D. A. Reed. Fault tolerance and recovery of scientific workflows on computational grids. In *CCGRID '08: Proceedings of the 2008 Eighth IEEE International Symposium on Cluster Computing and the Grid (CCGRID)*, pages 777–782, Washington, DC, USA, 2008. IEEE Computer Society.

[22] K. Keahey, T. Freeman, J. Lauret, and D. Olson. Virtual workspaces for scientific applications. In *SciDAC Conference*, 2007.

[23] Y.-S. Kee and C. Kessleman. Grid resource abstraction, virtualization, and provisioning for time-targeted applications. In *ACM/IEEE International Symposium on Cluster Computing and the Grid (CCGrid08)*, May 2008.

[24] Y.-S. Kee, C. Kessleman, D. Nurmi, and R. Wolski. Enabling personal clusters on demand for batch resources using commodity software. In *International Heterogeneity Computing Workshop (HCW08) in conjunction with IEEE IPDPS08*, April 2008.

[25] Y.-S. Kee, K. Yocum, A. A. Chien, and H. Casanova. Improving grid resource allocation via integrated selection and binding. In *International Conference on High Performance Computing, Network, Storage*, 2006.

[26] G. Malewicz. Parallel scheduling of complex dags under uncertainty. In *Proceedings of the 17th Annual ACM Symposium on Parallel Algorithms(SPAA)*, pages 66–75, 2005.

[27] Maui scheduler home page – `http://www.clusterresources.com/products/maui/`.

[28] G. V. Mc Evoy and B. Schulze. Using clouds to address grid limitations. In *MGC '08: Proceedings of the 6th international workshop on Middleware for grid computing*, pages 1–6, New York, NY, USA, 2008. ACM.

[29] J. Michalakes, J. Dudhia, D. Gill, T. Henderson, J. Klemp, W. Skamarock, and W. Wang. The Weather Reseach and Forecast Model: Software Architecture and Performance. *Proceedings of the 11th ECMWF Workshop on the Use of High Performance Computing In Meteorology*, October 2004.

[30] D. Nurmi, J. Brevik, and R. Wolski. QBETS: Queue bounds estimation from time series. In *Proceedings of 13th Workshop on Job Scheduling Strategies for Parallel Processing (with ICS07)*, June 2007.

[31] D. Nurmi, J. Brevik, and R. Wolski. VARQ: Virtual advance reservations for queues. *Proceedings 17th IEEE Symp. on High Performance Distributed Computing (HDPC)*, 2008.

[32] D. Nurmi, A. Mandal, J. Brevik, C. Koelbel, R. Wolski, and K. Kennedy. Evaluation of a workflow scheduler using integrated performance modelling and batch queue wait time prediction. In *Proceedings of SC'06*, Tampa, FL, 2006. IEEE.

[33] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov. The eucalyptus open-source cloud-computing system. In *9th International Symposium on Cluser Computing and the Grid (CCGrid) - to appear*, 2009.

[34] T. Oinn, M. Greenwood, M. Addis, M. N. Alpdemir, J. Ferris, K. Glover, C. Goble, A. Goderis, D. Hull, D. Marvin, P. Li, P. Lord, M. R. Pocock, M. Senger, R. Stevens, A. Wipat, and C. Wroe. Taverna: Lessons in Creating a Workflow Environment for the Life Sciences: Research Articles. *Concurr. Comput. : Pract. Exper.*, 18(10):1067–1100, 2006.

[35] M. Palankar, A. Iamnitchi, M. Ripeanu, and S. Garfinkel. Amazon S3 for science grids: a viable solution? In *Proceedings of the 2008 international workshop on Data-aware distributed computing*, pages 55–64. ACM New York, NY, USA, 2008.

[36] H. Qian, E. Miller, W. Zhang, M. Rabinovich, and C. E. Wills. Agility in virtualized utility computing. In *VTDC '07: Proceedings of the 3rd international workshop on Virtualization technology in distributed computing*, pages 1–8, New York, NY, USA, 2007. ACM.

[37] L. Ramakrishnan and D. Gannon. A survey of distribted workflow characteristics and resource requirements. Technical Report TR671, Department of Computer Science, Indiana University, Indiana, September 2008.

[38] L. Ramakrishnan, L. Grit, A. Iamnitchi, D. Irwin, A. Yumerefendi, and J. Chase. Toward a Doctrine of Containment: Grid Hosting with Adaptive Resource Control. In *Proceedings of the ACM/IEEE SC2006 Conference on High Performance Computing, Networking, Storage and Analysis*, November 2006.

[39] L. Ramakrishnan and D. A. Reed. Performability modeling for scheduling and fault tolerance strategies for scientific workflows. In *HPDC '08: Proceedings of the 17th international symposium on High performance distributed computing*, pages 23–34, New York, NY, USA, 2008. ACM.

[40] L. Ramakrishnan and D. A. Reed. Predictable quality of service atop degradable distributed systems. In *Journal of Cluster Computing*, 2009.

[41] D. A. Reed, C.-d. Lu, and C. L. Mendes. Reliability challenges in large systems. *Future Generation Computer Systems*, 22(3):293–302, 2006.

[42] R. Sakellariou, H. Zhao, E. Tsiakkouri, and M. Dikaiakos. Scheduling workflows with budget constraints. In S. Gorlatch and M. Danelutto, editors, *Integrated Research in GRID Computing*, CoreGRID, pages 189–202. Springer-Verlag, 2007.

[43] W. Smith, V. E. Taylor, and I. T. Foster. Using run-time predictions to estimate queue wait times and improve scheduler performance. In *IPPS/SPDP '99/JSSPP '99: Proceedings of the Job Scheduling Strategies for Parallel Processing*, pages 202–219, London, UK, 1999. Springer-Verlag.

[44] Q. Snell, M. Clement, D. Jackson, and C. Gregory. The performance impact of advance reservation meta-scheduling. In *6th Workshop on Job Scheduling Strategies for Parallel Processing*, pages 137–153, 2000.

[45] B. Sotomayor, K. Keahey, and I. Foster. Combining batch execution and leasing using virtual machines. In *High Performance Distributed Computing (HPDC)*, 2008.

[46] I. J. Taylor, E. Deelman, D. B. Gannon, and M. Shields. *Workflows for e-Science: Scientific Workflows for Grids*. Springer, December 2006.

[47] Torque home page – `http://www.clusterresources.com/pages/products/torque-resource-manager.%php`.

[48] VGrADS Demo Site. `http://vgdemo.cs.rice.edu/vgdemo/archives.jsp?display=whitelist`.

[49] A. YarKhan, J. Dongarra, and K. Seymour. Gridsolve: The evolution of network enabled solver. In *Proceedings of the 2006 International Federation for Information Processing (IFIP) Working Conference*, 2006.

[50] J. Yu and R. Buyya. Scheduling scientific workflow applications with deadline and budget constraints using genetic algorithms. *Scientific Programming*, 14(3-4):217–230, 2006.

[51] Y.Zhang, A. Mandal, H.Casanova, A. Chien, Y. Kee, K. Kennedy, and C. Koelbel. Scalable Grid Application Scheduling via Decoupled Resource Selection and Scheduling. In *Sixth IEEE International Symposium on Cluster Computing and the Grid (CCGRID'06)*. IEEE, May 2006.