Deadline-Sensitive Workflow Orchestration Without Explicit Resource Control

Lavanya Ramakrishnan[a], Jeffrey S. Chase[b], Dennis Gannon[c], Daniel Nurmi[d], Rich Wolski[d]

[a]*Lawrence Berkeley National Lab, Berkeley, CA*[1]
[b]*Duke University, Durham, NC*
[c]*Microsoft Research, Redmond, WA*
[d]*University of California, Santa Barbara,CA*

Abstract

Deadline-sensitive workflows require careful coordination of user constraints with resource availability. Current distributed resource access models provide varying degrees of resource control: from limited or none in grid batch systems to explicit in cloud systems. Additionally applications experience variability due to competing user loads, performance variations, failures, etc. These variations impact the quality of service(QoS) that goes unaccounted for in planning strategies. In this paper we propose **W**orkflow **OR**chestrator for **D**istributed **S**ystems (WORDS) architecture based on a least common denominator resource model that abstracts the differences and captures the QoS properties provided by grid and cloud systems. We investigate algorithms for effective orchestration (i.e., resource procurement and task mapping) for deadline sensitive workflows atop the resource abstraction provided in WORDS . Our evaluation compares orchestration methodologies over TeraGrid and Amazon EC2 systems. Experimental results show that WORDS enables effective orchestration possible at reasonable costs on batch queue grid and cloud systems with or without explicit resource control.

Key words: grid computing, cloud computing, workflow scheduling, orchestration, deadline-sensitive workflows

[1]The work was performed at Indiana University, Bloomington, IN

1. Introduction

Large scale computations from various scientific endeavors such as drug discovery, weather modeling, and other applications are composed as a sequence of dependent operations or workflows. A number of these workflows have user constraints associated with them including deadline and budget. In addition, these workflow often access shared resources or data and run computations on grid or cloud systems. For example, a weather prediction workflow is triggered by streaming sensor atmospheric data and consists of a number of data-processing steps that use distributed data and resources [8]. This workflow must complete in a timely manner to generate appropriate forecasts and initiate any emergency management measures that might be necessary. Thus deadline-sensitive workflows require careful coordination of workflow tasks with underlying resource behavior to ensure timely completion.

Resource mechanisms and protocols are available today to coordinate grid resources and ensure quality of service (QoS) [5, 6, 27]. There are tools for workflow planning using performance models [3, 10, 18] and execution systems or workflow engines for managing runtime environment of workflows [4, 15]. Today's planning techniques can provide a "yes" or "no" answer to the question of whether a workflow will meet its constraints (e.g., deadline) on a set of resources. However this information alone is insufficient for deadline-sensitive applications such as weather prediction, given the underlying uncertainty in resources. Users are willing to run the workflow so long as the odds of completion are "reasonable". Users are often willing to pay extra or trade-off application requirements to ensure timely workflow completion. Current systems do not allow these trade-offs or speculative scheduling based on QoS properties of the resources.

Grid and cloud systems provide varying degrees of resource control to an end user. Users interact with grid systems by submitting jobs to a batch queue, which executes the job on the user's behalf once enough resources become available. Cloud systems, unlike batch systems, enable *explicit resource control*, i.e., users request specific quantities and types of resources at specific times. Yet users of both these systems cannot expect strong QoS assurances due to both availability and reliability variations of underlying hardware and software services. Additionally, resource systems lack standardized interfaces and workflow tools interact with these systems using ad hoc mechanisms and comparison of QoS capabilities is extremely difficult.

In this paper, we use the term *workflow orchestration* to describe the holistic, coordinated, dynamic and adaptive approach to workflow planning that works with user requirements and variable resource characteristics while being agnostic to specific resource policy or systems. A fundamental research question this paper attempts to address in the context of the WORDS architecture is *how much explicit knowledge of and control over resources is necessary for effective workflow orchestration over grid and cloud systems?* To answer this question, we develop a lowest common denominator resource model that is powerful enough to implement workflow orchestration for deadline-sensitive workflows over systems like batch queue and cloud systems with or without explicit resource control. Specifically, we make the following contributions in this paper:

- We developed the **W**orkflow **OR**chestrator for **D**istributed **S**ystems (WORDS) architecture that facilitates the separation of concerns between resource and application layers in distributed resource environments.
- In the context of WORDS we designed a resource abstraction that consists of a standard set of interfaces and mechanisms required at the resource layer in grid and cloud systems to implement effective and predictable QoS for end users.
- We develop a probabilistic QoS model in WORDS to account for the uncertainty that comes from the resource layer characteristics.
- We evaluate a number of workflow orchestration strategies on top of the resource abstraction in WORDS for deadline-sensitive workflows.

The rest of this paper is organized as follows. We discuss the WORDS architecture and associated resource abstraction in Sections 2 and 3. We explore some *workflow orchestration* approaches for deadline-sensitive workflows atop the WORDS architecture in Section 4. We expand the orchestration approaches to schedule a workflow set with deadline and accuracy constraints in Section 5. Finally, we compare and contrast various workflow orchestration approaches in the context of scientific workflows over grid and cloud computing systems (Section 6).

2. Overview

In this section, we first define the specific problem associated with deadline sensitive workflows and detail the WORDS architecture.

2.1. Problem Description

A *deadline-sensitive workflow* W_x is a Directed Acyclic Graph (DAG) that must complete by a deadline D for the results of the computation to be useful. For example, meteorological workflows need to complete by a certain deadline since they influence emergency response measures.

The workflows have access to resources $R = \{R_1, ..., R_w\}$ across distributed sites. In addition for each task $Task_k$ its execution on resource R_j is given by $[n, T]$ where n is the number of processors required for the task and T denotes execution time of the application.

Resources are available through either grid or cloud mechanisms. Resource procurement is implicit in grid systems. Users submit a job description to a queue managed by batch queueing software such as Maui/PBS. The job waits its turn to acquire resources. When the requested resources become available and the job is at the head of the queue, the job starts executing. The job is killed if it exceeds the requested wall clock time. The users in this environment do not know when exactly their job might start, though more recently there are services [18] that provide the methodology for predicting bounds on the amount of queue wait times. Cloud providers provide stronger guarantees and immediate access to resources through *explicit resource control* since the requests are typically bounded in both time and space. The most prevalent example of a cloud system in operation today is Amazon's EC2 system. In Amazon EC2, resources are accessible to the user almost instantly, with startup time of the instance and image imposing the only delays. In addition there is a diversity between the cost models provided by these systems. However both these systems experience hardware and software failures that makes it hard if not impossible to make strong guarantees as those required by deadline-sensitive workflows.

The goal of the workflow orchestration is to find a schedule for workflow W_x on available resource set R such as to meet the deadline D over diverse resource platforms such as grid and cloud systems. In this paper, we present and evaluate an architecture over grid and cloud resources that provides an effective workflow orchestration for deadline-sensitive workflows. The WORDS architecture enables us to quantify the effectiveness of the

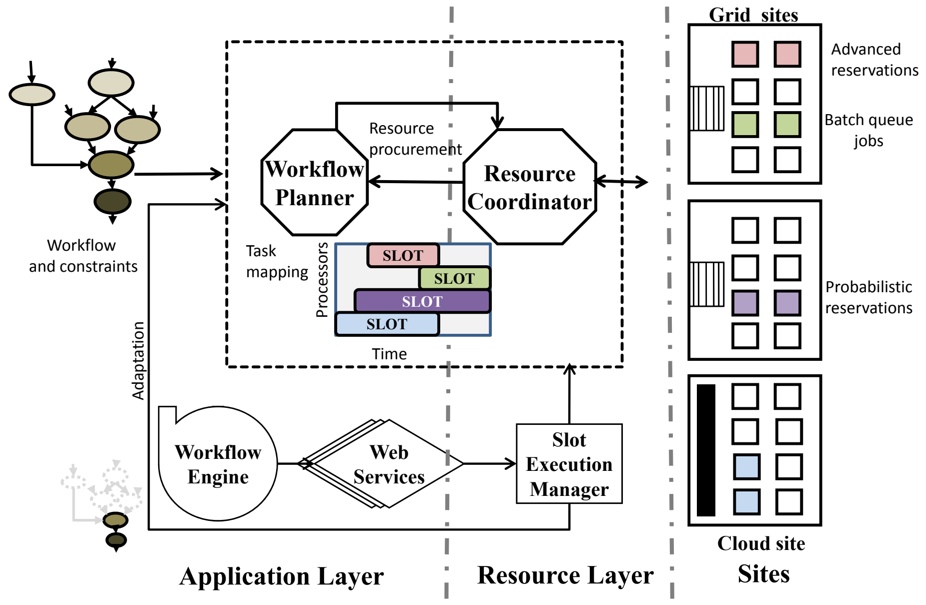


Figure 1: WORDS Architecture introduces a clean separation between resource level and application-level functionalities through a resource abstraction(slot). The workflow planner interacts with the resource coordinator to facilitate resource procurement.

schedule in meeting the deadline. Next, we describe in this architecture in detail.

2.2. WORDS

Figure 1 shows the WORDS architecture that introduces a clean separation between the resource and application layers. WORDS receives a specification of a workflow as a directed acyclic graph (DAG) and user constraints (e.g., deadline). The workflow planner communicates user requirements to the resource coordinator which initiates resource procurement. The resource coordinator interacts with both grid and cloud sites through conventional scheduling mechanisms and interfaces.

The resource coordinator interacts with various site-specific resource control mechanisms and returns a Gantt chart to the application layer. The Gantt chart consists of a set of resource slots from different sites and its associated properties. A *resource slot* is an abstract representation of a resource set on a site that has been assigned to the application or user by the resource layer. A resource slot has defined width (i.e., number of processors) and length (i.e., duration). The *resource slot* is central to our resource abstraction. A slot can be resources allocated to a job through the batch queue system or to a user in cloud systems or through advanced reservation or probabilistic mechanisms (more in Section 3.1).

The workflow planner determines a schedule by assigning tasks on the slots using criteria such as computational time, data transfers, success probabilities, cost, etc (more in Section 4). This process of resource acquisition and task mapping might be iterative with the goal of enhancing the schedule for some or all tasks in the workflow.

The execution system (bottom of Figure 1), consisting of the workflow engine and web services, is largely orthogonal to the orchestration components. The slot execution manager consults the orchestration system for resource related decisions (i.e., where and when should a task run). The workflow planner cannot anticipate all runtime failures that might occur. The WORDS architecture provides resistance to runtime failures through the execution system that is responsible for detecting deviations from the original schedule or other failures. Further discussion on the execution system and handling runtime failures is outside the scope of this paper.

The WORDS architecture provides a dynamic, adaptive resource abstraction that the higher level workflow orchestration can use for planning workflows to meet user constraints. Next, we discuss the resource abstraction provided by WORDS in greater detail.

3. Resource Abstraction

Fundamentally grid and cloud computing systems have different access models and policies. However there are also similarities - resources are assigned to jobs or leases for durations of time; resources are often provisioned across competing user groups and resource requests might not be fulfilled; large scale systems might also experience hardware and software failures. The resource abstraction needs to capture the various dimensions of resource property including cost, policy and variability associated with policy and hardware. The WORDS architecture is based on a least common denominator resource model that abstracts the specific properties of grid and cloud systems. The model captures the common minimal set of of properties across the systems that enables the higher-level workflow orchestration to provide effective QoS guarantees for deadline-sensitive workflows. The model might not capture additional resource properties that might be provided by specific systems. The degree of effectiveness of workflow orchestration over each system varies based on specific resource control policies. For example, if the resource coordinator returns a set of slots from cloud systems that enable explicit resource control (and hence higher levels of resource access guarantees) the workflow orchestration can provide higher-levels of QoS.

The resource abstraction captures allocation properties (i.e., duration, number of processors) cost and QoS properties. Next, we describe these properties in more detail.

3.1. Probabilistic QoS Model

Uncertainty is inherent to distributed systems. Resource providers find it hard to make strong guarantees on resource availability since with or without explicit resource control, strong QoS guarantees cannot be made in distributed systems due to the variability and complexity of the underlying resource characteristics and allocation and access policies. Thus a probabilistic QoS model is a natural choice for these environments.

In our resource abstraction, we define two probabilistic QoS properties on the slot - the probability of resource allocation at the expected time and the probability that the resource will not fail for the allotted duration. The probabilistic resource model allows providers to specify quantitative bounds on resource requests e.g., there is a 95% chance that a request for a three hour slot of 16 processors starting in one hour can be met and there is a 99% chance that resources will stay up during the required duration. These properties are selected based on the behavior of current day systems. Other parameters to quantify the QoS models might be necessary in other environments.

Resource Procurement. Explicit resource control is possible in today's batch systems through offline or online advanced reservations that allow users to specify a fixed start time at higher costs. Thus advanced reservations yield resource slots that have guaranteed start and end times and probability of procurement very close to one. However, advanced reservations mechanisms have shown to have a negative impact on utilization and hence resource providers often allow only limited use of this feature [27, 28]. In addition, TeraGrid sites require requests to be made at least 24 to 48 hours in advance, which is not practical for applications with dynamic loads. Similarly control in cloud systems require resource providers to over-provision resources for peak demand. Thus there are very limited mechanisms available to provide predictable resource control to applications especially those that are deadline-sensitive.

Probabilistic guarantees help resource providers manage the variability in QoS including unexpected load, utilization and other runtime factors. We use probabilistic resource reservations for enabling dynamic workflow orchestration over resources with little or no explicit resource control. As cloud systems advance, the same techniques can be applied to them since lease or cloud resource requests are analogous to job requests with fixed time units [11].

Resource Failure. In addition to resource procurement, hardware and software services have failure characteristics. We use the probability of resource availability as a metric for accounting for variability associated with failures etc.

3.2. Resource Cost models

An important decision factor when it comes to selecting resources is the resource cost. Scientific users are granted access to supercomputing resources through a competitive proposal review process and are allocated "service units" [29]. One SU originally represented one CPU-hour on an IA-64 cluster. A normalization factor is used based on benchmarking results to account for different machine configurations. Compute resources in EC2 [1] are available today as different instance types (e.g., small, large). Users are charged for the closest instance hour consumed. Thus EC2 resources can be considered to be available to users as increments of one hour "leases".

Fundamentally grid and cloud systems use different units for cost. The slot abstraction allows us to explicitly express and compare costs associated with resource procurement mechanisms such as advanced reservations. A direct comparison of the cost from grid and cloud systems for similar type of resources is outside the scope of this paper.

3.3. Implementation of Probabilistic QoS

For our implementation and evaluation of the probabilistic reservations, we use VARQ (Virtual Advanced Reservations for Queues) [19] based resource slots to determine if effective workflow orchestration is possible without explicit resource control in batch systems. A virtual advanced reservations obtained through VARQ is an instance of the resource slot abstraction with probabilistic bounds on obtaining a slot of certain duration by a given time. In overbooked leasing systems we can calculate an equivalent probability using the number of resource lease requests that might be overbooked. VARQ builds on queue wait time prediction techniques from QBETS [18] to give users the ability to request "virtual advanced reservations" i.e., a user can specify a fixed start time for the job. QBETS consumes historical resource request data and makes job completion probability predictions using statistical methods such as a clustering algorithm to categorize similar job requests, an on-line change point detection heuristic to detect abrupt variations in the data, and an empirical quantile prediction technique. Previous studies show that though the queue wait time experienced by jobs is highly variable, the upper bound predictions produced by QBETS are more stable, often over days or weeks. Thus VARQ computes a *probability trajectory*, at 30 second intervals, between the time a user makes a reservation request and the specified deadline and uses the trajectory to find the latest point in time where a resource request can be submitted to meet a specified minimum success probability. Through this methodology, users obtain access to probabilistic or virtual *advanced reservations* that attempt to achieve some level of resource control over systems that provide little or no explicit resource control. The mechanism does have certain cost trade-offs; for example, a resource request might start earlier than the predicted start time and a workflow might not be ready to run. In this case additional resource allocation time will be charged to the user even though the resource might be idle.

VARQ has been evaluated before and shown as a feasible approach of getting a probabilistic guarantee on resources for user jobs in batch queue systems, that do not have any explicit resource control. However, the use of VARQ for workflow scheduling and its impact on workflow orchestration strategies has not been studied in detail before. In our implementation, the workflow orchestration uses a VARQ client to query for alternative slot requests across different sites.

We use the Availability Prediction Service (AVP) [2] to determine the probability that a resource might fail during an allotted duration. AVP uses historical data collected on the systems to determine the probability of failure.

4. Workflow Orchestration

Our resource abstraction is analogous to the internet protocol hourglass model used in computer networks, where irrespective of the specific protocols in the application layer or transport layer, the only protocol used for passing data packets is the internet protocol. The hourglass model allows protocols

Algorithm 1 DAG Scheduler: Probabilistic DAG Scheduler for queue and slot systems

```
*Assign* latest completion times for the tasks by assigning deadlines to each task bottom-up
*Sort* the tasks by latest finish times
**for all** $T$ in $DAG$ in sorted order **do**
  $earliestStartTime \Leftarrow LatestFinishTime(Parents(T))$
  **for** each resource slot **do**
    **if** QUEUE **then**
      $latestFinishTime \Leftarrow Maximum(earliestStartTime + duration, taskDeadline)$
    **else**
      $latestFinishTime \Leftarrow$ find position where task will fit on slot
    **end if**
    **if** task can complete by deadline **then**
      $resourceAcqProb \Leftarrow ProbSlotAcquisition$
      $resourceUpProb \Leftarrow ProbSlotDoesNotFail$
      $taskSuccessProbabilityOnResource \Leftarrow resourceAcqProb * resourceUpProb$
      $taskSuccessProbabilityRelativeToParents \Leftarrow$ calculate task success probability considering placement of parent tasks
    **end if**
  **end for**
  $selectedResource \Leftarrow$ Resource where task has
  $Maximum(taskSuccessProbabilityRelativeToParents)$
**end for**
```

and algorithms in each layer in the WORDS architecture to evolve independent of changes in the other layer and communication between the layers is facilitated through the slot abstraction. Various resource protocols in grid and cloud systems [13, 26, 32] and workflow orchestration algorithms are being investigated for specific applications [16, 25, 31]. However, the resource abstraction and associated QoS model in WORDS provides a uniform knowledge of resource properties that was previously not available to higher-level tools. In this section we detail different workflow orchestration approaches that are possible using the resource abstraction in WORDS for deadline-sensitive workflows as described in Section 2.1. Specifically, we detail the impact of different resource procurement choices for a workflow (Section 4.1). Next, we describe modifications to an existing DAG scheduling algorithm that facilitates the use of the probabilistic QoS model (Section 4.2). We discuss the different orchestration implementations in Section 4.3.

4.1. Resource Procurement

The workflow planner interacts with the resource coordinator to procure resources. In batch systems resource acquisition is closely associated with the execution queue. Jobs are submitted to a queue from which jobs are mapped onto resources. However, as we move to cloud or lease based systems, resource acquisition is a distinct step. Resource procurement for a workflow can be:
Task-based. In a task based strategy, resources are acquired *just-in-time* for each task in the workflow, i.e., for a n-task workflow a separate resource request is made for each task T_1 to T_n. This is similar to the state of the art in workflow grid system where every task is submitted to the queue and waits its turn for execution. Once the task completes and returns to the workflow engine, the next set of tasks are launched In a task-based strategy any overheads associated with a request (e.g., batch queue wait time, virtual machine startup time) is incurred for each task.
Workflow-based. In a workflow based strategy, resources are acquired *prior to scheduling* for the entire workflow, i.e., a single request would be made that will satisfy all tasks T_1 to T_n in the workflow. We need mechanisms to determine appropriate resource requests for the entire workflow. When merging resource requests across different tasks, gaps in the schedule might develop resulting in resource wastage. Resource wastage is an additional cost that must be accounted for in workflow planning. In addition, a hybrid approach that requests for resources for part of the workflows is also possible.

The orchestration system might iteratively query for resources to improve the schedule at possibly higher costs. This is useful in situations where a user might not be satisfied with the initial schedule and might be willing to allocate a higher budget for the workflow execution. Iterative queries might be implemented for parts or the entire workflow. This is facilitated by mechanisms at the resource layer that provide higher QoS guarantees such as advanced reservations.

4.2. Task mapping

Scheduling parallel and distributed workflows on heterogenous resources is a known NP-complete problem and a number of heuristics have been proposed [20, 30]. These heuristics focus on optimizing the makespan of the workflow using projected application running times and data transfer times. However, these strategies are insufficient when users try to understand the properties of a proposed schedule (e.g., cost, chance of meeting the deadline). Thus in this paper we consider task mapping strategies focused on the probability of workflow completion before a deadline as basis for our task mapping approach.

Algorithm 1 describes the probabilistic task mapping approach for batch queue and slot systems. The first phase traverses the DAG from bottom-up and assigns deadlines for the tasks given a workflow deadline and execution time of each task. 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 only difference in the slot based system is that the algorithm tries to find a space on the slot where the task can be mapped. The difference arises from the resource model characteristics. In a batch queue system, requests are bound by the size of the cluster whereas when resource procurement is decoupled from the mapping, the scheduler is bound by the size of the slot.

Subsequently all task mappings that meet the task deadline are considered for selection and the best success probability mapping is selected. 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 parent is scheduled on the same resource slot then the probabilities of completion are identical since the probabilities of resource acquisition and resource failures of the slots are identical for both tasks. The slot acquisition and slot failure probabilities are constant for the allotted duration of the slot. If a Task T has multiple parents, the Task T has the same probability of finishing as its weakest parent. Parent tasks might have different success probabilities based on a number of factors including if their parents were on different slots. The weakest parent is the parent task that has the least success probability for completion. The process is repeated for all tasks in the workflow. The probability of a workflow completing is the minimum of the success probabilities of all leaf nodes.

This task mapping approach takes care of various dimensions of the resource and application characteristics. First, we consider the performance of applications on each resource to determine on which resources, the task is likely to meet its deadline. By using the success probability of resource procurement and the probability that the resource will not fail duration the allotted duration, we are able to pick a resource such as to increase the probability of the task and hence the workflow completing. For deadline-sensitive applications, using the performance or execution time of the individual tasks indicates whether a deadline can be met. However there is no difference to our applications between a workflow that completes a few minutes before the deadline and an hour before the deadline. Thus minimizing the makespan is not the goal of our DAG scheduling approach. Using the probability of getting the resource by the time and its failure characteristics, we are able to gauge the run-time characteristics of the workflow completing by the deadline. While our resource abstraction provides cost information of the slots, cost minimization is not the goal during task mapping for this particular case study. In previous work, we illustrate the use of budget and costs during resource procurement and task mapping [24].

4.3. Orchestration Implementation

Selecting different strategies for resource acquisition, task mapping and scheduling enhancement can result in distinct orchestration approaches with different trade-offs. Table 1 shows the orchestration approaches we implement for comparison and summarize the effects on factors such as makespan and cost that were discussed earlier. For batch systems we compare a traditional batch queue approach as used in systems today (tagged as *BQP*). In the BQP schedule, we identify the "weak" links (i.e., the tasks that have lower probabilities of completion) in the workflow for an initial schedule and use that as a basis for additional resource queries. This schedule enhancement can be implemented at a task-level (tagged as *Task*), workflow-level (*Slot*) or a hybrid approach for a subset of the tasks. Our hybrid approach (referred to as *Boundary*) is based on attempting to get a single advanced reservation on each site for parts of the workflow that are scheduled on it. In this mechanism we consider the earliest task and latest task scheduled on the resource to define the time boundary and the maximum width of any task on the resource as the slot width for the request. On EC2 systems we compare task and slot based implementations. We evaluate these various schemes in the context of scientific workflows.

In this paper, our focus is on deadline-sensitive workflows where users specify a deadline on the completion of the workflow. For such time-sensitive

Table 1: Workflow orchestration implementations. We apply various resource procurement and schedule enhancement strategies on batch grid systems and EC2 cloud systems.

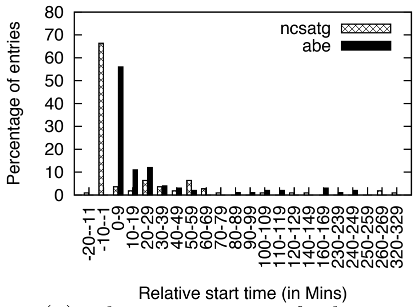
System	Type	Description
Grid/Batch	BQP	Uses batch queue probability prediction data to select resources for each task
	Task	Use BQP scheduler and sort tasks by their success probabilities. Attempt to enhance each task's success by VARQ based advanced reservations
	Boundary	Use BQP scheduler and procure advanced reservations grouping all the mappings on a single resource into a single slot request
	Slot	Query for VARQ slots for entire workflow and apply the probabilistic slot-based DAG scheduler
Cloud/EC2	EC2-Task	Resources procured independently for each task
	EC2-Slot	Resources are acquired for the entire workflow and slot based DAG scheduler is used

applications, cost is often not a limiting factor, hence cost minimization is not a goal for our workflow orchestration approach. The complexity of resource querying and acquisition is at most $O(n)$ and the complexity of the DAG scheduler is $O(n * 2)$. Thus, the worst case complexity for our workflow orchestration approach is $O(n^2)$.

5. Scheduling Workflow Sets

We consider a simple case of scheduling a workflow set with the constraint that M out of N workflows must complete by a given deadline with no fault tolerance. In our case, the workflow set consists of identical members. We implement a simple set of policies using the slot based mechanism to procure resources for a *workflow set* and use the slot based DAG scheduler to meet the constraint of scheduling at least M out of N workflows by a given deadline.

In our implementation, the resource acquisition policy asks for slots for the duration between expected start time and the deadline for the workflow set. We make a resource query that is designed to ask for a resource width


(a) advance request of 1 hour

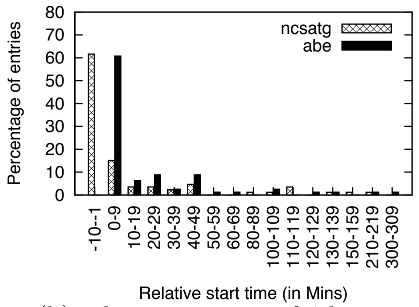
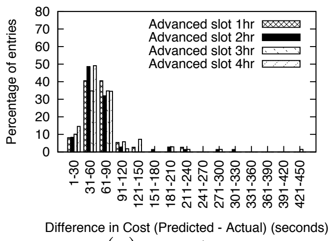
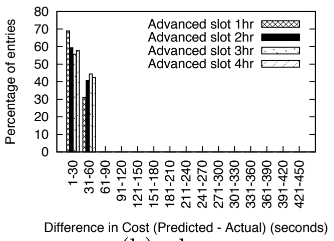

(b) advance request of 4 hours

Figure 2: Probabilistic advanced reservations have variable start times. We show the histogram of difference in actual start times from expected start times on two resources for requests made (a) 1 hour (b) 4 hours in advance. NOTE: Only intervals with entries have been shown in this graph.


(a) ncsatg


(b) abe

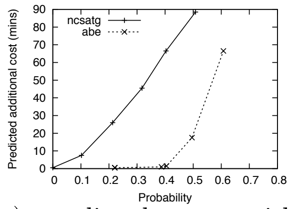

(c) predicted cost with probability

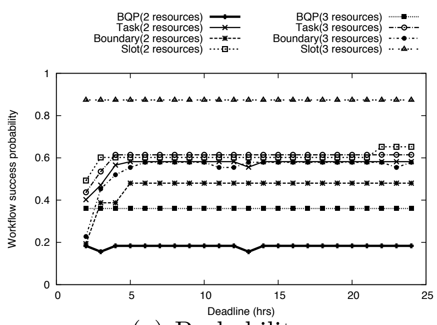
Figure 3: Probabilistic advance reservations incur additional costs if and when they start before expected start time. Here we show the cost variations between the predicted and actual cost over a set of requests on two TeraGrid resources (a) ncsatg (b) abe. (c) shows the effect on cost for different probability values.

that minimally can satisfy the constraint M and possibly more.

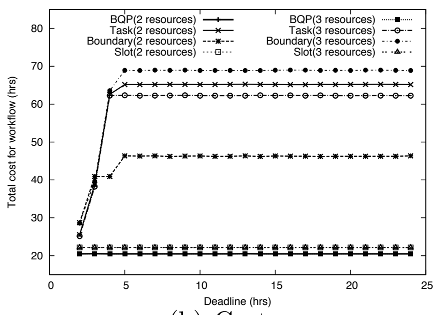
The results from the resource query are sorted by highest success probability and maximum processor width and the best result is picked for the schedule. The maximum number of possible DAGs are scheduled on these slots and we calculate the effective success probability of M-out-of-N workflows completing [22].

6. Evaluation

In this section, we present an experimental evaluation of the resource abstraction and the orchestration techniques in WORDS . First, we perform experiments on probabilistic resource procurement on batch systems to study

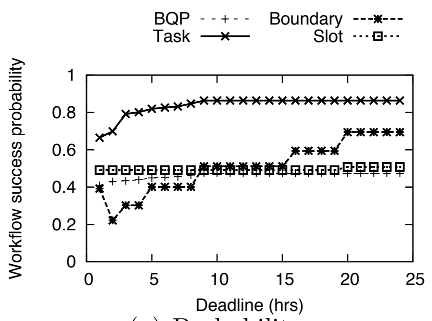


(a) Probability

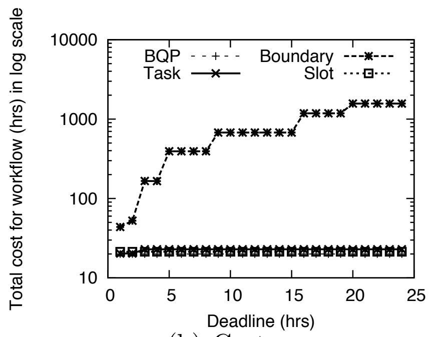


(b) Cost

Figure 4: Comparison of different resource acquisition techniques for the **lead** workflow. We compare (a) the effective probability and (b) cost as deadline varies up to 24 hours. In (b) the costs for BQP - 2 and 3 resource cases are identical and similarly the cost of Slot in the two and three resources in the set are identical



(a) Probability

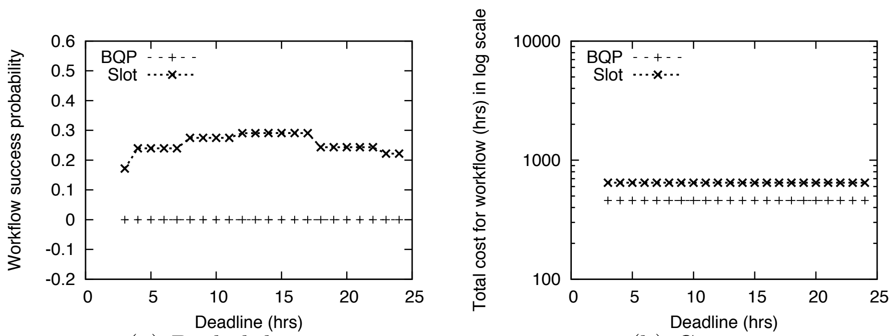


(b) Cost

Figure 5: Comparison of different resource acquisition techniques for **scoop** workflows. We compare (a) the effective probability and (b) cost (shown in log scale) as deadline varies up to 24 hours.

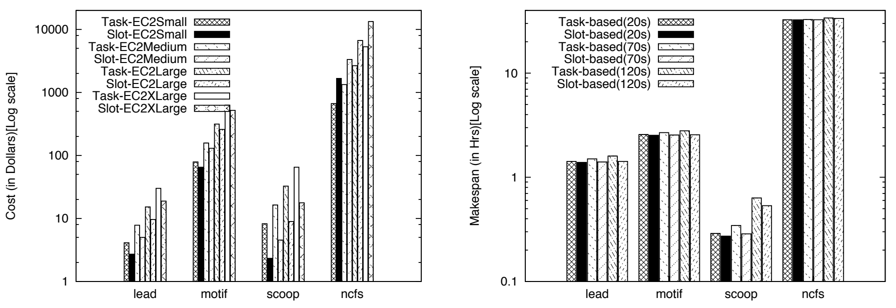
the feasibility of the approach as a means of resource procurement. Next, we compare our orchestration techniques in grid and cloud environments and study the effect on makespan and cost. Finally, we study the effect of user parameters such as deadline and accuracy on scheduling a set of workflows on probabilistic resources.

Our experiments consist of trials performed on the TeraGrid to illustrate the effects of probabilistic reservations on grid systems. The simulations data is from TeraGrid and Amazon EC2 that are leading examples of grid and cloud systems today. Our simulations compare the effect of varying orchestration technique parameters when using grid and cloud resources il-



(a) Probability (b) Cost

Figure 6: Comparison of different resource acquisition techniques for **motif** workflow. We compare (a) the effective probability and (b) cost (shown in log scale) as deadline varies up to 24 hours



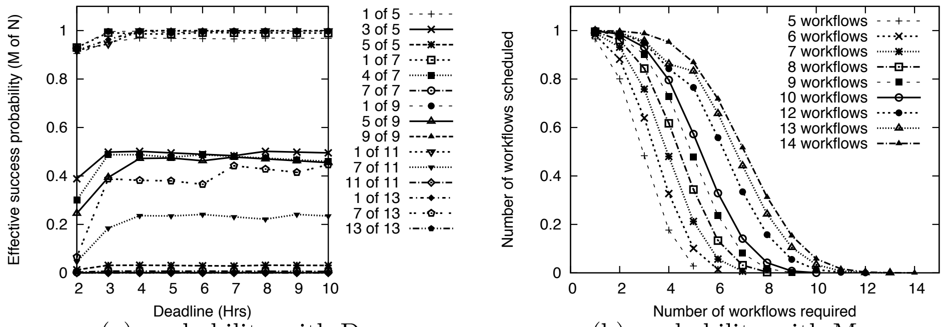
(a) Cost (b) Makespan

Figure 7: Comparison of (a) cost and (b) makespan from task-based and workflow-based scheduling for workflows on Cloud (EC2) resources. The Y axis is in log scale.

lustrating the trade-offs from these environments.

Workflows. We use four grid workflow examples that routinely run on TeraGrid or other high performance systems - **lead**, **motif**, **scoop** and **ncfs** [23]. The workflows have varying characteristics in terms of resource requirements and duration. The weather, storm-surge and flood-plain mapping workflows are time-sensitive.

Machines. For our batch experiments we use probabilistic resource data from three TeraGrid machines (tagged as *ncsatg*, *abe* and *uctg* in this paper). The TeraGrid clusters that have the following specifications: abe - 1200 node quad-core system, ncsatg - 631 node dual processors and uctg - 128 node dual processors. We obtain resources acquisition probabilities through QBETS and reliability probabilities through AVP [2] for failure probabilities.



(a) probability with D (b) probability with M

Figure 8: We apply a slot based workflow orchestration to a workflow set to meet the constraint of at least M out of N workflows must finish within the deadline D. We study the variation of (a) probability with deadline for different M/N (b) variation of effective probability with variation in M for different N values and deadline of 7 hours

We use VARQ to obtain probabilistic advanced reservations.

6.1. Probabilistic Advanced Reservations

In our first experiment we evaluate probabilistic advanced reservations on batch systems. Our experiment requests 90 minute, 16 node slots (the approximate time required for a single LEAD workflow) one, two, three and four hours in advance, with success probabilities ranging from 0.1 to 0.99 on TeraGrid machines *ncsatg* and *abe*. When considering probabilistic advanced reservations, the slots might arrive exactly at, before or after the expected start time. Figure 2 shows the start time variation for one and four hour advance requests over a period of four weeks. Similar results are observed for the two and three hour advance slots. The majority of the experiments start in the [-10,10] minute window around the expected start time. If the slot arrives on time or later, there is no extra cost to the workflow since the workflow is ready to run. However if the slot arrives earlier, the idle time is the extra cost that is incurred due to using virtual advanced reservations for higher level of guarantees. Figure 3 shows the distribution of the difference between the predicted and actual costs for all advanced requests on (a) *ncsatg* and (b) *abe*. The difference in cost between predicted versus the actual cost demonstrates that VARQ and QBETS is able to provide an upper bound on the cost. In all our experiments the cost incurred is always equal to or lower than that predicted by VARQ. The largest percentage of runs have a prediction that is higher by 31 to 90 seconds on ncsatg and by 1 to 60 seconds on abe. Thus the error in the overhead prediction is minimal. Finally, Figure 3(c) shows the variation in predicted cost with probability values. The predicted cost increases as the desired probability value increases i.e., higher levels of guarantee come at higher cost. This is due to the fact that for assuring a higher level of guarantee VARQ might need to submit the job significantly earlier than the desired start time and the chances of the job starting earlier are higher thus resulting in higher costs.

6.2. Orchestration comparison

In this experiment we use simulation to compare the orchestration techniques discussed in Section 4, using a workflow planner simulation. The simulation uses probabilistic data from TeraGrid machines. We recalculate the probabilities for tasks when schedules are enhanced by one or more mechanisms. We use the cost models described earlier(Section 3.2) to calculate the cost for each mechanism. The cost is represented as total number of resource hours used. Thus if a workflow took one hour on 16 processors the cost is represented as 16 hours. On batch systems, resources can be vacated when a job or all jobs on a slot are done, thus incurring no costs for additional slot time at the end of the schedule. Thus to understand the effectiveness of each of our orchestration strategies, we compare success probability of the workflow, makespan, and associated resource usage costs. This enables us to understand the trade-offs between desired success probability, throughput and resulting costs.

The probabilistic advanced reservation technique does have a known limitation; if there are multiple concurrent large resource requests made through VARQ, the queries could potentially perturb the predictions by dominating the workload behavior of the system. The perturbations induced by such requests are being studied separately. Thus for large workflows (motif, ncfs) we compare only the *BQP* and *Slot* mechanisms since the predictions from VARQ are not guaranteed to be accurate. Each run was repeated multiple times over a period of three weeks. The probability predictions are very stable resulting in identical output.

Batch systems, small workflows. Figure 4 shows the probability and cost comparisons for the lead workflow for deadlines ranging from two to twenty-four hours on two (*ncsatg* and *abe*) and three sites (additional site *uctg*). The additional sites has a slightly higher slot acquisition probability resulting in higher effective success probabilities on workflow completion. In these graphs, the success probability is a measure of the chance of the workflow completing by a given deadline given resources can be obtained and resources do not fail during the allotted duration.

For the lead workflow, the *Slot* mechanism assures the highest level of probability among the four techniques. The cost of the slot system is slightly higher than with vanilla *BQP* but considerably lower than both *Task* and *Boundary*-based. We see that there is a slight drop in the success probabilities for a deadline of 13 hours. This variation results from the granularity of the parameter sweep in the heuristic used in VARQ queries. We ask for slots in the probability range of 0.1 to 0.99 and the VARQ query divides the parameter range into 10 equal-sized parts and steps through to find a suitable result. A static advanced reservation on the TeraGrid for a 16 processor, 1.5 hour slot for LEAD workflow would cost anywhere from 24 CPU hours to 48 CPU hours (for premium factors of 1 to 2). The *Slot* based mechanism costs less than that.

Figure 5 shows the probability and cost comparisons for the scoop workflow for deadlines ranging from one to twenty-four hours. In this case, using *Task*-based slots for the individual tasks yields a higher probability than trying to get one big slot for the five parallel tasks. The *Boundary* slot also yields higher probability values for deadlines that are higher than 15 hours. In terms of cost, however, the boundary slots are more expensive (about 100 to 1000 hrs) compared to less than 25 hours for other mechanism. The static advanced reservation for 80 processors/17 minutes would cost between 22 and 44 CPU hours for this workflow and the *Slot* mechanism is on the lower end of this range.

Batch systems, medium and large workflows. Figure 6 shows the probability and cost comparisons for the motif workflow for the *BQP* and *Slot*-based mechanisms. The success probability of the workflow from a *Slot*-based system is higher than the *BQP* schedule. However as the deadline increases we see that the probability drops as a result of the reliability prediction for a 256 processor slot dropping. The *Slot* mechanism has a steady cost that is slightly higher than *BQP*. We compare the BQP approach with slot-based approach for the ncfs workflow for a 36 hour deadline(Table 2). While the success probability from the *Slot* mechanism is slightly higher, the costs are also higher.

These experiments show that trying to procure resources individually for each task that is the current state of art, results in a very low probability of completion of the workflow by the deadline. Using a slot based orchestration, possible through the resource abstraction, enables us to increase the

Table 2: Comparisons for an ncfs workflow scheduled for a deadline of 36 hours

	BQP	Slot
Probability	0.0037	0.0066
Cost (Hours)	5631.5	16640.4

probability of workflows completing by the deadline. Thus WORDS based scheduling approaches results in a more effective workflow orchestration. Traditional workflow planning mechanisms that use performance based planning have no knowledge of what the probability of workflow completion are. Our experiments with different sized workflows also show that the benefits of the slot mechanism might be slightly diminished as the parallelism and the duration of the tasks increases beyond a certain amount. However this effect will vary based on the workflow size, size of resources and the load on the resources. More detailed experiments might be needed in the future to study the correlation of these factors, but the results from this paper demonstrate that WORDS provides a strong foundation for more effective orchestration than current day systems for deadline-sensitive workflows.

Cloud (EC2). Cloud systems today implement *explicit resource control.* However they do have distinct overheads and cost models that affect the nature of workflow orchestration. For this set of experiments we assume EC2 systems have high acquisition (0.9999) and success probabilities (0.9999). We compare and contrast a *Task*-based and *Slot*-based policy. We calculate the EC2 costs for the instance-hours used by the workflow. We consider both computational costs (for different instance sizes) as well as data transfer costs for input and output data transfers to and from the cloud.

Figure 7(a) shows that for all instance sizes, the slot-based system incurs lower cost than a task-based mechanism for the lead, motif and scoop workflows. However for the ncfs workflow, where each task executes for many hours, leaving resources idle in the slot system increases the cost significantly compared to the task-based approach. Earlier experiments reveal that startup overhead for a small instance image varies from 20 to 30 seconds for 1 to 8 virtual machines [21]. We compare the makespans for overheads of 20,70 and 120 seconds. Figure 7(b) shows the effect of startup and shutdown overheads on the makespan. In the task-based strategy the startup and shutdown overheads get added to each task's execution time. Our results show that the slot based system produces better makespans than the task-based systems. As the overheads increase, the difference also increases, as expected.

6.3. Workflow Sets

We perform a set of experiments with the **lead** workflow set to meet the constraint that minimally M out of N workflows must complete by deadline D. We assume workflows are scheduled for a start time that is 12 hours which is a reasonable time frame for advanced reservation requests. We explore the variation of the following parameters- effective success probability, deadline, M and N. Figure 8(a) shows the variation in the effective success probability of getting M out of N workflows with deadline and different M/N pairs. For short deadlines, limited resource time is available and we see slightly lower success probabilities. As expected, the success probability achievable increase as the deadlines are further out and remains fairly steady thereafter. For a given workflow set with N workflows, as M (the required number of workflows) increases we see that the effective success probability decreases. We observe that for short deadlines, the number of workflows scheduled is often less than N (the total), however at larger deadlines, all N workflows are scheduled. Finally, Figure 8(c) shows the variation in the effective success probability with varying M at a deadline of 7 hours. We see that there is a rapid decrease in probability as M increases for a given N since as more workflows are required to complete the guarantee that the system can make is lower that all the required ones will complete. Thus by changing the deadline and the value of M the user can determine various schedules that meet the user's needs.

6.4. Summary

From our evaluation we see that probabilistic resource decisions help us understand the possibility of meeting a workflow deadline. Effectiveness of workflow orchestration varies based on workflow characteristics (e.g. structure, deadline) and resource availability. Generally, slot based acquisition works well for our small and medium sized (lead,scoop, motif) workflow examples on both batch and cloud systems. For our larger sized workflow example (ncfs) the benefits are not substantial due to increased costs. In addition, we show that our probabilistic reservations accurately predict an upper bound on the additional cost. The probabilistic reservations cost lesser than advanced reservations on TeraGrid systems. By using slot-based orchestration approaches we are able to achieve higher-levels of guarantee and hence more *effective workflow orchestration* atop grid and cloud systems even when there is *no explicit resource control.*

7. Related Work

Today, workflow management and resource management tools are focused on scheduling individual DAGs largely towards managing turnaround time of the workflow and do not consider other QoS properties. . However there are no tools or systems available today that can harness a distributed set of resources and trade-off multiple resource selection QoS parameters such as cost, performance and reliability. Here we describe the related work to our workflow orchestration system.

Resource Management. Several research efforts have proposed bounded resource units such as leases, slices, advanced reservations, etc [6, 9, 12, 13]. These abstractions define properties for time and resource information but have little or no QoS information. The concept of decoupled resource selection and scheduling [32] and the slot abstraction [12, 26] has been discussed earlier. However, to our knowledge, the interaction and the interfaces between the application layer requirements and resource model variability and their impact on high-level workflow orchestration have not been studied before.

Workflow Scheduling. Workflow tools provide planing and optimization techniques and fault tolerance to react to changes in grid resource and services performance and reliability. The tools do not consider the spectrum of QoS issues that arises from the interplay of user constraints and resource behavior. Mandal et. al [17] propose a heuristic strategy using performance model based in-advance scheduling for optimal load-balancing on grid resources using the GrADS infrastructure [14]. Blythe et. al. [3] identify and evaluate two resource allocation strategies for workflows - task-based and workflow-based. The task-based algorithm greedily allocates tasks to resources. Workflow-based algorithms find an optimal allocation for the entire workflow and perform better for data-intensive applications.

Various DAG scheduling algorithms have been proposed for grid environments for optimizing makespan, meeting deadline and/or budget constraints or dealing with uncertainty [16, 25, 31]. The underlying assumption of all these algorithms is that resources are guaranteed to be available at a given time, whereas resource availability is highly variable. Deelman et al. [7] detail the computational and storage costs of running the Montage workflow on Amazon EC2 resources. Our approach is orthogonal and is focused on probabilistic resource acquisition and workflow mapping for deadline-sensitive applications.

Heuristic techniques are often used to qualitatively select and map resources to available resource pools and optimize one ore more resource selection criteria such as performance and reliability. However these techniques are insufficient for making complex trade-off decisions between one or more workflows.

8. Conclusions

In this paper we present the WORDS architecture that provides a clean separation between resource and application layer for deadline-sensitive workflow orchestration. The core of the WORDS architecture is a probabilistic QoS based resource abstraction that enables higher-level tools to implement effective workflow orchestration across systems with different levels of resource control. We design, implement and evaluate task-based and workflow-based orchestration algorithms in the context of the WORDS architecture. A workflow-based dynamic resource acquisition and planning strategy works well for all workflows in our example set on both cloud and grid systems but sometimes at a higher cost. Experiments demonstrate that *effective orchestration* is possible even on batch queue systems that have *no explicit resource control* through slots implemented with virtual advanced reservations. WORDS provides a strong foundation for dynamic, adaptive next-generation workflow orchestration in distributed systems.

9. Acknowledgements

This work was funded through a National Science Foundation(NSF) cooperative Agreement issued to Rice University (No. CCR-0331645) with a sub agreement to the University of North Carolina at Chapel Hill and through NSF Cooperative Agreements ATM-0331480. This research was supported in part by the NSF TeraGrid resources. This work was supported by the Director, Office of Science, of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231.

References

[1] Amazon Web Services `http://aws.amazon.com/`.

[2] Availability Prediction Service. `http://nws.cs.ucsb.edu/ewiki/nws.php?id=Availability+Prediction+Service`.

[3] 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 *CCGRID*, pages 759–767, 2005.

[4] Condor DAGMan. `http://www.cs.wisc.edu/condor/dagman/`.

[5] K. Czajkowski, I. Foster, and C. Kesselman. Agreement-based Resource Management, 2005.

[6] K. Czajkowski, I. Foster, C. Kesselman, V. Sander, and S. Tuecke. SNAP: A Protocol for Negotiating Service Level Agreements and Coordinating Resource Management in Distributed Systems, 2002.

[7] E. Deelman, G. Singh, M. Livny, B. Berriman, and J. Good. The Cost of Doing Science on the Cloud: The Montage Example. In *Proceedings of SC'08*, Austin, TX, 2008. IEEE.

[8] K. K. Droegemeier and et. al. Service-Oriented Environments for Dynamically Interacting with Mesoscale Weather. *Computing in Science and Engg.*, 7(6):12–29, 2005.

[9] I. Foster, A. Roy, and V. Sander. A Quality of Service Architecture that Combines Resource Reservation and Application Adaptation, 2000.

[10] Y. Gil, E. Deelman, J. Blythe, C. Kesselman, and H. Tangmunarunkit. Artificial Intelligence and Grids: Workflow Planning and Beyond. IEEE Intelligent Systems, special issue on e-science, Jan/Feb, 2004.

[11] L. E. Grit. *Extensible Resource Management for Networked Virtual Computing*. PhD thesis, Durham, NC, USA, 2007.

[12] D. Irwin, J. Chase, L. Grit, A. Yumerefendi, D. Becker, and K. G. Yocum. Sharing Networked Resources with Brokered Leases. In *USENIX Annual Technical Conference*, pages 199–212, 2006.

[13] Y.-S. Kee, D. Logothetis, R. Huang, H. Casanova, and A. Chien. Efficient Resource Description and High Quality Selection for Virtual Grids. In *Proc. of 5th IEEE Symp. on Cluster Comp. and the Grid*, 2005.

[14] K. Kennedy and et. al. Toward a Framework for Preparing and Executing Adaptive Grid Programs. In *Proceedings of NSF Next Generation*

Systems Program Workshop (International Parallel and Distributed Processing Symposium 2002), Fort Lauderdale, FL, April 2002.

[15] B. Ludscher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger, M. Jones, E. Lee, J. Tao, and Y. Zhao. Scientific Workflow Management and the Kepler System, 2005.

[16] 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.

[17] A. Mandal, K. Kennedy, C. Koelbel, G. Marin, J. Mellor-Crummey, B. Liu, and L. Johnsson. Scheduling Strategies for Mapping Application Workflows onto the Grid. In *High Performance Distributed Computing (HPDC 2005).*, pages 125–134. IEEE Computer Society Press, 2005.

[18] 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.

[19] D. Nurmi, J. Brevik, and R. Wolski. VARQ: Virtual Advance Reservations for Queues. *Proceedings 17th IEEE Symp. on High Performance Distributed Computing (HDPC)*, 2008.

[20] 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.

[21] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov. Eucalyptus:A Technical Report on an Elastic Utility Computing Archietcture Linking Your Programs to Useful Systems. Technical Report 2008-10, University of California, Santa Barbara, California, August 2008.

[22] G. Radke and J. Evanoff. A Fast Recursive Algorithm to Compute the Probability of M-out-of-N events. In *Reliability and Maintainability Symposium, 1994. Proceedings., Annual*, 1994.

[23] 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.

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

[25] 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.

[26] G. Singh, C. Kesselman, and E. Deelman. Application-level Resource Provisioning on the Grid. In *Proceedings of 2nd IEEE Intl Conference on e-Science and Grid Computing*, Amsterdam, 2006. IEEE.

[27] W. Smith, I. Foster, and V. Taylor. Scheduling with Advanced Reservations. In *Parallel and Distributed Processing Symposium (IPDPS 2000)*, pages 127–132.

[28] 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.

[29] TeraGrid Service Units. `http://kb.iu.edu/data/ascf.html`.

[30] N. B. Tracy D. Braun, Howard Jay Siegel. A Comparision of Eleven Static Heuristics for Maping a Class of Independent Tasks onto Heterogeneous Distributed Computing Systems. J. of Parallel and Distributed Computing 61, 810-837., 2001.

[31] 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.

[32] 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. CCGrid, May 2006.