

# A Simulation Toolkit to Investigate the Effects of Grid Characteristics on Workflow Completion Time

Michael O. McCracken  
University of California, San Diego  
mike@cs.ucsd.edu

Allan Snaveley  
University of California, San Diego  
allans@sdsc.edu

## ABSTRACT

Advances in technology and the increasing number and scale of compute resources have enabled larger computational science experiments and given researchers many choices of where and how to store data and perform computation. Analyzing the time to completion of their experiments is important for scientists to make the best use of both human and computational resources, but it is difficult to do in a comprehensive fashion because it involves experiment, system and user variables and their interactions with each configuration of systems. We present a simulation toolkit for analysis of computational science experiments and estimation of their time to completion. Our approach uses a minimal description of the experiment's workflow, and separate information about the systems being evaluated.

We evaluate our approach using synthetic experiments that reflect actual workflow patterns, executed on systems from the NSF TeraGrid. Our evaluation focuses on ranking the available systems in order of expected experiment completion time. We show that with sufficient system information, the model can help investigate alternative systems and evaluate workflow bottlenecks. We also discuss the challenges posed by volatile queue wait time behavior, and suggest some methods to improve the accuracy of simulation for near-term workflow executions. We evaluate the impact of advance notice of predictable spikes in queue wait time due to down-time and reservations. We show that given advance notice, the probability of a correct ranking for a sample of synthetic workflows could increase from 59% to 74% or even 79%.

## 1. INTRODUCTION

In this paper we discuss the analysis of time to solution for large-scale computational experiments. Competition for limited computational resources is a common problem in improving time to solution, and improving understanding of the impact of this competition on a given workflow is the motivation for this work.

Time to solution is affected by many factors, including system performance, local site queues, and network bandwidth. Therefore, computational scientists are faced with a complex optimization space when evaluating the efficiency of their workflow. They can

attempt to reduce their time to solution by requesting allocation on alternative systems. They could potentially use a different queue priority, make reservations, and even modify their workflow. However, the information needed to decide among these tactics to reduce time to solution can be difficult to obtain, and evaluating many combinations of possibilities including different systems and workflows is difficult. Expert users can guess well based on experience with familiar systems, but exhaustively evaluating the space of options, including unfamiliar or currently unavailable systems, is very difficult. Anecdotal evidence suggests that users are overly conservative in making these decisions, and often choose to stick with a familiar system instead of one that may allow them to complete their experiment much sooner [37].

We suggest that simulation tools can allow scientists to rapidly compare potential variations of an experiment, such as system changes or changes in job size and input/output requirements. This results in better-informed decisions and increased scientific productivity, as measured by reduced overall time to solution.

In this paper, we review factors affecting time to completion for large experiments in section 2. In section 3 we introduce LED, a lightweight experiment description language that is designed for ease of prototyping potential changes in experiment workflow such as using new systems or changing job sizes. In section 4 we discuss a discrete-event simulation toolkit for building models of computational science experiments that uses LED as input, and provides ways to integrate system-specific parameters for network performance and queue wait time. In section 5, we show that a presented model, built using our toolkit provides useful information about the effects of system characteristics and experiment time to completion that will enable users to evaluate choices that can both accelerate research progress and productivity. We discuss related work in section 6. We conclude with notes on future directions in section 7.

## 2. MODELING WORKFLOWS

A computational science experiment can range from a short simulation on a single workstation to a large-scale, multiple-job effort requiring many people and institutional resources. In this paper we focus on large-scale experiments involving highly parallel code running on managed, shared clusters, such as those in the NSF TeraGrid [19].

The workflows we discuss in this work are relatively simple compared to the complex examples that motivate the development of workflow management and execution tools. However, many heavy users of computational resources do have such comparatively simple workflows. These workflows are not so simple that they can not benefit from analysis, because even a basic "pipeline" workflow at large scale likely involves multiple potential sites, large data transfers between computational jobs, and visualization steps to

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

WORKS09 November 15, 2009, Portland Oregon, USA.

Copyright 2009 ACM 978-1-60558-717-2/09/11 ...\$10.00.

verify correct progress.

The potentially large number of configurations of available resources introduces a great deal of complexity into the task of “running a simulation”. For example, a research group may have allocations on many compute resources with differing policies, and so every time a job is submitted, a choice is made either implicitly or explicitly to use a particular system at one of potentially many priority levels. Most codes can run at a variety of scales, and often a trade-off between performance and allocation cost is implied in the choice of a scale to run. Then follows a choice about where to store data, and what resource to use for generating visualizations that are commonly required to evaluate results.

We have built a model of the process of running simulations at large scale, which allows a scientist to evaluate the impact of these decisions systematically. High demand for limited resources is the common thread in the parameters of our model. Time dilation, the factor increase in time to completion over actual job run time, is typically  $10\times$  on TeraGrid resources [33]. Computational performance is not always the controlling factor of overall time to solution [37], and available differences in per-job performance between systems may affect an experiment less than variability in queue wait time [24]. Our model focuses on bottlenecks such as queue wait time, data transfer speed between sites, and visualization steps. The following sections discuss a characterization of these bottlenecks as parameters of our simulation framework.

## 2.1 Data Transfer

Large scale computational science experiments often generate large data sets which must be transferred between systems for analysis, and on to archival storage before the experiment is complete. At large scale, this can be a significant factor in time to solution. However, compute-performance-focused models do not address data transfer outside the program, and the cost of moving data can vary widely, making it important to consider when evaluating several systems.

The TeraGrid SpeedPage [26], a web archive of daily GridFTP [12] performance experiments, is a good reference for network performance seen in production HPC systems. It reports results from an automated benchmark that performs GridFTP transfers of about 4 GB between TeraGrid sites, occurring twice daily. A December 2008 analysis of SpeedPage data showed that incoming transfer bandwidth between TeraGrid systems ranges from 32.9 MB/s at the 25<sup>th</sup> percentile to 69.17 MB/s at the 75<sup>th</sup> percentile.

As an example of data transfer bottlenecks in large experiments, Enzo, a cosmology simulation program [25], produced approximately 13 Terabytes of output data per 18-hour job in the 2007 INCITE experiment, writing a 770 Gigabyte data set every 45 to 60 minutes of compute time. The runs used all 6,080 processors of NERSC Seaborg, an IBM Power-3 system with 44 Terabytes of shared storage [20]. The data was required to be visualized and saved to archival storage, each at a different location, after each simulation run [13]. Thus, depending on available scratch disk space at each location, transfer rate could limit the rate of progress of the computation.

A recent article showed that at currently measured internet speeds of 20 Mbits/s, physically shipping disks overnight has a  $45\times$  latency advantage over using WAN links for a dataset of 10 terabytes [2]. Furthermore, network cost-performance has only increased  $3\times$  from 2004 to 2009, while disk capacity and performance have improved  $10\times$  [2], and a similar  $10\times$  improvement can be found in CPU processing power over the same time scale [17]. Although recent tests of high-end networking equipment have shown bandwidth of up to 7.48 Gbps using InfiniBand WAN extensions [28], improvements

in network performance still may not keep pace with disk capacity and processor performance.

## 2.2 Batch Job Queues

Batch job queues are among the most visible and often-discussed productivity problems in HPC [33]. It is common knowledge that sharing resources incurs significant overhead in an experiment. When viewed in terms of overall completion time, it can be more of a factor than computational performance differences between systems. In a study using batch queue prediction to generate schedules for a grid workflow, Nurmi et al. show that using queue information alone resulted in schedules with makespans about three times faster than schedules produced with only computational performance information [24].

To cope with long queue wait times, a user may be able to use a different system, request special priority, or alter the job size. Larger jobs reduce the number of jobs that must go through the queue, while smaller jobs may increase the likelihood of early execution due to back-filling. However, experience shows that due to uncertainty about possible benefits, users are often reluctant to switch systems, despite potentially large reductions in time to solution.

## 3. EXPERIMENT DESCRIPTION

In order to allow users to quickly describe their workflows for input to a simulation model, allowing them to analyze how the workflow could be affected by these bottlenecks, we have developed a language called Language for Experiment Description, or **LED**. A workflow described in LED format includes only the performance-critical components of a planned experiment.

Each experiment is described as a sequential list of tasks or parallel task sets. Lists represent tasks with sequential dependencies, and sets represent tasks that are independent of each other. Descriptions can be nested, so that one can describe an independent set of lists which contain sequentially dependent tasks. For convenience, the language permits finite loops, but after processing, the input to the simulator is a directed, acyclic graph (DAG). For complex workflows, it is also possible to describe an arbitrary DAG directly in LED with dependency lists, but for many common patterns it is simpler to use the list and set notation. It should take about the same amount of time as a conversation or short email with a colleague to describe the experiment workflow in LED.

LED models workflows at a higher level of abstraction than executable workflow languages such as Swift [39], and other workflow systems that describe DAGs of tasks, such as Kepler [15] and Pegasus [8, 7]. Our focus is on quickly describing the parameters of a workflow that most affect its time to completion, so LED descriptions do not include enough detail to manage actual scheduling and execution of a workflow, with the provenance, fault tolerance, and administrative issues that implies.

It is possible to describe real complex experiments using LED. As an example, we have fully described the WRF Nature Run [18] experiment, which involved many stages of spin-up using several machines, eventually resulting in a calculation of record scale and resolution. This experiment description is 47 lines of YAML and was put together over a short email exchange with the lead scientist on the project.

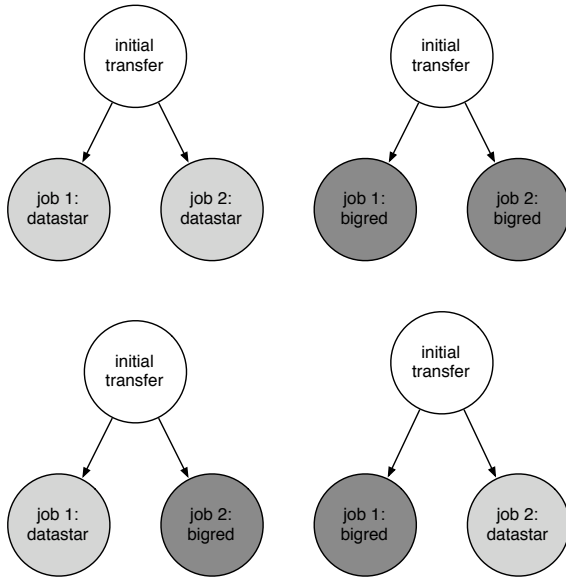
LED is a subset of YAML [38], a markup language intended to be easily read and written by humans. Anyone familiar with a scripting language like Python will find it natural. This design also makes it convenient to generate LED descriptions automatically by building the description using native Python data structures and serializing to LED. This is useful in building higher level, application-specific tools using the simulation framework.

```

experiment:
  name: "Simple parallel experiment"
  tasks:
    - name: "initial transfer"
      type: transfer
      size: 1000
      location: ranch
      destination: sdsc-HPSS
    - name: "independent jobs"
      type: "set"
      tasks:
        - name: "job 1"
          size: 256
          location: [datastar, bigred]
          type: "compute"
          duration: 12
        - name: "job 2"
          size: 256
          location: [datastar, bigred]
          type: "compute"
          duration: 12

```

**Figure 1: A LED experiment description with a transfer and independent tasks.**



**Figure 2: Workflow DAGs described by example in figure 1**

Each task in LED represents some operation on data, either a computation or a network transfer. Computation tasks must specify an estimate for their duration, a location, and a processor count. This is used to generate a queue wait time estimate during the simulation. Transfer tasks must specify the amount of data being transferred and the endpoints of the transfer. Figure 1 is an example experiment description in LED with two independent tasks, represented as a set. As shown in the task locations, lists of values are valid in LED, for all types of data values. Thus, a single LED description can represent several variations of an experiment. The example in figure 1 defines four separate experiments, one for each combination of the task locations. Each experiment has two tasks. Figure 2 shows a graphical representation of the four experiments defined by the example in figure 1.

### 3.1 Translation from Workflow Execution Languages

LED explicitly supports manual generation of workflow descriptions as a tool for prototyping potential changes, and we believe that it is important to allow fast prototyping in this manner for a prediction tool. However, if an experiment’s workflow has already been generated for a workflow execution system or language, it is best if that effort can be leveraged to generate an LED description. We are investigating ways to translate workflows in other languages into LED format.

Preliminary work has been done on a translator from the DAX XML format used by the Pegasus workflow execution system [7]. The translator uses the dependency list feature of LED to generate a DAG of computation tasks from the nodes in the DAX DAG, and generates data transfer tasks based on the data dependencies in the DAX file. We have translated examples in DAX from the archive presented by Bharathi et al. [3]. Evaluating a simulation model based on these translated examples will be the subject of future work.

At present the translation requires some manual interaction, to specify a list of systems that each computation could potentially run on. This information would be provided by the system for the workflow scheduler when executing the workflow, but we currently require user input as the translator is not integrated with the Pegasus system. In the future, this list could be saved so that translation could be nearly automatic across minor changes to the source workflow.

## 4. SIMULATION FRAMEWORK AND PARAMETERIZATION

To support a variety of analyses of experiments described using LED, we have built a simulation model framework consisting of a discrete event simulation engine written using Python and SimPy [30].

Keeping in mind that accurate parameterization is often the most challenging part of building a simulation tool, and hence additional parameters do not necessarily result in more accurate simulations [11], we have begun with a minimal set of parameters to describe workflows, shown in table 3(a). Workflow and system parameters are defined independently, to ease workflow definition and allow many workflow simulation users to share the effort of maintaining accurate and up-to-date system parameters.

Our discrete-event simulation engine reads the experiment description and statistical descriptions of system parameters and performs repeated simulations of the workflow on specified systems, to generate a probability distribution of experiment completion times. The framework is easily scriptable, so large simulation experiments and other tools can easily be built using it. We show one example -

the validation experiment in section 5, involved custom driver scripts to generate predictions for a large set of experiments across several systems.

Workflow Parameters	System Parameters
Job Processor Count	Network Bandwidth
Job Count	Queue Wait Time
Job Compute Time	
Task Dependency Graph	
Output Data Size	

(a) Simulation Model Parameters

Parameter	Values Used
Job Processor Count	256, 512, 1024
Output Data Size	1GB
Job Count	2, 4, 8
Inter-Job Dependency	serial, none
Archive Intermediate Results	yes, no
Job Compute Time	12 hours
System	BlueGene, Mercury, Lonestar

(b) Parameters Used to Generate Synthetic Workflows

**Figure 3: Description of Model Parameters used in this paper**

## 4.1 Network Bandwidth Parameterization

Because high granularity grid applications are very sensitive to network performance, monitoring and reporting network characteristics and using them in grid scheduling has been an important area of research.

We observe that for experiments in which network transfer time is a bottleneck, bandwidth, not latency, is the most common limiting factor. Although projects are increasingly using low-latency and high bandwidth links for synchronized collaborative analysis of large-scale datasets, as in [14], our experience shows that the most common bottleneck for large experiments is block transfer of very large files.

Although details of supercomputer center network connections are often published, these are generally peak numbers, and so any realistic parameterization requires some measurement. The Network Weather Service (NWS) is a project to measure the performance and availability of Grid computing systems [36, 35]. NWS monitors several aspects of computing systems, including uptime, CPU load, bandwidth and latency. The NWS system uses this time-series data as inputs to numerical models that generate predictions for the behavior of the measured systems in the near term. Network bandwidth is measured using raw TCP/IP transfers of 64 Kilobytes each. The goals of the NWS system are to allow very-near-term predictions to support immediate scheduling decisions, and as such require very frequent measurement. With high frequency, it is important not to impose a great deal of measurement overhead, and this motivated the relatively small transfer sizes, along with methods for forecasting larger transfers from small samples.

Because we do not share the goal of informing an immediate scheduling decision, we do not require a high rate of measurement. Therefore, we choose a more direct method of observing bandwidth of large transfers. Because users are likely to use a file transfer method designed for high performance, we directly observe the performance of large transfers using such a method. The method we observe is GridFTP, an extension to the FTP protocol with client and server implementations tuned for high performance with large transfers [1, 12]. In particular it supports parallel transfers, using multiple

servers and adaptive TCP/IP window size and related parameters to achieve high performance on long transfers.

For results presented in this paper, we parameterize the model using measurements of large block transfers of 100MB and 1GB. Effective bandwidth between systems was averaged over time with a simple moving average.

## 4.2 Queue Wait Time Parameterization

In this section we describe some approaches for parameterization based on analysis of batch queue logs. The evaluation of these approaches is based on data from an experiment which ran over the course of two weeks. Hundreds of jobs across five systems on the NSF TeraGrid were submitted and observations of queue delays were recorded for each system, broken down by job size and count. This experiment is part of the evaluation experiment described in section 5.

An initial parameterization method targeted at relatively short term predictions, such as predicting the wait time for a job to be submitted in the next day, might simply use the average of the wait times of jobs of the same size over a history window.

Early results using this approach resulted in unacceptable experiment completion time prediction error. In order to understand that error, we looked at the behavior of queue delay over time. Figure 4 shows data gathered in the experiment from the SDSC BlueGene system. The blue line shows queue delay in seconds for an individual job. The big peak in queue wait time from around 14 hours to 5 days is the result of a system downtime. Similar peaks can be seen in the queue logs of each of the observed systems, and in each case we tracked each peak to an individual event that caused the queue to be put on hold for some amount of time - either a planned high-priority run, or a system downtime.

It is clear that a parameterization should be chosen based on how the simulation will be used. If the goal is to predict the completion time of a short workflow for an immediate scheduling decision, a parameterization based on current data that tracks changes quickly is necessary. However, sometimes the goal of analysis is a more general prediction independent of a particular time, for long-range planning. In these cases, a parameterization that changes more slowly and so will smooth out the spikes, is more useful.

With these observations in mind, we developed two different parameterizations of queue wait time using log data. The first is a long-run summary of at least a week of probe data. It is a 25% trimmed mean, which results in a predictor that avoids tracking spikes.

The second is a moving average which tracks change-points using a method inspired by the data smoothing step described by Nurmi et al. in [4, 23]. The second, “tracking” parameterization uses an unbounded window that resets the window whenever it encounters three consecutive values outside the inner 80% of current values: either three in a row less than the 10% quantile or three in a row greater than the 90%. The window is reset to the three most recent values. This yields a predictor that tracks large spikes, which is useful for shorter workflows and predictions to be used immediately.

Figure 4 includes data from of the two predictors in the case of 512-processor jobs on BlueGene. Again, the series in blue is the actual wait times, the series in yellow is the long-run predictor, and the series in red is the second, change-point tracking, predictor.

We show two parameterizations based on unaided automatic analysis of observed batch queue history that match historical data relatively well. However, there are limitations to this history-based approach, because job queues in practice experience bursts of load that are potentially impossible to predict well from historical data. This is particularly important when simulating the execution of a

## Queue Wait Time Parameterization Comparison

### SDSC Blue Gene, 512-Processor Jobs

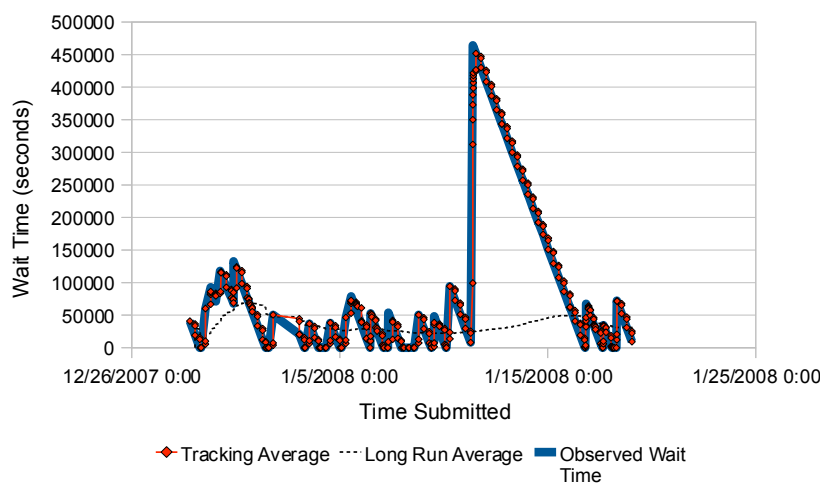


Figure 4: SDSC BlueGene Queue Delay and Two Parameterizations

workflow soon, but not immediately.

One possible approach to improving the accuracy of a queue wait time parameterization would be to leverage existing work on predicting job wait time using the contents of the existing queue, but such simulators require detailed knowledge of each site’s scheduling algorithm and policies [32, 9].

We have had experience with a few sites in the TeraGrid and at other sites and it is our experience that scheduling policy and practice can vary widely and rarely follows a straightforward algorithm. In attempting to develop a practical tool, we take the approach that observation is the most reasonable way to gather parameters for a large number of sites, which is a goal of our framework. Attempting to track the scheduling algorithms and policy tweaks with a detailed simulation for more than a few sites would be prohibitive.

However, we do believe that a limited amount of interaction with site queueing software and administrators would be worth pursuing. In particular, advance notice of planned downtime and notification of large reservations or large spikes in the number of submitted jobs could improve the parameterization scheme for our simulation model. Such spikes have two effects that could be tracked better, even with very short advance notice - first the spike itself affects the wait time of immediately executing jobs, but also the recovery from a spike is usually an orderly draining of the queue. Knowing that a given discontinuity in the observed wait time is an expected spike would allow the parameterization to include that draining period more accurately than it could if adjusting to it after the fact. In section 5.3 we discuss the improvements we expect in ranking accuracy due to this kind of added information.

## 5. EVALUATION

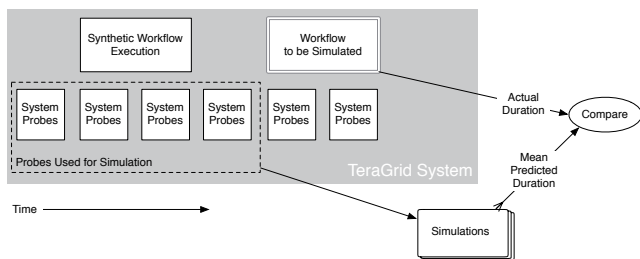
We have evaluated the accuracy of the simulator built using our toolkit by comparing predictions of smaller representative experiments run on three production TeraGrid systems: the SDSC BlueGene [29], NCSA Mercury [22], and TACC Lonestar system [21]. We also evaluated the corresponding long-term data archive facilities at each site, to highlight the point that transfer to archive storage is on the critical path for true completion of computational science experiments.

To evaluate the accuracy of our model, we compare predictions to actual completion times and present the mean absolute error. We also evaluate the model’s potential for guiding decisions by calculating the probability of the simulation model producing a correct ranking of the candidate system configurations in terms of overall completion time for a set of experiments started at the same time.

### 5.1 Simulation Model Error

Previous work highlighted some common patterns of large scale scientific experiments [16, 3]. Some experiments must be broken into many sequential jobs, sometimes involving intermediate transfers and analysis computation between tasks. Others consist of a large number of independent tasks generated by a parameter sweep. We generated synthetic workflows that are representative of these patterns to evaluate our model. We used the INCA [31] monitoring framework to execute and monitor the synthetic experiments. A script generated a set of experiments every 12 hours. Each set contains one experiment for each combination of the variables in Table 3(b). In order to gather enough data, we limited the size of experiments to two jobs lasting 12 hours each, corresponding to the smallest maximum job duration set by policy across all the sites. A description in our simulator’s input language was automatically written for each synthetic experiment. On each target system, a script ran every half hour to process the experiments, executing successive steps by submitting jobs or spawning file transfers if the previous step has completed. Just as with the queue wait time probes, an experiment’s queued jobs exit immediately, and the total experiment completion time is padded by 12 hours per sequential job, and by 12 hours total for any size set of independent jobs.

Figure 5 shows a timeline for the experiment. Each experiment workflow is executed twice a day on a system, while queue wait time probes on that system are submitted more frequently. The simulations were performed after runs were finished and recorded. Each experiment is simulated separately, using data from probes prior to the experiment’s execution to parameterize the simulator. The changepoint tracking moving average parameterization discussed in section 4.2 was used. Repeated simulations were run and error was calculated by the difference between the mean simulated du-



**Figure 5: Timeline of Accuracy Experiment**

ration and the recorded actual duration. A total of 215 synthetic experiments were successfully run over the course of two weeks on the three systems. One hundred simulations were then run for each experiment. Although the simulations were run after the fact, they were parameterized using probe data gathered for up to two weeks before the experiment began.

Figure 6 shows three tables of percentage mean absolute error per generated experiment type, with separate tables for each experiment type. For each workflow type we have calculated the mean absolute error and show the error percentage compared to the mean actual completion time. Average percentage error for the SDSC Blue Gene is 38.5%, for NCSA Mercury, 16.3%, and for TACC Lonestar, 42.8%. This level of accuracy represents a significant improvement over the guesswork in common practice, but admittedly does leave room for future improvement. We propose as an initial improvement the additional information suggested at the end of section 4.2. We discuss in section 5.3 the potential impact of this information on the ranking results of this experiment, and it is likely that the proposed changes would also improve the error of individual near-term predictions.

## 5.2 Evaluating Probability of a Correct System Ranking

Although absolute error is important, we believe that a more useful measure of the utility of a simulation model is how often using the model can lead to a good decision. In our context, this means how often the model would correctly rank the systems by experiment completion time when simulating the same workflow at the same time on all three systems.

We have designed an experiment to evaluate our model in this manner. In order to faithfully simulate this decision, and avoid confounding effects due to resource variations over time, we use only records of workflow executions that were submitted to all three machines at the same time and completed successfully. Of the 215 executions that were completed and used to evaluate prediction error, there were only seven sets of workflow executions that were the same type, were launched on all three systems concurrently and completed successfully. Each of these 21 workflow executions transferred 1GB of data and consisted of two jobs. All three of the workflow patterns were represented at least once, and all job sizes except 1024 processors were represented. Additionally, 22 other sets of workflow executions that were submitted to two of the systems (NCSA Mercury and SDSC BlueGene) completed successfully. For these 22 sets, the trial was posed as correctly ranking the two systems with recorded completion times.

Our experiment involved 100 trials simulating the sets of workflow executions. A trial involved simulating each of the three workflows in a set once, using probe data from before the set’s execution time. The trial was marked as a “success” if the simulations correctly identified the fastest system, as determined by the recorded

actual completion times. The proportion of successes to the total number of trials is taken as the probability that the model will choose correctly. A trial overall is considered a success if the probability that the model will choose correctly is high - in practice the model either correctly ranked the systems 100% or 99% of the time or 0%.

The top row of table 8, marked “Unaided” shows the results of this experiment broken down by workflow type. Overall, 17 of the 29 total trials were successful, for a success rate of about 59%. It is difficult to draw conclusions about ranking different types of workflows from this relatively small sample, because some workflow types were only represented once. However, some analysis of the queue wait times of jobs used for prediction history does yield insight into the sources of ranking error. The following section discusses each ranking failure in detail and gives an evaluation of the potential impact of the additional information described at the end of section 4.2.

## 5.3 Evaluating Potential Impact of Additional Information

Figure 7 shows time-series data for queue wait time on each of the three machines in our experiment. Data from both probe jobs and jobs that are part of experiment workflows are included, so this data represents jobs submitted more frequently than every 12 hours, but this does not represent the state of the entire job queue of each machine. The x-axis is the time the job was submitted, and the y-axis is the job’s queue wait time in hours. Solid lines represent the measured queue wait time, and dotted lines represent the value of the tracking parameterization at that time. Predictions of workflows that start at a given time will use the value of the parameterization at that time to predict the wait time of every job in the workflow. Points in time where at least one set of workflows were ranked incorrectly by our model are marked by a vertical dashed red line.

In order to evaluate the potential improvement that additional information about spikes as discussed at the end of section 4.2 would have, we investigate each set of ranking failures in detail. As before, we include all cases where at least two workflows completed and were ranked. In some cases, we find evidence that a simulation using a parameterization with advance notice of predictable spikes could have made a better ranking than our unaided parameterization.





There are four distinct timestamps where workflows are ranked incorrectly, and we examine each in detail in the following. The graph does not show how many or what type workflows failed at those times, but we include that information below where relevant.

Looking at the top graph in figure 7, we see that at 01/17 12:00, our model ranks three sets of 256-processor workflows incorrectly. The parameterization at this point has BlueGene with a higher wait time than Mercury, but there is significant variability, and Mercury has higher wait times for much of the next 6 hours. Because this is not due to a predictable spike, we must count this as an incorrect ranking.





Looking at the bottom graph in figure 7, at 01/17 14:12, our model ranks two sets of 512-processor workflows incorrectly. The parameterization here again has BlueGene with a higher wait time than Mercury, but the variability is significant for the next few hours. While a better-informed tracker might be able to correctly predict the queue draining from the preceding spikes, it is not clear whether it would result in a correct ranking, so we still count it as an incorrect ranking.

Both graphs in figure 7 have failed rankings at 01/18 02:12. Looking at the top graph, a set of 256-processor workflows with independent (concurrently submitted) jobs are ranked incorrectly at this point because the parameterization has Mercury with much lower wait times than BlueGene, but Mercury is about an hour away from





Percentage Mean Absolute Error for Workflows with Dependent Jobs

		256 × 2	512 × 2	256 × 8	512 × 4	1024 × 2
	SDSC BlueGene	28.19 (7)	50.29 (7)	26.55 (11)	30.72 (11)	36.65 (13)
	NCSA Mercury	9.75 (7)	21.07 (7)	8.25 (10)	23.49 (10)	-
	TACC Lonestar	62.19 (2)	-	30.16 (1)	53.54 (1)	-

Percentage Mean Absolute Error for Workflows with Dependent Jobs and Intermediate Transfers

		256 × 2	512 × 2	256 × 8	512 × 4	1024 × 2
	SDSC BlueGene	35.43 (7)	49.17 (7)	20.31 (9)	38.16 (9)	39.75 (13)
	NCSA Mercury	7.43 (7)	22.67 (7)	7.34 (9)	17.61 (10)	-
	TACC Lonestar	27.82 (1)	33.01 (2)	-	-	-

Percentage Mean Absolute Error for Workflows with Independent Jobs

		256 × 2	512 × 2	256 × 8	512 × 4	1024 × 2
	SDSC BlueGene	47.22 (10)	51.01 (9)	-	-	59.90 (7)
	NCSA Mercury	13.01 (7)	21.77 (7)	-	-	-
	TACC Lonestar	-	38.45 (1)	-	-	-

Column labels represent Processor Count × Jobs in Workflow  
 Values in parentheses are number of workflows predicted for the presented data.

**Figure 6: Mean Absolute Error**

a downtime spike. Assuming that this spike was foreseeable within an hour, it is possible that a parameterization with that advance knowledge would have supported a correct ranking in this case.

Looking at the bottom graph for the same time, three sets of 512-processor workflows are also ranked incorrectly. This is due to a spike in wait time for jobs on BlueGene to about 10 hours, occurring about 10 hours earlier, while for several hours after the incorrect ranking, BlueGene has a wait time close to zero. If this spike was noted and a parameterization that models queue draining correctly was used, the ranking could have been correct for all three of these cases.

Finally, both graphs also have failed rankings at 01/18 08:12. At this point, both a 256-processor and a 512-processor set of workflows with independent jobs are ranked incorrectly. The wait time for BlueGene will spike about an hour after this time, but the unaided parameterization has BlueGene near zero while Mercury is higher, after recovering from two separate spikes at 256 and 512 processors. A parameterization with advance notice of the upcoming spike in BlueGene’s wait time should be able to correctly rank these two cases.

Starting from the initial ranking success rate of 17/29 or 59%, if we simply discard the six failures which we claim would be improved by a parameterization with advance knowledge of spikes, we get a success rate of 17/23, or 74%. If we assume that an improved simulation would get all six of those failures right we get a success rate of 23/29 = 79%. The bottom two rows of the tables in figure 8, marked “Advance Notice A” and “Advance Notice B” show breakdowns of the results if the affected ranking failures are removed or marked as successes, respectively. We are currently working to verify these estimates in simulation using an improved parameterization.

We conclude from this that it appears there is some promise in using simulation to rank systems by likely workflow completion time,

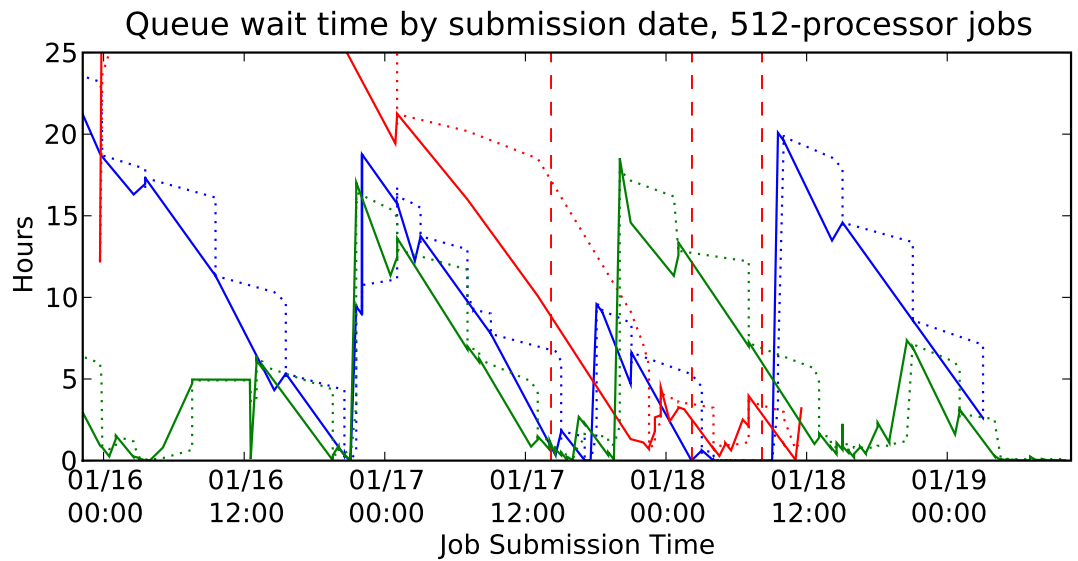
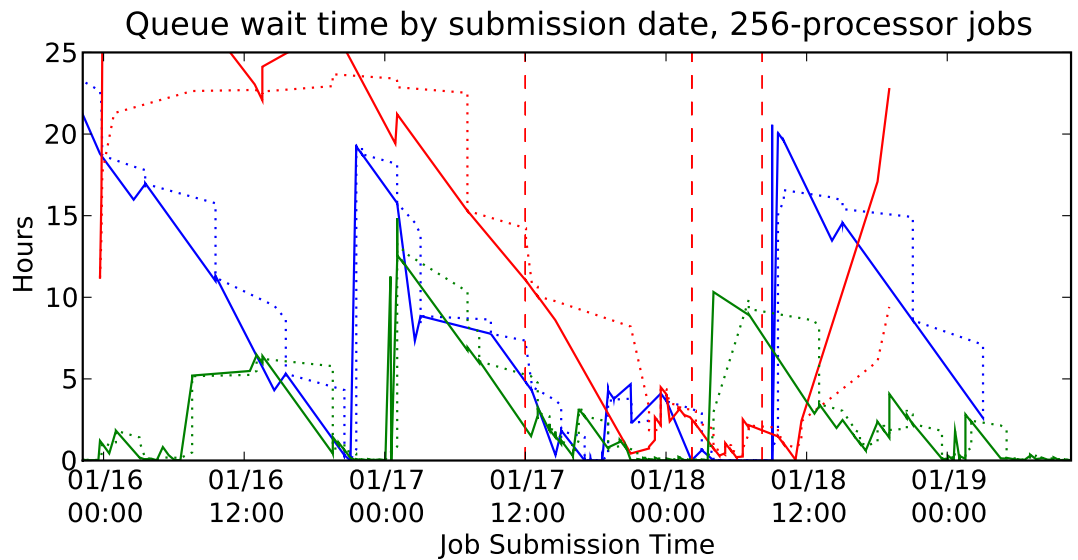
but that integration with system schedulers and advance knowledge of reservations, large queued jobs, and system maintenance plans could significantly improve the accuracy of such a system.

## 6. RELATED WORK

Scientific workflow management is an active area of related work. In particular, many workflow languages and scheduling and execution systems have been developed for use on computational grid resources. In the context of these systems, workflows are composed of large numbers of loosely coupled large and small tasks, from querying remote databases to serial and parallel computational tasks.

There are many systems that could be described as workflow execution systems, some more focused on science than others. A starting point for publications about these systems is available in [34, 6]. Most systems either use a language or GUI to define the workflow directly, or define the workflow as the tasks generated while executing a script. A common goal of the systems intended for computational science is to insulate the scientist from details of data location and resource scheduling, whereas our goal is to allow interested scientists to explore the relationship between those variables and time to solution. Our simulator’s workflow description is not a workflow execution language. We expect that a scientist who has a workflow expressed in such a language would want to translate from that language in order to use our simulator alongside the main workflow management system.

SimGrid is a large and feature-rich framework for writing simulations of grid applications to evaluate distributed algorithms for use on clusters, grids, and in Peer-to-Peer networks [5]. It is intended to support repeatable evaluation of applications, scheduling algorithms and heuristics. SimBatch provides batch queue delay prediction to simulations written using SimGrid [10]. The SimGrid project includes GRAS, an API to write grid applications which can then



**Figure 7: A Subset of the Experiment with Queue Wait Times as Solid Lines, Tracking Average Values as Dotted Lines, and Ranking Failures Marked by Vertical Red Dashed Lines**

Number of Correct Ranking Trials for Workflow Sets with All Three Systems Represented

	$256 \times 2$ dependent jobs	$256 \times 2$ dependent jobs with transfers	$256 \times 2$ independent jobs	$512 \times 2$ independent jobs	$512 \times 2$ dependent jobs with transfers
Unaided	1/2	1/1	0/1	1/1	1/2
Advance Notice A	1/2	1/1	0/1	1/1	1/1
Advance Notice B	1/2	1/1	0/1	1/1	2/2

Number of Correct Ranking Trials for Workflow Sets with Only Two Systems Represented

	$256 \times 2$ dependent jobs	$256 \times 2$ dependent jobs with transfers	$256 \times 2$ independent jobs	$512 \times 2$ independent jobs	$512 \times 2$ dependent jobs with transfers	$512 \times 2$ dependent jobs
Unaided	1/2	2/3	4/6	3/5	1/2	2/4
Advance Notice A	1/2	2/3	4/4	3/3	1/2	2/3
Advance Notice B	1/2	2/3	6/6	5/5	1/2	3/4

**Figure 8: Fraction of Successful Ranking Trials, Including Estimates of Successful Rankings due to Advance Notice of Spikes**

be either simulated or executed directly with no changes to their code [27]. SimGrid is a collection of libraries with C and Java interfaces, so simulations are themselves significant programs, and thus may not be appropriate for scientists looking for an easy way to evaluate resource choices.

Statistical models of job run times have been used to predict the queue wait time of individual jobs based on the characteristics of other jobs in the queue [32, 9]. More recently, Brevik et al. discuss a technique for predicting bounds on queuing delay for single jobs submitted to shared systems, using statistical analysis of past jobs [4]. Nurmi et al. expand on the historical analysis approach in another paper, using predictions to improve automatic scheduling of entire workflows on a grid [24]. In contrast, our approach focuses on planning, which allows users to explore both present and hypothetical options.

## 7. CONCLUSIONS AND FUTURE WORK

We have presented a simulation toolkit and one example model of scientific experiments using a minimal description of workflow and system information. We have evaluated the model against executions of synthetic workflows on real systems and show that with advance notice of predictable events that cause large spikes in queue wait time, the model can choose the system which will complete a workflow soonest. Although our results show that this technique has promise, the absolute prediction error leaves room for improvement, and we will investigate techniques to increase accuracy. We suggest that integration with schedulers and advance knowledge of system maintenance schedules may be important for accurate and reliable simulation modeling of near-term workflow executions, and show the impact of such integration in our simulations, increasing the success rate of ranking systems from 59% to as high as 79%. We believe that simulation is already a useful tool for long-term planning of workflow executions, because its ability to quantify the impact of workflow and system changes is an improvement on current practice, even given the prediction error we experience in our testing.

Because our work currently assumes that users can predict their job run time, we will also explore modeling compute task duration in more detail, using existing work on performance models to generate a scaling factor for compute time on different machines. The impact of increasingly frequent system failure is difficult for an individual user to deal with. System software and programming models need

to be reworked to support graceful adaption to failure. These will likely require code changes, and so estimates of the time lost to failure for an experiment can be useful to help decide when such changes are necessary. We will extend our prediction technique to include models of system failure to support planning such changes.

## 8. ACKNOWLEDGMENTS

The authors would like to thank Nicole Wolter, Robert Harkness and Mahidhar Tatineni for their help and comments on system and experiment characteristics. We also thank Jeanne Ferrante and Gregory Johnson for support and comments on earlier versions of this paper. Finally, we thank the anonymous reviewers for their thorough and helpful comments.

This work was supported in part by a grant from the National Science Foundation entitled “The Cyberinfrastructure Evaluation Center,” and in part by NSF grant #CCF-0446604. This work was sponsored in part by the Department of Energy, Office of Science through award entitled “HPCS: Application Development Time Evaluation.”

## 9. REFERENCES

- [1] W. Allcock, J. Bresnahan, R. Kettimuthu, M. Link, C. Dumitrescu, I. Raicu, and I. Foster. The Globus striped GridFTP framework and server. In *Supercomputing*. ACM Press, 2005.
- [2] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, 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.
- [3] S. Bharathi, A. Chervenak, E. Deelman, G. Mehta, M.-H. Su, and K. Vahi. Characterization of scientific workflows. In *In proceedings of 3rd Workshop on Workflows in Support of Large-Scale Science (WORKS08)*. ACM, 2008.
- [4] J. Brevik, D. Nurmi, and R. Wolski. Predicting bounds on queuing delay for batch-scheduled parallel machines. In *PPoPP*, pages 110–118, New York, NY, USA, 2006. ACM Press.
- [5] H. Casanova, A. Legrand, and M. Quinson. SimGrid: a generic framework for large-scale distributed experiments. In

- 10th IEEE International Conference on Computer Modelling and Simulation, 2008.
- [6] B. Clifford. Grid workflow. Presentation at International Summer School on Grid Computing, July 2007. [http://library.iceage-eu.org/resolve/resolver.jsp?svc\\_dat=details&rft\\_dat=lib:8709](http://library.iceage-eu.org/resolve/resolver.jsp?svc_dat=details&rft_dat=lib:8709).
- [7] E. Deelman. Managing workflows with the Pegasus workflow management system. presentation, 2007.
- [8] E. Deelman, G. Singh, M.-H. Su, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, G. B. Berriman, J. Good, A. Laity, J. C. Jacob, and D. S. Katz. Pegasus: a framework for mapping complex scientific workflows onto distributed systems. *Scientific Programming Journal*, 13(3):219–237, 2005.
- [9] A. B. Downey. Predicting queue times on space-sharing parallel computers. In *11th Intl. Parallel Processing Symp.*, pages 209–218, 1997.
- [10] J.-S. Gay and Y. Caniou. Simbatch: an api for simulating and predicting the performance of parallel resources and batch systems. Technical Report 6040, INRIA, 2006.
- [11] J. Gibson, R. Kunz, D. Ofelt, and M. Heinrich. FLASH vs. (simulated) FLASH: Closing the simulation loop. In *Architectural Support for Programming Languages and Operating Systems*, pages 49–58, 2000.
- [12] Globus Alliance. GridFTP. website. [http://www.globus.org/grid\\_software/data/gridftp.php](http://www.globus.org/grid_software/data/gridftp.php).
- [13] R. Harkness. personal communication. Jan. 2007.
- [14] R. Kooima. *Planetary-scale Terrain Composition*. PhD thesis, University of Illinois at Chicago, 2008.
- [15] B. Ludäscher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger, M. Jones, E. A. Lee, J. Tao, and Y. Zhao. Scientific workflow management and the Kepler system. *Concurrency and Computation: Practice and Experience*, 18(10):1039–1065, 2006.
- [16] M. O. McCracken, N. Wolter, and A. Snaveley. Beyond performance tools: Measuring and modeling productivity in HPC. In *Proceedings of ICSE Workshop on Software Engineering in High-Performance Computing*, May 2007.
- [17] H. Meuer, E. Strohmaier, J. Dongarra, and H. Simon. Top 500 supercomputer sites. <http://top500.org/>, Sept. 2008.
- [18] J. Michalakes, J. P. Hacker, M. O. McCracken, R. Loft, and A. Snaveley. WRF nature run. In *Proceedings of the 2007 ACM/IEEE conference on Supercomputing*, Nov. 2007.
- [19] National Science Foundation. TeraGrid project. website. <http://teragrid.org>.
- [20] N. (NERSC). NERSC Seaborg IBM SP system. <http://www.nersc.gov/nusers/systems/SP/>.
- [21] NSF TeraGrid. TACC Lonestar TeraGrid info site. <http://teragrid.org/userinfo/hardware/resources.php?type=compute&select=single&id=8>.
- [22] NSF TeraGrid. NCSA Teragrid IA-64 cluster info site. <http://www.ncsa.uiuc.edu/UserInfo/Resources/Hardware/TGIA64LinuxCluster/>, Jan. 2008.
- [23] D. Nurmi, J. Brevik, and R. Wolski. QBETS: Queue bounds estimation from time series. In *Workshop on Job Scheduling Strategies for Parallel Processing*, 2007.
- [24] D. Nurmi, A. Mandal, J. Brevik, C. Koelbel, R. Wolski, and K. Kennedy. Evaluation of a workflow scheduler using integrated performance modeling and batch queue wait time prediction. In *Proceedings of Supercomputing 2006*. ACM/IEEE, Nov. 2006.
- [25] B. W. O’Shea, G. Bryan, J. Bordner, M. L. Norman, T. Abel, R. Harkness, and A. Kritsuk. Introducing Enzo, an AMR cosmology application. In T. Plewa, T. Linde, and V. G. Weirs, editors, *Adaptive Mesh Refinement - Theory and Applications*, Springer Lecture Notes in Computational Science and Engineering, 2004.
- [26] Pittsburgh Supercomputing Center. TeraGrid SpeedPage. website. <http://speedpage.psc.teragrid.org>.
- [27] M. Quinson. Gras: A research & development framework for grid and p2p infrastructure. In *18th IASTED International Conference on Parallel and Distributed Computing and Systems*, 2006.
- [28] N. S. V. Rao, W. Yu, W. R. Wing, S. W. Poole, and J. S. Vetter. Wide-area performance profiling of 10GigE and InfiniBand technologies. In *SC ’08: Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, pages 1–12, Piscataway, NJ, USA, 2008. IEEE Press.
- [29] SDSC User Services. SDSC BlueGene user guide. <http://www.sdsc.edu/us/resources/bluegene/>.
- [30] SimPy Developer Team. SimPy simulation framework homepage. <http://simpy.sourceforge.net/>, 2007.
- [31] S. Smallen, K. Ericson, J. Hayes, and C. Olschanowsky. User-level grid monitoring with inca 2. In *GMW ’07: Proceedings of the 2007 workshop on Grid monitoring*, pages 29–38, New York, NY, USA, 2007. ACM.
- [32] W. Smith, V. Taylor, and I. Foster. Using run-time predictions to estimate queue wait times and improve scheduler performance. In D. G. Feitelson and L. Rudolph, editors, *Workshop on Job Scheduling Strategies for Parallel Processing*, pages 202–219. Springer Verlag, 1999.
- [33] A. Snaveley and J. Kepner. Is 99% utilization of a supercomputer a good thing? In *SC ’06: Proceedings of the 2006 ACM/IEEE conference on Supercomputing*, page 37, New York, NY, USA, 2006. ACM Press.
- [34] I. J. Taylor, E. Deelman, D. Gannon, and M. Shields, editors. *Workflows for e-science: scientific workflows for grids*. Springer, 2007.
- [35] R. Wolski, L. Miller, G. Obertelli, and M. Swamy. Performance information services for computational grids. In J. Nabrzyski, J. Schopf, and J. Weglarz, editors, *Resource Management for Grid Computing*, chapter 1. Kluwer, 2003.
- [36] R. Wolski, N. Spring, and J. Hayes. The network weather service: A distributed resource performance forecasting service for metacomputing. *Journal of Future Computing Systems*, October 1999.
- [37] N. Wolter, M. O. McCracken, A. Snaveley, L. Hochstein, T. Nakamura, and V. Basili. What’s working in HPC: Investigating HPC user behavior and productivity. *CTWatch Quarterly*, 2(4A), November 2006.
- [38] YAML home page. <http://yaml.org>.
- [39] Y. Zhao, M. Hategan, B. Clifford, I. Foster, G. vonLaszewski, I. Raicu, T. Stef-Praun, and M. Wilde. Swift: Fast, reliable, loosely coupled parallel computation. In *IEEE International Workshop on Scientific Workflows*, 2007.