

Evaluation of a Workflow Scheduler Using Integrated Performance Modelling and Batch Queue Wait Time Prediction

Daniel Nurmi¹, Anirban Mandal², John Brevik¹, Chuck Koelbel², Rich Wolski¹, Ken Kennedy²

¹Computer Science Department
University of California, Santa Barbara
Santa Barbara, California

²Computer Science Department
Rice University
Houston, Texas

Abstract

Large-scale distributed systems offer computational power at unprecedented levels. In the past, HPC users typically had access to relatively few individual supercomputers and, in general, would assign a one-to-one mapping of applications to machines. Modern HPC users have simultaneous access to a large number of individual machines and are beginning to make use of all of them for single-application execution cycles. One method that application developers have devised in order to take advantage of such systems is to organize an entire application execution cycle as a workflow. The scheduling of such workflows has been the topic of a great deal of research in the past few years and, although very sophisticated algorithms have been devised, a very specific aspect of these distributed systems, namely that most supercomputing resources employ batch queue scheduling software, has heretofore been omitted from consideration, presumably because it is difficult to model accurately. In this work, we augment an existing workflow scheduler through the introduction of methods which make accurate predictions of both the performance of the application on specific hardware, and the amount of time individual workflow tasks will spend waiting in batch queues. Our results show that although a workflow scheduler alone may choose correct task placement based on data locality or network connectivity, this benefit is often compromised by the fact that most jobs submitted to current systems must wait in overcommitted batch queues for a significant portion of time. However, incorporating the enhancements we de-

scribe improves workflow execution time in settings where batch queues impose significant delays on constituent workflow tasks.

1 Introduction

As grid programming tools have matured, the possibility of using multiple shared clusters and/or supercomputers has emerged as a viable platform for executing large parallel workflows [Deelman et al. 2004]. Previous work has shown that, if the machines are dedicated (but the networks interconnecting them are not), it is possible to use an application-specific performance model (parameterized dynamically) to choose the execution setting that optimizes overall execution time [Mandal et al. 2005]. When the machines are not dedicated, however, they are each typically managed by a separate batch-queue scheduler that implements a local space-sharing policy. To execute, workflow components must be submitted to these queues to wait until a suitable machine partition becomes available. If the machine is fully utilized, the time in queue can be highly variable and potentially long; longer in many cases than the execution time itself.

The problem of efficiently scheduling workflow tasks to batch queue controlled resources is rapidly becoming crucial to the community's effort to support a wide variety of HPC applications using Grid research technologies. As part of the VGrADS (Virtual Grid Application Development Software) research project, which focuses on the development of programming tools to lessen the complexity of grid application development, maintenance, and support, we have focused on showing that the problem of resource selection amongst a wide variety of distributed resources (including batch queue controlled supercomputers) can be efficiently handled through use of prediction techniques and

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. SC2006 November 2006, Tampa, Florida, USA 0-7695-2700-0/06 \$20.00 ©2006 IEEE

look-ahead scheduling. Further, a key focus of VGrADS is the simplification of grid programming for the application developer. Concordantly, we show that it is possible to efficiently schedule application tasks automatically across a diverse resource pool having widely and dynamically varying performance characteristics.

Specifically, in this paper, we investigate how a new technique for predicting batch queue delay [Brevik et al. 2006; Downey 1997a; Downey 1997b; Smith et al. 1999] can be used to improve the execution times of large-scale workflows in shared grid environments. Our method uses an application scheduler and performance model to determine a mapping of workflow components to cluster processors based on predicted start-up delay (resulting from queuing time) and predicted execution time. The scheduler parameterizes its model with delay predictions that are generated dynamically at the time the workflow is initiated. Based on the mapping it determines to be most advantageous, submits jobs that it generates on-the-fly to the local batch queues controlling the processors it wishes to use. That is, given a set of clusters controlled by batch queues, our system maps workflow components to job submission requests in a way that attempts to optimize workflow turn-around time.

We investigate the efficacy of this approach using the EMAN application [Ludtke et al. 1999], the VGrADS EMAN scheduler [Mandal et al. 2005], and new Network Weather Service (NWS) [Wolski et al. 1999a; Wolski et al. 1999b] queue delay prediction functionality [Brevik et al. 2006; Downey 1997a; Downey 1997b; Smith et al. 1999] targeting TeraGrid* as a grid of clusters. Our goal in this investigation is to explore how the VGrADS performance modeler and application scheduler can improve overall application turn-around time for the user when combined with the NWS methodology for predicting batch-queue delay.

Our results indicate that queue prediction substantially reduces turn-around time for the EMAN workflow in this setting. By integrating this functionality into of the VGrADS programming environment and execution systems, we expect to provide similar benefits for workflow problems that have similar characteristics.

2 Related Work

Past research relevant to this study has been done in three distinct fields. The first is that of characterizing HPC applications as workflows to be executed on large-scale systems. From this work, a significant amount of of the literature involves efficient scheduling of workflows onto systems. Scheduling application components onto multiprocessors is a hard problem. In most cases the problem is NP-complete, since the minimum multiprocessor scheduling problem is NP-complete [Garey and Johnson 1979].

*NSF TeraGrid Project <http://www.teragrid.org>

Therefore, most of the literature deals with finding good heuristic solutions. There is a body of work on multiprocessor heuristic scheduling of independent application components. Braun et al. [2001] give an overview of different heuristics. However, these heuristics can't be directly applied for scheduling workflows because of task dependencies.

Kwok *et al.* [1999] give a survey on different heuristic scheduling techniques for scheduling application DAGs (or workflows) onto homogeneous platforms. These heuristics also can't be applied directly to highly heterogeneous HPC systems. Topcuoglu *et al.* [2002], Sih *et al.* [1993] and Oh *et al.* [1996], *etc.* consider DAG scheduling for heterogeneous platforms. Most of the heuristics are generalizations of the list-scheduling based heuristics for homogeneous platforms, which have several drawbacks in heterogeneous HPC systems. First, they do not consider the global effect of the current scheduling decision. Second, they do not group tasks for scheduling and third, heterogeneity makes the average values they use for edge and node weights questionable. Mandal et al. [2005] and Blythe et al. [2005] describe strategies for scheduling workflows onto heterogeneous, distributed Grid resources using performance models. But, they assume instant resource availability, which is in general not true for modern HPC systems.

The second field of import is that of creating accurate performance models that can be used in a predictive way. Accurate component performance models are used as surrogates of actual execution times during the workflow scheduling process. Accurately predicting the performance of a parallel and distributed application is a hard problem and has been studied extensively in the literature. Several modeling techniques like analytical, simulation-based, hybrid modeling etc. have been used for accurately predicting application performance on a wide range of architectures. Sundaram-Stukel et al. [1999] and Kerbyson et al. [2001] describe detailed analytical performance modeling techniques for special case applications. Marin et al. [2004] describe a semi-automatic strategy of modeling static and dynamic characteristics of applications in an architecture-neutral fashion using a combination of static and dynamic analysis of application binaries. The Prophecy framework [Taylor et al. 2003] uses historical performance data, system features and application details (all stored in a database) to drive analytical modeling of application components on different platforms. Pillana et al. [2005] use a hybrid modeling technique using discrete event simulation and mathematical modeling. The workflow scheduler can use any available accurate performance predictor to drive the scheduling process. We use modeling techniques by Marin et al. [2004] for the purpose of this work.

Finally, a great deal of work exists attempting to char-

acterize job workloads on various HPC systems in order to predict the amount of time individual jobs spend waiting in batch queues. Many previous studies attempt to show that job wait times can be predicted accurately under the assumptions that we know the length of time the job will execute and that we have perfect knowledge of the scheduling algorithm. If these conditions are met, it has been shown by Smith, Taylor and Foster [1999] that the mean job wait time can be predicted but there is substantial between the predicted and the actual observed wait times. In their work, they employ empirical workload traces to derive a model for job execution time. From the model, and assuming perfect knowledge of the scheduling algorithm, they can calculate mean job wait times via simulation in faster-than-real time. Another approach from Downey [1997a; 1997b] uses a similar set of assumptions to derive queue wait times, but he instead uses a log-uniform distribution to model remaining job execution times. From this model, he shows that it is possible to predict when a certain-sized “cluster” of resources becomes available and thus can predict how long a job at the head of the queue will wait. Both of these efforts require that the scheduling algorithm is both exactly known and not impacted by policy change during the lifetime of the experiment. In practice, it is rarely the case that we have access to the precise scheduling policy being employed by a given system, and the policies are clearly not static over a long period of time.

In a paper by Brevik, Nurmi and Wolski [2006], the authors explore an alternative method for job wait-time prediction that uses only the observed job wait times in order to make bound predictions on individual wait times. In that work, they suggest that often a user may be more interested in upper- and lower-bound predictions, together with a measure of *confidence* or certainty, for job wait times, since a point-valued prediction for, say, the mean (expected) wait time is not particularly meaningful for such highly right-skewed data. The work produced a software infrastructure that has been shown to predict quantile bounds for individual job wait times on 7 separate HPC systems over a 9-year period.

In the present work, we show that combining state-of-the-art workflow scheduling, performance modeling, and batch queue prediction technologies result in a workflow scheduling system that can attain much faster workflow turnaround time than when using any of these components alone. In the next section, we will briefly outline the performance-modeling technique, the batch-queue prediction technique, and finally the merger of these technologies into a high-quality workflow scheduler.

3 Performance Modeling

This section describes our performance modeling methodologies. We first describe how to estimate the execution time of a workflow component on a resource. We analyze an application component’s behavior by modeling its characteristics in isolation of any architectural details. We then estimate the component’s execution cost on a target platform described by its available hardware resources (e.g. number and type of execution units, cache size and memory access latency etc.). To characterize a component’s single-node performance, we consider both the number of floating point operations executed as well as its memory access pattern.

To measure the amount of computation performed by an application component for a particular program input, we use hardware performance counters to collect floating-point operation counts from several executions of the program with different, small-size input problems. We then apply least square curve-fitting on the collected data to predict for an actual input data-set.

To understand a component’s memory-access pattern, we collect histograms of memory reuse distance (MRD) – the number of unique memory blocks accessed between a pair of references to the same block – observed by each “load and store” instruction [Marin and Mellor-Crummey 2004]. Characterizing memory access behavior for programs in this way has two major advantages. First, data reuse distance is independent of cache configuration or architecture details. Second, reuse distance is a measure of data reuse, which is the main determinant in cache performance. We collect reuse distance information for each reference in the program for several small-size input problems. We use the memory reuse distance data to model the behavior of each memory instruction and to predict the fraction of hits and misses for a given problem size and cache configuration. Our modeling strategy dynamically finds groups of accesses that have similar growth functions for the reuse distance and models each such group using two polynomials; one of which models how the number of accesses in that group changes with problem size and the other how the average reuse distance of those accesses changes with problem size. To determine the cache miss count for a different problem size and cache configuration, we evaluate the MRD models for each reference at the specified problem size and count the number of references with reuse distance greater than the target cache size.

We use the following simplified model to predict the execution time for the workflow components.

$$EstExecTime(psize) = \frac{A + B + C + D}{CpuClock(arch)}$$

where

$$A = k_0 \times \frac{\text{totalFp}(psize)}{\text{FpPipelineNum}(arch)} \times \text{FpRptRt}(arch)$$

$$B = k_1 \times \text{L1MissCnt}(psize) \times \text{L1MissPnlty}(arch)$$

$$C = k_2 \times \text{L2MissCnt}(psize) \times \text{L2MissPnlty}(arch)$$

$$D = k_3 \times \text{L3MissCnt}(psize) \times \text{L3MissPnlty}(arch)$$

In the equations, k_0, k_1, k_2 and k_3 are constants, $psize$ is the problem size and $arch$ is the target architecture. $\text{FpRptRt}(arch)$ is the repeat rate of the floating point pipeline. It is the number of cycles that occur between the issue of one instruction and the issue of the next instruction to the same execution unit. MissPnlty , the penalty for a miss in an arbitrary level of the memory hierarchy, is the difference between the access time to the next memory level and the access time to the current memory level.

$$L(j)\text{MissPnlty}(arch) = P - Q$$

$$P = L(j+1)\text{Latency}(arch)$$

$$Q = L(j)\text{Latency}(arch)$$

We also obtain the communication performance model for a particular workflow component C on a particular resource R . First, we find the set of resources to which the predecessors of C have been mapped. We then add the cost of data movement from each of those resources to R to obtain the estimated overall communication cost. Costs are estimated as a function of measured latency/bandwidth values between resource pairs and the known volume of communication annotated on the workflow edges.

We obtain the final performance model of a workflow component by adding the estimated execution time to the estimated communication time.

4 Prediction of Wait Times in Batch Queues

Typically, HPC resources are managed using *space sharing*, a high-level scheduling strategy according to which each application is allocated a dedicated set of resources for the duration of its execution. Most modern space-sharing systems use standard resource-management and scheduling software, such as LoadLeveler, EASY [Lifka et al. 1995], PBS, NQS/NQE, Maui and GridEngine[†], to manage the mapping of applications to resources. Since there are typically more jobs needing access to dedicated resources than

[†]Most batch queue software documentation can be found online at the following locations

PBS - <http://www.altair.com/software/pbspro.htm>

NQS/NQE - http://docs.cray.com/books/2148_3.3/html/2148_3.3

Maui - <http://www.clusterresources.com/products/maui>

GridEngine <http://gridengine.sunsource.net>

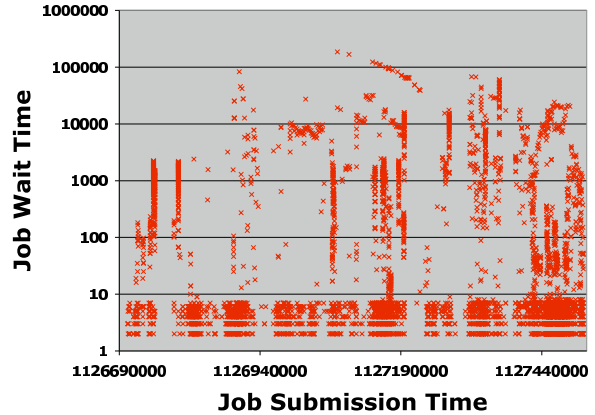


Figure 1. Graph depicting job wait times from one machine, in one queue, over a 3 week period.

there are resources available at any point in time, these software systems typically implement some form of *batch queue* for application jobs that are ready to execute as soon as resource become available. One problem that arises in this environment is that the user has little idea how long his or her job will wait in the batch queue. We have observed that a job on a heavily utilized machine will often wait in the queue for more time than it requires to execute. Since scientists are beginning to have access to multiple batch-queue-controlled HPC machines, we contend that understanding the relative batch-queue delay between machines is becoming an important issue.

Although some work has focused on point-value predictions for job wait times in a batch queue [Downey 1997a; Downey 1997b; Smith et al. 1999], we believe that often a more relevant question is that of determining bounds on the amount of time a single job will wait in a queue. The determination of an exact value is practically impossible, and even the average (expected) wait time is of limited practical utility, given the complex and highly skewed nature of wait-time data. In our own investigations, we have looked data from 7 different HPC sites over a 9-year period. It clear that job wait times make for a dataset that is difficult to model. In Figure 1, we show a typical snapshot of a queue from a highly utilized batch-queue-controlled machine. From this graph, we can see that there are distinct modes, drastic changes in regimes (*change-points*), periods of relative inactivity followed by bursts of data, etc.

We have developed a method, the Binomial Method Batch-Queue Predictor (BMBP), which we have shown to make accurate and correct predictions (in a very specific sense to be discussed below) for bounds on job wait times. BMBP accomplishes this by predicting quantiles directly from historical job wait time data. As an example, BMBP

is able determine longest time that a particular job is “probably” going to wait in a particular queue, in the sense that there is, say, a 75% chance that the job will begin executing in less than that much time. BMBP treats interprets the problem as that of finding an upper bound on the .75 quantile of the random variable of possible wait times for the particular job, using a historical trace of wait times that jobs have experienced in the queue. Note that a separate issue is that of *confidence*, namely the level of certainty that the upper bound determined really is at least as large as the .75 quantile; BMBP typically uses a confidence level of 95%. However, both of the numerical values (the quantile of interest and the confidence level) can be adjusted by the user; for example, we can equally well ask BMBP for the 0.5 quantile, or median, with 90% confidence. In other words, BMBP allows the user to ask the question, “What is the longest that my job is *likely* to wait in the queue?” where “likely” can be interpreted by the user via a quantile (and, if desired, a confidence level).

4.1 The Binomial Method Batch Predictor

Quantile bound predictions made by the BMBP is based on the following simple observation: If X is a random variable, and X_q is the q quantile of the distribution of X , then a single observation x from X will be greater than X_q with probability $(1 - q)$. Thus (under suitable assumptions about independence and identical distribution) as a sequence of independent Bernoulli trials with probability of success equal to q , where an observation is regarded as a “success” if it is less than X_q . If there are n observations, the probability of exactly k “successes” is described by a Binomial distribution with parameters q and n . Therefore, the probability that k or fewer observations are greater than X_q is equal to

$$\sum_{j=0}^k \binom{n}{j} \cdot q^{n-j} \cdot (1 - q)^j$$

Using this basic method, we can find the smallest value of k for which Equation 4.1 is larger than some specified confidence level, and the k^{th} value in a sorted set of observations (of sufficient size) will be greater than or equal to the X_q quantile of the distribution from which the observations were made with the specified level of confidence.

In practice, the BMBP is implemented as a trace-based simulation which takes as input a historical job trace, a user specified quantile and a confidence level. After a short training period, the simulation walks through the historical trace data until it arrives at the present. During the simulation, we employ a simple changepoint detection technique that is used to effectively trim the number of historical data points the predictor is taking into account, allowing the method to

only use history which is relevant to the current wait time conditions. In addition, we employ a technique to “group” jobs based on the number of nodes they request, which has the effect of tightening the prediction made for a specific job by using only a history containing other jobs of similar size. At the point where the simulator has processed all jobs up until the present, we can ask the question, for a job of specific node requirements, for the latest quantile prediction with the specified level of confidence. The simulator has been shown to correctly capture the population quantile of interest for almost all of the job traces for which we have access. In our previous experiment [Brevik et al. 2006], the quantile predictions were successful for 51 out of 55 traces.

In this work, we integrate our BMBP batch queue bound prediction technique into a performance model enhanced workflow scheduler, which is the topic of the next section.

5 Scheduler Design

The core idea of this work is to equip the VGrADS workflow scheduler [Mandal et al. 2005] with detailed information about the expected performance of an application on specific resource architectures and to use predictions of when resources become available to decrease the overall makespan of a workflow when executed on real systems. In previous works, the authors have shown that performance modes and batch queue wait time prediction techniques can correctly predict their respective quantities. In this section, we briefly describe the way our novel workflow scheduler makes use of this predictive information to produce a “plan” that should reduce the observed makespan for a single workflow.

The original workflow scheduling algorithm runs three heuristics (min-min, max-min and suffrage [Tracy D. Braun et al 2001]) and works as follows. For each heuristic, until all components in the workflow are mapped, the current set of available components is identified. The rank matrix is then calculated for the set of available components. The (i, j) entry in the rank matrix encodes the expected computation and communication performance of application component i on resource j (obtained using the performance-modeling techniques described in section 3). Then, the estimated completion time of a component i on a resource j , $(ECT(i, j))$, is obtained by adding the rank value to the maximum of the following two entities – (1) the estimated availability time of resource j - $EAT(j)$ that maintains the state of resource j and (2) the maximum estimated completion time among the parents of component i . Using the ECT values, the current scheduling heuristic is run to obtain a mapping for the current set of available components. This is repeated until the entire workflow is mapped. As a result, the mappings and makespans for each of the three heuristics are known. The scheduler finally chooses the mapping (or

“plan”) that gives the minimum makespan among the three.

We modified the original workflow scheduling algorithm to incorporate the batch queue wait times. Initially, for each heuristic, the estimated availability time of each of the resources is populated using the predicted wait time for the resource. We use the 95% upper bound on the median queue wait time prediction as the predicted wait time. Hence, during the scheduling process, the estimated completion time for a component takes into account the queue wait time (since ECT is a function of estimated availability time, rank and maximum ECT of parent components). We keep track of queue wait times for each cluster and the number of nodes that correspond to the queue wait time. With each new mapping to a resource j , we update EAT values of a specified number of nodes in the cluster (of resource j) with estimated queue wait time. A component will not need to wait on a resource if some other component had already been mapped to the resource (previous acquisition of the resource implies that wait time has been already been taken into consideration). We run the heuristics using the modified ECT values to obtain three mappings. We choose the mapping (“plan”) with the minimum overall makespan as the final schedule.

6 Experimental Procedure

The ultimate goal of this work is to determine how much a workflow schedule can be improved by integrating accurate resource performance models and batch-queue prediction. We have devised several experiments to show the effectiveness of such an integration.

The first experiment compares the makespan for workflow schedules using both performance model predictions and BMBP with those using only performance model information for resource selection. Accordingly, in our simulations we generate two schedules (one generated with batch-queue predictions taken into consideration and one without) and run each schedule serially. Repeating this experiment many times allows us to make a statistical comparison of the overall turnaround times.

The second experiment attempts to validate the results of the previous experiment in practice by reproducing the same conditions as before, except that we use realistic EMAN input data, resulting in much longer runtimes. For this experiment, we ran several instances of a real EMAN job using BMBP and non-BMBP schedules.

To better understand the experimental setup, this section will begin with a brief discussion of the EMAN application and workflow followed by synopsis of the five machines we used, and finally an in-depth description of the experimental testbed.

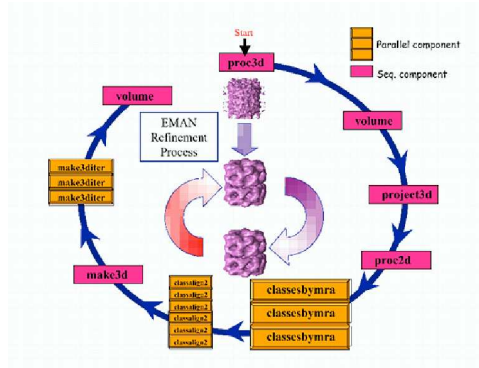


Figure 2. EMAN refinement workflow

6.1 Experimental Environment

For this work, we have chosen to use five supercomputers, of which three are part of the core TeraGrid [NSF] project, at five different locations around the country. The first is the Rice Terascale Cluster of 119 Intel Itanium 2 nodes located at Rice University in Houston, Texas. The second is a cluster of 128 Intel XEON nodes located at the University of California, Santa Barbara. Finally, we are using the SDSC, NCSA, and UC/ANL Teragrid machines which consist of 262, 887, and 62 Intel Itanium 2 processors and are located in San Diego, Urbana Champagne, and Chicago respectively. All of these systems are being monitored by the batch-queue prediction software infrastructure described in Section 4 and have had performance models for the individual resources pre-calculated (see Section 3). In addition these sites all use some combination of Maui and PBS to do scheduling and resource management.

6.2 The Application

We used the EMAN [Ludtke et al. 1999] application as a test workflow application. EMAN (Electron Micrograph Analysis) is a bio-imaging application developed at the Baylor College of Medicine. It primarily deals with 3D reconstruction of single particles from electron micrographs. Human expertise is needed to construct a preliminary 3D model from the “noisy” electron micrographs. Of the steps in the EMAN application workflow, the refinement from a preliminary 3D model to the final 3D model is the computationally intensive step that benefits most from harnessing the power of HPC systems. The EMAN refinement can be represented by the workflow depicted in Figure 2. It is essentially a linear workflow with some sequential and parallel stages. The important and time-consuming steps are the large parameter sweep steps like “classesbymra”. We use two EMAN workflows having essentially the same structure - (1) a regular one corresponding to the “rdv” data

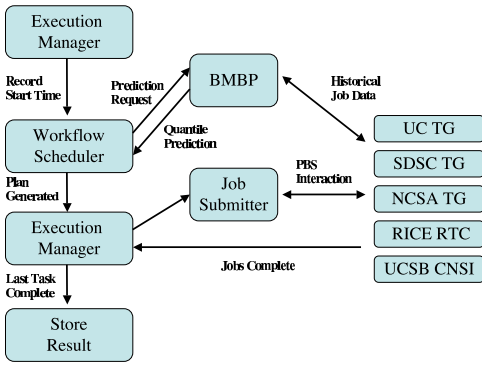


Figure 3. Experimental testbed architecture.

set and (2) a scaled down one corresponding to the "groel" data set.

6.3 Experimental Procedure

Our two experiments are similar, differing mainly in the size of the EMAN application we execute (scaled down or regular) and in the choice of workflow scheduler (BMBP-enhanced and non-BMBP enhanced versions).

Each measurement is initiated by taking UNIX time stamp to indicate when that measurement begins. Next, our novel workflow scheduler generates a "plan" of workflow task to resource mappings describing where and when to execute individual tasks. This step takes place in both the small and large EMAN experiments, using both the BMBP-enhanced version and the regular performance-model-only version of the workflow scheduler. Once the scheduler has produced the "plan," we feed it to our execution runtime manager, which is a simple program that automatically creates job-submission files for the various tasks in the plan and submits them to the site controlling the chosen individual resources. The jobs then wait in the site's batch queue; when they are executed, they first send a notice back to the execution manager indicating that the resource has become available and the application has begun execution. When the tasks finish, the job scripts send one final message to the manager indicating task completion. When the last task in the workflow has finished execution, the execution manager records a "finished" UNIX timestamp and the measurement is complete. At the end of each measurement, we are left with a total turnaround runtime (the "finished" timestamp minus the "started" timestamp) as well as which resources were chosen from which machines for each task in the workflow. Figure 3 shows the interaction points of each described component in our experimental testbed for a single experimental trial.

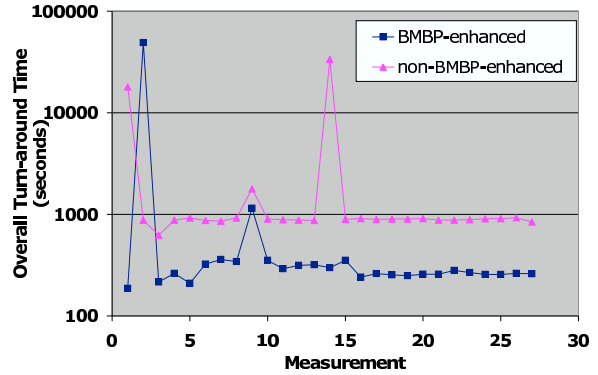


Figure 4. Total turn-around time for small EMAN runs for both BMBP enhanced schedules and non-BMBP enhanced schedules

7 Results

In this section, we will expose the results of our experiments and show how BMBP enhanced workflow schedules compare against schedules which used performance models alone.

Figure 4 shows the result of our first experiment, which compares the observed makespans from small EMAN runs using both BMBP enhanced schedules and schedules without any batch queue wait time knowledge. It is clear from the graph that although there are certain instances for which the BMBP schedule actually took longer than non-BMBP schedules, the majority of the experiments show that the use of BMBP significantly reduces the amount of time from workflow start to finish. Of the 27 measurement pairs, the BMBP-enhanced schedule produced a shorter makespan 26 times. The median BMBP enhanced schedule makespan was 262 seconds, while the median non-BMBP enhanced schedule was 895 seconds, a difference of 633 seconds (approx. 10.5 minutes). For most trials, the BMBP-enhanced schedules are producing observed makespan values which are roughly three times faster than non-BMBP-enhanced schedules, which is a substantial performance improvement considering the only difference is whether we take into account batch queue delays when calculating the makespans or use performance models alone.

Although we believe the reduction in observed makespan is mostly due to the fact that we are able to decide where to place tasks based on both resource performance models and batch queue wait time predictions, there are other related reasons for this benefit. One such interesting result of the experiment is shown in Table 1, which shows the average number of nodes and sites contacted for BMBP and non-BMBP schedule makespans. From this table, we can see that when the BMBP enhanced scheduler was used, the

	Avg. Res. Count	Avg. Site Count
BMBP	90.0	1.8
Non-BMBP	98.0	2

Table 1. Table of average number of unique resources and unique sites used for each conducted experiment.

plan used on average fewer unique nodes during its execution phase. Although we expected this to happen some of the time, we were surprised to see that in many of the experiments the BMBP enhanced plan used far fewer unique resources, which intuitively translates to less total batch queue waiting time. In cases where the plan used fewer nodes to complete the task, the scheduler has decided that it is more efficient to run some of the tasks in serial on a single node from one machine than it would be to run the tasks in parallel on multiple machines and incur large batch queue wait times. Without batch queue wait time predictions, such a decision would most likely never be made unless the performance models for one resource architecture predicted significantly longer task execution times, which was not the case for our fairly homogeneous machine set.

From the first experiment, it seems that using batch queue predictions to decide where best to run workflow tasks helps reduce the observed makespans. However, we understand that in a real environment there are potential hidden factors not directly related to batch queue wait times which could impact our relatively small makespans. For instance, an overloaded head node may take a few more seconds to complete a single submission process than an unloaded head node, and since we are submitting somewhere between 90 and 100 jobs per trial, these seconds could potentially add up to effect the overall makespan of a single experiment. For this reason, we attempted to perform a more realistic application execution in order to show that the difference is in fact due to our ability to choose resources that are likely to become available sooner than when we ignore batch queue wait times. Over a one week time period, we wished to gather enough BMBP and non-BMBP schedule measurements to show a similar figure and analysis as in our first experiment. The result of the experimental process was somewhat surprising as, unfortunately, none of the non-BMBP enhanced schedules were able to even complete due to a variety of scheduled downtime and machine hardware crashes during the experimental period. Recall that we run our experiments by first running a BMBP enhanced schedule, then a non-BMBP enhanced, and repeat. During our one week period, we first executed a BMBP schedule, which completed in approximately half a day (45384 seconds). The non-BMBP experiment began immediately thereafter, but after two days (172800 seconds) the experi-

ment had not completed and one of the sites being used suffered an unexpected power outage. When the machine was back online, we started the experiment again with a BMBP schedule which again took approximately half a day (49153 seconds) to complete. Again, a non-BMBP schedule began executing; somewhere between 1.5 and 2 days later, a different site brought its machine down due to an overheating machine room which again caused our trial to terminate without completing.

Given a large amount of experimental and allocation time, we expect eventually to be able to complete a few non-BMBP schedules, but this result is in some sense even more valuable in the short term. We are experiencing first hand the fact that in an environment where a single application is attempting to utilize widely distributed resources, one very real problem is that of resource failure from any one significant component. One potential solution that surely helps reduce the impact of failure is to execute parts of an application that require widely distributed resources as quickly as possible in order to avoid potential individual component failures. Since our BMBP schedules were able to schedule jobs on these resources in a way that resulted in relatively short makespans, around half a day, they were able to effectively avoid the downtimes that effected the non-BMBP schedules, for which portions of the application were still queued after 1.5 to 2 days.

8 Conclusions

The modern HPC user is beginning to face the problem of deciding where to execute his or her applications to achieve the shortest turnaround time but cannot effectively do so without having some idea of the amount of time their job will remain queued. Additionally, systems for scheduling application tasks across distributed resources commonly submit jobs to batch-queue-controlled resources. In both cases, the ability to have some notion of time spent waiting in queues is becoming a critical issue for the efficient mapping of tasks to resources.

In this work, we introduce the idea that although an application workflow scheduler may be able to intelligently select a task to resource mapping that is optimal in terms of process execution time, in reality the fact that resources are not instantly available can significantly impact the overall execution time of an application. In order to overcome this difficulty, we have integrated into a workflow scheduler our Binomial Method Batch Predictor (BMBP) which is used to predict bounds, with specified confidence, on the amount of time a single job is expected to wait in a batch queue before it's resources become available. Hypothesizing that such knowledge can help a scheduler better select resources to reduce the overall makespan of a real application (EMAN) on real systems (five HPC site machines in five disjoint loca-

tions in the US), we performed two experiments comparing BMBP-enhanced schedules to schedules which used performance models alone to determine task placement. In one experiment, we found that the plan produced by the BMBP enhanced scheduler resulted in observed makespans significantly smaller than the non-BMBP scheduler, and in the other experiment, the results showed that the use of BMBP enhanced schedules allowed the real EMAN application to successfully finish where not one of our non-BMBP schedules was able to complete since they ran for so long that at least one of the underlying machines failed causing the application to terminate.

In the near future, we will be integrating our batch-queue methodology into different state-of-the-art workflow schedulers in an attempt to verify our belief that such predictions are uniformly beneficial. We also are currently working on several techniques that use batch-queue predictions to implement a virtual resource provisioning system that would allow a user and/or system to secure the equivalent of an advanced reservation using batch-queue-controlled resources. Many of our techniques are also being integrated into the VGrADS infrastructure.

References

- BLYTHE, J., JAIN, S., DEELMAN, E., GIL, Y., VAHI, K., MANDAL, A., AND KENNEDY, K. 2005. Task Scheduling Strategies for Workflow-based Applications in Grids. In *IEEE International Symposium on Cluster Computing and the Grid (CCGrid 2005)*, IEEE Press.
- BREVIK, J., NURMI, D., AND WOLSKI, R. 2006. Predicting bounds on queuing delay for batch-scheduled parallel machines. In *Proceedings of PPOPP 2006*.
- DEELMAN, E., BLYTHE, J., GIL, Y., KESSELMAN, C., MEHTA, G., PATIL, S., SU, M.-H., VAHI, K., AND LIVNY, M. 2004. Pegasus: Mapping scientific workflows onto the grid. In *Proceedings of the 2nd European Across Grids Conference*.
- DOWNEY, A. 1997. Predicting queue times on space-sharing parallel computers. In *Proceedings of the 11th International Parallel Processing Symposium*.
- DOWNEY, A. 1997. Using queue time predictions for processor allocation. In *Proceedings of the 3rd Workshop on Job Scheduling Strategies for Parallel Processing*.
- GAREY, M. R., AND JOHNSON, D. S. 1979. *Computers and Intractability: A Guide to the Theory of Np-Completeness*.
- KERBYSON, D. J., ALME, H. J., HOISIE, A., PETRINI, F., WASSERMAN, H. J., AND GITTINGS, M. 2001. Predictive performance and scalability modeling of a large-scale application.
- KWOK, Y.-K., AND AHMAD, I. 1999. Static scheduling algorithms for allocating directed task graphs to multiprocessors. *ACM Computing Surveys* 31, 4, 406–471.
- LIFKA, D., HENDERSON, M., AND RAYL, K. 1995. Users guide to the argonne SP scheduling system. Tech. Rep. TM-201, Argonne National Laboratory, Mathematics and Computer Science Division, May.
- LUDTKE, S., BALDWIN, P., AND CHIU, W. 1999. Eman: Semiautomated software for high-resolution single-particle reconstructions. *J. Struct. Biol.* 128, 82–97.
- MANDAL, A., KENNEDY, K., KOELBEL, C., MARIN, G., MELLOR-CRUMMEY, J., LIU, B., AND JOHNSON, L. 2005. Scheduling Strategies for Mapping Application Workflows onto the Grid. In *14-th IEEE Symposium on High Performance Distributed Computing (HPDC14)*, 125–134.
- MARIN, G., AND MELLOR-CRUMMEY, J. 2004. Cross-architecture performance predictions for scientific applications using parameterized models. In *Proceedings of Joint International Conference on Measurement and Modeling of Computer Systems - Sigmetrics 2004*, 2–13.
- NSF TeraGrid Project. <http://www.teragrid.org/>.
- OH, H., AND HA, S. 1996. A static scheduling heuristic for heterogeneous processors. In *Euro-Par, Vol. II*, 573–577.
- PLLANA, S., AND FAHRINGER, T. 2005. Performance prophet: A performance modeling and prediction tool for parallel and distributed programs. In *Proceedings of The 2005 International Conference on Parallel Processing (ICPP-05)*.
- SIH, G., AND LEE, E. 1993. A compile-time scheduling heuristic for interconnection-constrained heterogeneous processor architecture. *IEEE Transactions on Parallel and Distributed Systems* 4, 2, 175–186.
- SMITH, W., TAYLOR, V. E., AND FOSTER, I. T. 1999. Using run-time predictions to estimate queue wait times and improve scheduler performance. In *IPPS/SPDP '99/JSSPP '99: Proceedings of the Job Scheduling Strategies for Parallel Processing*, Springer-Verlag, London, UK, 202–219.
- SUNDARAM-STUKEL, D., AND VERNON, M. K. 1999. Predictive analysis of a wavefront application using LogGP. *ACM SIGPLAN Notices* 34, 8, 141–150.

- TAYLOR, V., WU, X., AND STEVENS, R. 2003. Prophecy: an infrastructure for performance analysis and modeling of parallel and grid applications. *SIGMETRICS Perform. Eval. Rev.* 30, 4, 13–18.
- TOPCUOGLU, H., HARIRI, S., AND WU, M.-Y. 2002. Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE Transactions on Parallel and Distributed Systems* 13, 3, 260–274.
- TRACY D. BRAUN ET AL. 2001. A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems. *Journal of Parallel and Distributed Computing* 61, 810–837.
- WOLSKI, R., SPRING, N., AND HAYES, J. 1999. The network weather service: A distributed resource performance forecasting service for metacomputing. *Future Generation Computer Systems* 15, 5-6 (October), 757–768.
- WOLSKI, R., SPRING, N., AND HAYES, J. 1999. Predicting the cpu availability of time-shared unix systems on the computational grid. In *Proceedings 8th IEEE Symp. on High Performance Distributed Computing*.