A Service for Queue Prediction and Job Statistics

Warren Smith
Texas Advanced Computing Center
University of Texas at Austin
Research Office Complex 1.101, J.J. Pickle Research Campus
10100 Burnet Road (R8700)
Austin, Texas 78758-4497

***Abstract*—Science Gateways often execute computations on clusters managed by batch schedulers. These schedulers queue computations until resources are available and then execute them. Clusters are typically scarce resources so the amount of time a computation waits before it begins to execute can be significant. Gateways can potentially reduce the turn around time of their computations and improve their user experience if they use estimates of how long computations will wait in a queue before beginning to execute. This paper describes a service that provides such estimates as well as current and historical job information. An instance of this service has been deployed on TeraGrid and is available to TeraGrid science gateways.**

I. Introduction

Many Science Gateways [17] need to perform significant amounts of computation. One approach is for the gateway to purchase dedicated hardware to support this computation. This is often not feasible given funding constraints and the policies of funding agencies so an alternative of using shared computational resources is typically taken.

An example of this situation is the TeraGrid [14] and the TeraGrid science gateways [18]. The NSF-funded TeraGrid typically includes 10-15 computational clusters that support parallel computing as well as several special purpose systems for visualization, data storage, and serial computing. The number and mix of systems varies over time as older systems are retired and newer systems are brought on line. Atop this hardware is a common software environment that attempts to provide specific capabilities (e.g. job management, file transfer, environment management, information query) in a uniform manner.

There are a wide variety of science gateways that use TeraGrid resources to support their users. These gateways vary in their scientific domain, user interface, size of user community, the resources and services that they use, and so on. However, one way these gateways often use TeraGrid is to submit jobs for execution on TeraGrid clusters. Furthermore, a specific gateway often has access to a number of TeraGrid systems so the gateway needs to select a system to submit a job to.

One of the difficulties that a gateway faces when running a job on a batch-scheduled resource is the uncertainty about how long the job will wait before beginning to execute. This uncertainty has several impacts. For a job that has already been submitted, there is typically an end user or another step in a scientific workflow that is waiting for the job to complete. An end user would like an estimate of when the job will start so that they can plan their work and a workflow tool might like an estimate of when the job will start so that it can plan other steps in the workflow. For a job that is about to be submitted, a gateway needs to decide where to submit the job and with what configuration. Estimates of when a job would start on candidate systems allow the gateway to better pick where to submit a job. Estimates of when a job would start using different configurations (number of processing cores and run time) allow a gateway to pick a configuration that minimizes total turn around time for the job.

The Karnak service described in this work attempts to address the difficulties described above by providing predictions of when batch schedulers will execute jobs. The service provides predictions for hypothetical jobs that may be submitted as well as jobs that have already been submitted. In addition, the service provides current and historical information about jobs so that users can apply their own heuristics to select where to submit computations.

This service has been deployed on the TeraGrid and currently provides predictions and job information for several TeraGrid resources. The Karnak service provides information to clients in several formats using a REST [5] style. The server can be accessed via a web browser, command line clients, and via the TeraGrid User Portal.

The remainder of this paper describes the specific capabilities provided by the service, provides information about the use cases supported by these capabilities, discusses the interfaces to the service, analyzes the prediction performance of the service, and provides implementation details. The paper ends with a discussion of related work and future work.

II. Capabilities

The Karnak service provides several types of predictions as well as current and historical information about the systems it is monitoring. The two major types of predictions available are for jobs that have already been submitted and for jobs that may be submitted. This is an important distinction and, as described in the implementation section below, are handled separately by the service.

For jobs that have already been submitted, the service provides a prediction for a job identified by the user. The user

978-1-4244-9752-2/11/$26.00 ©2011 IEEE

identifies a job using a system name and the job identifier provided by the batch scheduler for that system at submission time. The prediction takes the form of either a duration or a date and time and a confidence interval. The service allows the user to select whether they would like a prediction formatted as the date and time the job will start or as a duration from the current time that the job will wait in the queue before starting. The confidence interval is an important piece of information because it indicates how accurate the service expects the prediction to be. The X% confidence interval for a prediction p is expressed as a duration d such that X% of the time, the actual value will fall within the interval $[p - d, p + d]$.

The service also provides predictions for jobs that have not yet been submitted. In this case, the user describes their job using the number of processing cores and the amount of wall time that will be requested. In addition, the user specifies a set of system-queue pairs. The service then provides predictions for the specified job on each of the system-queue pairs. Similarly to above, the predictions take the form of either a start date/time or a wait time and a confidence interval.

Finally the service provides statistical information about jobs that are currently being managed by the schedulers of systems it is monitoring and about historical jobs. This information allows users to observe the current state of resources, see how quickly resources have started jobs in the recent past, and perform their own heuristics for resource selection.

III. Use Cases

This service is useful in a number of situations. One way to use the service is to use the information it provides to help perform long-term resource selection (e.g. deciding which systems to apply for allocation on). For example, users can look at historical information about job sizes and queue wait times to decide if a system is a good fit for their job mix and to get a general idea of how long their jobs would wait on a system.

A gateway can use predictions of jobs that may be submitted as part of the resource selection process. The gateway can combine these predictions (and confidence intervals) with information about run time performance, system reliability, remaining allocation, and so on when deciding where to submit a job. In addition to selecting a cluster, these predictions can be used to select queues and job configurations. For example, if a job has a deadline and isn't predicted to complete before that deadline in the normal priority queue, the job could be submitted to a higher priority/cost queue. The gateway could also ask for predictions about various processing core and run time combinations to see if any of them have significantly earlier expected completion times.

Similarly, a gateway can use the current and historical job information provided by the Karnak service as part of the resource selection process. If a gateway wishes to use its own process to estimate queue wait times, the gateway can use statistical job information from the Karnak service to do this.

After jobs have been submitted, a gateway can use the predictions provided by the Karnak service to provide estimates to gateway users. Such information improves the user experience as well as allows gateway users to better plan their activities. A gateway can also use predictions of submitted jobs to optimize scientific workflows. The gateway can replan a workflow if the predictions for queued jobs diverge from earlier expectation. Predictions can also be used to submit dependent jobs before the job they depend upon complete and reduce the total time to execute a workflow.

IV. Interfaces

The Karnak service provides several different interfaces to support different user communities. The service itself is accessed using the REST style [5]. The service uses the HTTP protocol with data formatted as HTML, XML, JSON, or plain text.

The HTML format allows users to directly access the service using a web browser [16]. The web pages that are viewed this way are very simple, but all of the information provided by the service is accessible. A few simple HTML forms are also provided so that users can submit descriptions of hypothetical jobs that they would like predictions for.

The XML format supports easy parsing in a variety of programming languages. The JSON format is also easily parseable and is also convenient for the developers of web portals and gateways since such projects are often already using JSON. Finally, the plain text interface supports the implementation of very thin client programs as described next.

A. Command Line Interface

A set of simple programs provides command line access to the Karnak service. The programs are implemented using Python, have no external dependencies, and provide output as plain text (the default), XML, or HTML. The programs naturally group into ones that provide information and ones that provide predictions. The following programs provide information:

- The `ksystems` program provides a high level description of current system status. It lists the systems known to the Karnak service and provides a count of the number of running jobs, number of waiting jobs, and number of used processing cores for each system.
- The `ksystem` program provides a little more detail about a single system. The detail include the queues available on the system and a count of the number of running jobs, number of waiting jobs, and number of used processing cores for each queue;
- The `kqueue` program provides current and historical information about jobs in a specific queue. The current information again includes a count of the number of running jobs, number of waiting jobs, and number of used processing cores. The historical information includes information about jobs that have started and completed in the last hour, the last four hours, the last day, and the last week. For each of these time frames, a job count, average number of processing cores, average requested run time, and either the average wait time (for jobs that

have started) or the average run time (for jobs that have completed.

- The `kjobs` program provides summary information about jobs queued and waiting to run on a system. The information about each job includes the identifier assigned by the batch scheduler, the time the job was submitted, the number of processing cores requested by the job, and the amount of run time requested by the job.

The following programs provide predictions:

- The `kwait` command provides a predicted wait time for a job that has already been submitted to a scheduler. The user provides the system name and job identifier selected by the batch scheduler on the command line. The program outputs the job identifier, number of processing cores requested, amount of run time requested a predicted wait time, and a 90% confidence interval.
- The `kstart` command provides a predicted start date/time for a job that has already been submitted to a scheduler. The user provides the system name and job identifier selected by the batch scheduler on the command line. The program outputs the job identifier, number of processing cores requested, amount of run time requested a predicted wait time, and a 90% confidence interval.
- The `kwouldwait` program provides wait time predictions for a hypothetical job. The user provides a number of processing cores and requested wall time on the command line as well as a list of queue/system pairs. The program provides a predicted wait time and 90% confidence interval for each of the specified queues and systems.
- The `kwouldstart` program provides start date/time predictions for a hypothetical job. The user provides a number of processing cores and requested wall time on the command line as well as a list of queue/system pairs. The program provides a predicted start date/time and 90% confidence interval for each of the specified queues and systems.

B. TeraGrid User Portal Interface

The TeraGrid User Portal has recently been enhanced to incorporate predictions from the TeraGrid Karnak service. This is currently done in two different ways. First, after a user logs in to the portal, the user can request queue wait predictions for a job she may want to submit. This functionality is provided in a stand alone "HPC Queue Prediction" page where the user can specify the number of processing cores, duration of execution, and the queues and systems to predict for. This page has several convenient features such as automatically selecting default queues, allowing the user to select and deselect queues and systems, and providing both a predicted wait time (e.g. 4 hours) and a predicted start time (e.g. today at 2pm).

The second way in which Karnak predictions are incorporated into the TeraGrid User Portal is when a user views information about her jobs on TeraGrid systems on the "My Jobs" page. This page shows information about running and waiting jobs. For waiting jobs, the provided information includes a prediction from the Karnak service.

V. Performance

The prediction accuracy of the TeraGrid Karnak service is not currently being monitored for all of the systems that it provides predictions for. Therefore, this section presents the performance of the prediction technique used by the service on historical data. The procedure is to optimize the prediction accuracy over a workload consisting of several weeks of historical job information and present the prediction performance after this optimization.

Two types of workloads are generated. The first type of workload contains information about jobs that are about to be submitted to a scheduler. For each of these jobs, a prediction is made for the job just before it is submitted and information about the wait time of the job is incorporated into the predictor as the job begins to execute. The second type of workload contains information about jobs that are already queued. For each of these jobs, predictions are made the first time they are seen in a queue. When a job starts, information about the wait time of the job is incorporated into the predictor.

Performance results are available for 5 TeraGrid resources (Ranger, Abe, Lonestar, Cobalt, Pople) using jobs that started in August of 2010. For Ranger, Abe, and Lonestar, the results are for jobs that started between August 23rd and August 31st. For Cobalt and Pople, the results are for jobs that started any time in August. The difference in chosen time ranges is due to the volume of jobs each system handles.

There are a variety of different metrics that can be used to quantify how well predictions are performed. We choose to use a metric known as accuracy. The accuracy of a queue wait time estimate E_{wt} for an actual wait time of A_{wt} is:

$$accuracy = \begin{cases} 1 & \text{if } E_{wt} = A_{wt}, \\ A_{wt}/E_{wt} & \text{if } E_{wt} > A_{wt}, \\ E_{wt}/A_{wt} & \text{if } E_{wt} < A_{wt} \end{cases}$$

Good predictions result in accuracies close to (but not above) 1 and bad predictions result in accuracies close to (but not below) 0. A good characteristic of this metric is that due to the range of values that accuracy can take, it limits the negative impact on average accuracy of any single prediction. This is in contrast to a metric such as average prediction error where a few very poor predictions can have a significant impact. Another good characteristic of this metric is that it is always defined. A metric such as percent error is not defined when the actual queue wait time of a job is 0. On the other hand, a poor characteristic of this metric is that it can give disproportionate weight to small errors that are of no consequence to users. An example is when a job is predicted to wait for 1 minute but it actually starts immediately. This situation is described by an accuracy of 0.

Table I shows the results when predicting the wait times of jobs before they are submitted. There is significant variation in the workloads with the number of jobs predicted ranging from 2,402 (Cobalt) to 43,021 (Abe) and the average wait time

ranging from 0.83 hours (Abe) to 7.14 hours (Cobalt) hours. Furthermore, if we look at other time ranges not shown in the table, the mean wait time can vary significantly even on the same system.

The table also shows that prediction accuracy ranges from 0.34 to 0.57. The lowest prediction accuracies occur on the Cobalt and Pople systems. Two potential reasons for this are because they have the longest average queue wait times and because they have the fewest data points to use to make predictions. These accuracies are lower than we would like, but it is difficult to determine if they are lower than we should expect. This is caused by different research groups that develop queue wait time prediction techniques using different metrics and different workloads for evaluations. One study [13] that simulated a simple scheduling algorithm on 4 workloads (not the workloads used here) found that the prediction algorithm used in the QBETS [3] tool resulted in accuracies that ranged from 0.20 to 0.72. This may indicate that the prediction accuracies presented here are typical.

In addition to accuracy, another metric to examine is the percent of actual queue wait times that fall within the predicted 90% confidence interval. This is shown in the rightmost column of Table I and shows that this varies from 79% to 86%. These values are below 90% so we still have some improvements to make in our technique for calculating confidence intervals.

Table II shows results when predicting the wait times of jobs that have already been queued. These workloads cover the same time period as the workloads in the previous table but have fewer data points because some jobs start so quickly that they do not appear in the queue. This is an effect of how job data is gathered on TeraGrid: Snapshots of the queue are taken several minutes apart and published into the TeraGrid information service.

The table shows that the prediction accuracy ranges from 0.33 to 0.45. This is lower accuracy than was achieved when predicting jobs before they are submitted. A contributing factor is that the jobs that start very quickly aren't included in this data are they are the jobs that are easier to predict. If we ignore jobs that wait less than 10 minutes before starting, the prediction accuracies in Table I decrease by an average of 0.10.

The accuracies of these predictions are again lower than we would like. For one point of comparison, [13] used a technique of predicting queue wait times by predicting application execution times and simulating a simple FCFS scheduling algorithm. The accuracy of this technique on their workloads ranged from 0.35 to 0.69. These accuracies are higher than the ones presented here but this is not really a fair comparison because a FCFS scheduling policy is much simpler than those that are used in practice on TeraGrid systems.

The percent of the predictions that are within the 90% confidence interval is shown in the rightmost column of Table II. These percentages vary from 70% to 90% and shows again that we need to make improvements in how we calculate confidence intervals.

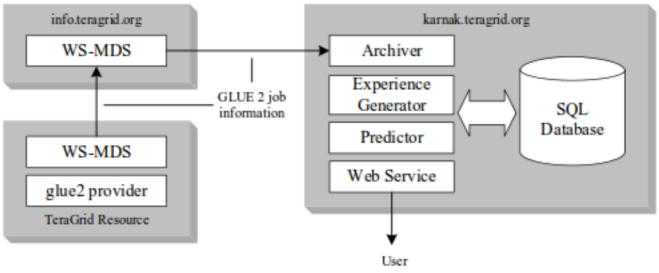

Fig. 1. Architecture of the Karnak deployment on TeraGrid.

VI. Implementation

The implementation of the Karnak service was challenging for several reasons. One challenge was creating an architecture that provides acceptable response times, reliability, and simplicity while processing a continuous stream of information and user requests. This needed to be accomplished when running on a virtual machine with constrained resources. The next section describes the architecture developed to meet these goals. Another challenge was implementing a prediction technique that provides sufficiently accurate predictions and fast response times. The technique used by the service is described in Section VI-B.

A. Karnak Architecture on TeraGrid

The source of job information for the Karnak service on the TeraGrid is the TeraGrid GLUE 2 implementation [15]. GLUE 2 [1] is a data definition standardized by the Open Grid Forum to describe distributed infrastructures like the TeraGrid. TeraGrid represents GLUE 2 information using a TeraGrid-defined XML schema. As shown in Figure 1, a glue2 information provider is running on a TeraGrid resource (cluster), is gathering information about that resource, and is producing XML documents with this information. The TeraGrid uses the Globus WS-MDS [6] to publish this information from the resource to the centralized TeraGrid information services including a centralized WS-MDS instance. Finally, the Karnak service queries the centralized TeraGrid WS-MDS for GLUE 2 job information.

The Karnak implementation consists of several components that collaborate to provide predictions and information to users. The Archiver daemon retrieves GLUE 2 job information and writes information to a SQL database. The GLUE 2 job information from a resource is organized as a list of jobs that represent a snapshot of the queue state on that resource at a specific time. These snapshots change over time as jobs complete, start, and are submitted. The Archiver processes these snapshots into records describing the state changes of each job and also stores the order of the jobs in each snapshot.

The Archiver is the only component of the Karnak service that is dependent on the infrastructure on which it is deployed. This daemon must be able to retrieve and process job information and the mechanisms for doing this and the

TABLE I
PREDICTION PERFORMANCE FOR JOBS THAT ARE NOT YET QUEUED.

System	Number of Jobs	Mean Wait Time (hours)	Accuracy	Mean Confidence Interval (hours)	Percent in Confidence Interval
Ranger	15210	2.63	0.43	10.74	80.16
Abe	43021	0.83	0.57	0.53	80.38
Lonestar	36047	1.12	0.53	1.51	79.05
Cobalt	5871	7.14	0.39	12.00	79.49
Pople	2402	6.92	0.34	15.86	85.55

TABLE II
PREDICTION PERFORMANCE FOR QUEUED JOBS.

System	Number of Jobs	Mean Wait Time (hours)	Accuracy	Mean Confidence Interval (hours)	Percent in Confidence Interval
Ranger	9861	3.11	0.33	8.49	84.36
Abe	11520	2.56	0.40	2.76	69.61
Lonestar	14238	2.06	0.45	3.32	82.27
Cobalt	3513	11.80	0.36	14.12	73.33
Pople	1114	11.42	0.44	25.89	90.04

data formats can vary. However, this dependency is contained within this single component which makes it feasible to deploy the Karnak service in other infrastructures.

The Experience Generator performs two different functions. First, it generates experiences as jobs on TeraGrid resources begin to execute and inserts these experiences into the SQL database. For each job that starts, it generates an experience that describes it and the queue state just before the job is submitted. This daemon also generates one or more experiences that describe the job and the queue state at various times as the job is waiting to execute. Second, it performs scheduling simulations as needed. Simulations are performed using current job information as queue states change over time. Simulations are also performed for historical queue states if a simulation at the time needed to generate an experience is not available.

The Predictor loads experiences from the SQL database and provides predictions on request. The Predictor is embedded in the Web Service and generates predictions using the techniques described in the next section.

Finally, the Web Service [16] implements the REST interface and interacts with a variety of clients. The Web Service also interacts with the Predictor to make predictions and the SQL database to provide information about current and historical jobs.

B. Prediction Technique

The Karnak service uses a prediction algorithm and implementation that has been developed previously [11]. The algorithm forms predictions using instance-based learning techniques [2]. An overview of instance-based learning is shown in Figure 2. The first step in forming a prediction is that **a query is presented to the experience base and relevant experiences are returned.** An experience describes something that happened in the past and consists of input and output features. Input features describe the conditions under which an experience was observed and the output features describe what happened under those conditions. Each feature typically

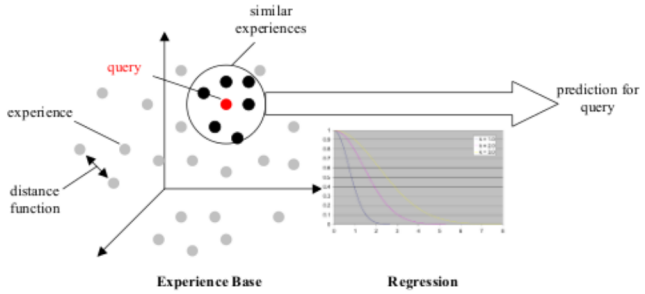

Fig. 2. An overview of instance-based learning.

consists of a name and a value where the value is of a simple type such as integer, floating point number, or string. A query is an experience with only input features where we are trying to predict the output features.

The Karnak service uses two different experiences when predicting queue wait times. The first type of experience is shown in Table III and describes a job that has not yet been submitted to a batch scheduler. The amount of information in this experience is somewhat limited for several reasons. First, before the job is submitted, the Karnak service does not know how the scheduler will order it relative to other jobs and therefore cannot perform a scheduling simulation that includes this job. Second, because the Karnak service does not exchange sensitive information with a user (such as user and project names on TeraGrid resources), it does not know this information about a job that has not yet been submitted.

The second type of experience is shown in Table IV and describes a job that has been submitted to a batch scheduler. In comparison to Table III, you can see that additional information is included because the job has been ordered by the scheduler relative to other jobs and the Karnak service can access sensitive information about queued jobs. One interesting piece of information included in this experience is

TABLE III
EXPERIENCE USED FOR JOBS THAT ARE NOT YET SUBMITTED.

Input Features		
Feature	**Type**	**Description**
Queue	String	Name of queue job submitted to
Cores	Integer	Number of processing cores requested
Requested Run Time	Integer	User-specified maximum execution time
Count	Integer	Number of jobs ahead
Work	Long	Amount of work waiting ahead
Jobs Running	Boolean	Whether jobs are being run or not
Time	Long	Time experience was generated
Output Features		
Feature	**Type**	**Description**
Wait Time	Integer	Amount of time job waited before starting

a simulated wait time that is generated by running a relatively simple scheduling simulation over the jobs being managed by a scheduler, as described in Section VI-A

Experiences are stored in an experience base: A specialized database that stores experiences for efficient insertion and search. This database is not a relational database as it is optimized to return experiences that are relevant or similar to a query. The definition of similarity we use (described next) results in a metric space and we use an MTree [4] to improve query times.

The similarity of two experiences (or a query and an experience) is quantified by a distance function. There are a variety of distance functions that can be used [19] and we have chosen to use the Heterogeneous Euclidean Overlap Metric. This distance function can be used on features that are linear (numbers) or nominal (strings). We require support for nominal values because important features such as queue names, user names, and project names are nominal. The distance between two experiences x and y is expressed as

$$D(x,y) = \sqrt{\sum_f w_f d_f(x,y)^2}$$

where f is a feature, d_f is the per-feature distance, and w_f is a feature weight. The feature weight allows us to indicate that it is more important that some features are similar than other features, but we must select these feature weights appropriately. The per-feature distance is

$$d_f(x,y) = \begin{cases} 1 & \text{if } x_f \text{ or } y_f \text{ is unknown,} \\ \mathit{overlap}_f(x,y) & \text{if } f \text{ is nominal,} \\ \mathit{diff}_f(x,y) & \text{if } f \text{ is linear} \end{cases}$$

with the following definitions:

$$\mathit{overlap}_f(x,y) = \begin{cases} 0 & \text{if } x_f = y_f \\ 1 & \text{otherwise} \end{cases} \qquad \mathit{diff}_f = \frac{|x_f - y_f|}{max_f - min_f}$$

The per-feature distance between two nominal values is 0 if they are the same and 1 otherwise. For linear values, the distance is their difference scaled by the range of values for that feature in the experience base. This scales the per-feature distance to be between 0 and 1.

We use two different approaches to find similar experiences. One approach is to return the N nearest neighbors to a query as the relevant experiences. The other approach is to return all experiences within some range r (by distance) of the query. These two approaches perform better in different situations so we typically use both. Therefore, when performing predictions, we must select a value for N or r.

The final step to form a prediction is that **an estimated value and confidence is constructed for each output feature using relevant experiences.** For this work, the output features that we are predicting are linear so we provide an estimated value and a confidence interval. The X% confidence interval provides a +- value around an estimate that the actual value should be in X% of the time.

We are currently using a kernel regression to form these estimates and confidence intervals, although there are a number of different ways it can be constructed. The equation to form an estimate E for output feature f of a query point q is

$$E_f(q) = \frac{\sum_e K(D(q,e))e_f}{\sum_e K(D(q,e))}$$

where K is the kernel function, D is the distance function described previously, e is one of the N nearest experiences to q in the experience base, and e_f is the value of feature f of experience e. This kernel regression performs a distance-weighted average of the values of the output feature in relevant experiences to create a prediction and confidence interval. A distance-weighted average is performed under the assumption that experiences that are more similar to a query will have more similar results.

The kernel function used to weight the values of features should approach a constant value as the distance approaches 0 and should approach 0 as the distance function approaches infinity. There are a wide variety of kernel functions that can be used. We experimented with several different functions but did not find one that consistently resulted in better predictions. We therefore use a simple Gaussian function as our kernel. We also include a kernel weight parameter k so that we can compact or stretch the kernel to give lower or higher weights to experiences that are farther away. This results in a kernel function of

$$K(d) = e^{-(\frac{d}{k})^2}$$

In addition to searching the experience base to form predictions, we also **insert experiences into the experience base**

TABLE IV
EXPERIENCE USED FOR JOBS THAT HAVE BEEN SUBMITTED.

Input Features		
Feature	**Type**	**Description**
Job Name	String	Name provided by the user
User	String	User submitting the job
Project	String	Allocation to charge job to
Queue	String	Name of queue job submitted to
Cores	Integer	Number of processing cores requested
Requested Run Time	Integer	User-specified maximum execution time
Count	Integer	Number of jobs ahead
Work	Long	Amount of work waiting ahead
User Count	Integer	Number of jobs ahead from this user
User Work	Long	Amount of work waiting ahead from this user
Jobs Running	Boolean	Whether jobs are being run or not
Simulated Wait Time	Integer	Amount of time a scheduling simulations says the job will wait
Time	Long	Time experience was generated
Output Features		
Feature	**Type**	**Description**
Wait Time	Integer	Amount of time job waited before starting

when more information is available. For example, as jobs start or complete during the day or as file transfers complete. This is relatively straightforward, but we must specify the maximum size of the experience base and the policy to use when the experience base exceeds this maximum size. We have chosen a simple FIFO replacement policy under the assumption that the oldest experiences are the least relevant.

In the previous discussion, we described the parameters that need to be selected: the number of nearest neighbors or range, the feature weights, the kernel width, and the maximum experience base size. Choices for these parameters can have a significant affect on prediction accuracy. Our approach to determine the optimum values for these parameters is to **perform a genetic algorithm search over training data to try different configurations and maximize the prediction accuracy**. We then use the results of these searches when making future predictions.

Genetic algorithms [7] are a stochastic optimization technique that is particularly effective when the search space contains many local maxima, such as in our problem. Its stochastic component allows it to often avoid being trapped in a local maxima and therefore have a better chance to find a global maxima. In this way, it is similar to other stochastic optimization techniques, such as simulated annealing [8].

Selecting the optimum configuration for a genetic algorithm search requires trial and error. Over a period of time, we have arrived at a configuration that performs reasonably well. This configuration uses multiple populations in each generation with migration between populations every few generations, crossover with rank-based parent selection so that more fit individuals are guaranteed to be parents of the next generation, mutation of a small fraction of the population, elitism where the few best individuals survive unchanged to the next generation, and the search continues until there are several generations with no improvement in fitness.

However, performing these searches can take a nontrivial amount of time. A separate search needs to be performed for each system and these searches need to be performed periodically (e.g. every month). We have therefore begun to identify prediction configurations that are generally good across systems and time periods. This allows us to perform searches less often. In addition, we are investigating on-line techniques to adjust prediction configurations on the fly rather than performing off-line genetic algorithm searches.

VII. RELATED WORK

The QBETS batch queue prediction service [3] has been available to TeraGrid users for some time. The service can be queried using simple command line programs and has been integrated into the TeraGrid User Portal. For a number of reasons, even though QBETS is available to TeraGrid users, the TeraGrid no longer considers it a production service. QBETS provides two types of predictions. It provides a probability that a hypothetical job will start by a user-specified deadline. Alternatively, it provides a time in the future where the hypothetical job will start before then X% of the time. The user specifies the value of X. It is difficult to compare the performance of Karnak and QBETS because, in addition to needing to use the same set of workloads for comparison, the services provide different information to users. One difference to note is that QBETS only provides predictions of hypothetical jobs while Karnak predicts hypothetical jobs as well as jobs that are queued.

In [11], we describe how the prediction techniques that are used by Karnak can be used to predict queue wait times, job run times, and file transfer times. We also describe prototype services that we implemented at that time. We previously used a categorization approach to predict queue wait times [12]. Other work predicts wait times using the same categorization technique as we used [9] and the same instance-based learning approach we present here [10].

An alternate approach that we [12] and others [13] have explored in the past is to compute wait time predictions by performing simulations of the exact scheduling algorithm and using predictions of job run times in the simulations. The scheduling algorithms used on TeraGrid systems are complex

and difficult to duplicate. The Karnak service therefore performs approximate scheduling simulations and incorporates those simulated wait times into the experiences used for instance based learning.

VIII. Conclusions and Future Work

This paper presents the Karnak service that provides predictions of how long jobs will wait in batch scheduling queues before they begin to execute and provides statistical information about current and historical jobs. This service has been deployed on the TeraGrid and currently provides predictions for several TeraGrid resources. When looking at historical data, the prediction accuracy for 5 TeraGrid systems ranges from 0.34 to 0.57 for jobs before they are submitted and 0.33 to 0.45 for jobs that are waiting in the queue.

For a job that has not been submitted, a gateway needs to decide where to submit the job and with what configuration. The Karnak service provides estimates of when a job would start on candidate systems and allows the gateway to better pick where to submit a job. In addition, a gateway can use the Karnak service to get estimates of when a job would start using different configurations (number of processing cores and run time) and then pick a configuration that minimizes total turn around time for the job.

For a job that has already been submitted by a gateway, there is typically an end user or another step in a scientific workflow that is waiting for the job to complete. A gateway can query the Karnak service and provide the resulting predictions to end users so that they can plan their work. The gateway can use such predictions itself to execute workflows more efficiently. For example, the gateway can use an estimate of when a job will start to plan other steps in a scientific workflow.

We have a number of future plans for the Karnak service. We will continue to improve the quality of the predictions provided by the service. This will be accomplished by experimenting with variations of the machine learning techniques we are using, by improving the accuracy of the scheduling simulations, and by investigating alternative techniques for computing confidence intervals. We will also continue to improve the reliability of the service and move the service to full production on TeraGrid.

We will also collaborate with users to enhance the service to provide the capabilities that they need. One particular area we wish to investigate with users is providing additional statistical job information. The currently available information is predefined and users may want to retrieve customized information. For example, the service could provide information about any jobs that satisfy a user-specified filter that describes job characteristics. In addition, the information provided about jobs could be enhanced to include distributions or even raw wait times.

Another potential area of future work is to provide predictions for additional properties. For example, our previous work [11] includes predicting job run times and file transfer times. These types of predictions could be provided by the Karnak service if they are of use to gateways and end users.

IX. Acknowledgments

This work was supported by the TeraGrid under National Science Foundation grant number 0503697.

References

[1] S. Andreozzi, S. Burke, F. Ehm, L. Field, G. Galang, B. Konya, M. Litmaath, P. Millar, and J. Navarro. GLUE Specification v. 2.0. Technical Report GFD-R-P.147, The Open Grid Forum, March 2009.

[2] C. Atkeson, A. Moore, and S. Schaal. Locally Weighted Learning. *Artificial Intelligence Review*, 11:11–73, 1997.

[3] J. Brevik, D. Nurmi, and R. Wolski. Predicting Bounds on Queuing Delay for Batch-scheduled Parallel Machines. In *Proceedings of ACM Principles and Practices of Parallel Programming*, March 2006.

[4] P. Ciaccia, M. Patella, and P. Zezula. M-tree: An Efficient Access Method for Similarity Search in Metric Spaces. In *Proceedings of the 23rd VLDB Conference*, 1997.

[5] R. T. Fielding and R. N. Taylor. Principled Design of the Modern Web Architecture. *ACM Trans. Internet Technol.*, 2:115–150, May 2002.

[6] I. Foster. Globus Toolkit Version 4: Software for Service-Oriented Systems. *Lecture Notes in Computer Science*, 3779:2–13, 2005.

[7] D. E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, 1989.

[8] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science, Number 4598, 13 May 1983*, 220, 4598:671–680, 1983.

[9] H. Li, D. Groep, J. Templon, and L. Wolters. Predicting Job Start Times on Clusters. In *Proceedings of the IEEE International Symposium on Cluster Computing and the Grid*, pages 301–308, April 2004.

[10] H. Li, D. Groep, and L. Wolters. Efficient Response Time Predictions by Exploiting Application and Resource State Similarities. In *Proceedings of the 6th IEEE/ACM International Workshop on Grid Computing*, pages 234–241, November 2005.

[11] W. Smith. Prediction Services for Distributed Computing. In *Proceedings of the 2007 International Parallel and Distributed Processing Symposium*, March 2007.

[12] W. Smith, V. Taylor, and I. Foster. Using Run-Time Predictions to Estimate Queue Wait Times and Improve Scheduler Performance. In *Proceedings of the IPPS/SPDP'99 Workshop on Job Scheduling Strategies for Parallel Processing*, 1999.

[13] O. Sonmez, N. Yigitbasi, A. Iosup, and D. Epema. Trace-Based Evaluation of Job Runtime and Queue Wait Time Predictions in Grids. In *Proceedings of the 18th ACM international symposium on High Performance Distributed Computing*, HPDC '09, pages 111–120, New York, NY, USA, 2009. ACM.

[14] The TeraGrid. http://www.teragrid.org.

[15] TeraGrid CTSS v4 GLUE2 Information Providers. http://software.teragrid.org/pacman/ctss4/glue2/.

[16] The Karnak Prediction Service. http://www.karnak.teragrid.org.

[17] N. Wilkins-Diehr. Science Gateways - Common Community Interfaces to Grid Resources. *Concurrency and Computation: Practice and Experience*, 19(6):743–749, 2007.

[18] N. Wilkins-Diehr, D. Gannon, G. Klimeck, S. Oster, and S. Pamidighantam. TeraGrid Science Gateways and Their Impact on Science. *IEEE Computer*, 41(11):32–41, 2008.

[19] D. R. Wilson and T. R. Martinez. Improved Heterogeneous Distance Functions. *Journal of Artificial Intelligence Research*, 6:1–34, 1997.