Prediction Services for Distributed Computing

Warren Smith
Texas Advanced Computing Center
The University of Texas at Austin
wsmith@tacc.utexas.edu

Abstract

Users of distributed systems such as the TeraGrid and Open Science Grid can execute their applications on many different systems. We wish to help such users, or the grid schedulers they use, select where to run applications by providing predictions of when tasks will complete if sent to different systems. We make predictions of file transfer times, batch scheduler queue wait times, and application execution times using historical information and instance-based learning techniques. Our prediction errors for data from the TACC lonestar system are 37 percent of mean file transfer time, 115 percent for mean queue wait time, and 72 percent of mean execution time. Our approach achieves significantly lower prediction error on other workloads. We have wrapped these prediction techniques with web services, making predictions available to users of distributed systems as well as tools such as resource brokers and metaschedulers.

1. Introduction

In recent years, large distributed systems have been deployed to support scientific research. Examples are the TeraGrid [21], the Open Science Grid [17] (OSG), and the Enabling Grids for E-scienceE [7] (EGEE) grid. The existence of such grids allows users to execute their applications on a variety of different computer systems. The obvious question users have each time they wish to run an application is which computer system should they use? Many factors go into making this decision: The computer systems that the user has access to, the user's remaining allocations on these systems, the cost of using different systems, and so on. One important piece of information that users would like, which we provide in this work, is predictions of when an application would complete if sent to a specific parallel computing system.

1-4244-0910-1/07/$20.00

The main components of this overall prediction are predictions of file transfer times, batch scheduler queue wait times, and execution times. We keep historical databases of this information and use instance-based learning techniques to form predictions from this data. To form a prediction, the instance-based learning techniques examine the appropriate database to find relevant past experiences and then form a prediction based on what happened in those past experiences.

We have wrapped these prediction techniques with web services so that users and tools can easily access them. Users can access predictions via command line tools or the predictions can be integrated into user portals and science gateways. Tools such as resource brokers and metaschedulers can also use these prediction web services to select where to execute user jobs in an automated fashion.

The next section presents our instance-based learning approach to prediction. Subsequent sections describe how we apply this technique to predicting the execution times, queue wait times, and file transfer times and analyze the performance of our approach. Section 6 describes our prediction services and the final section presents conclusions and future work.

2. Prediction Technique

We form predictions using instance-based learning (IBL) techniques [2]. An overview of instance-based learning is shown in Figure 1. 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 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.

Experiences are stored in an experience base: A special-

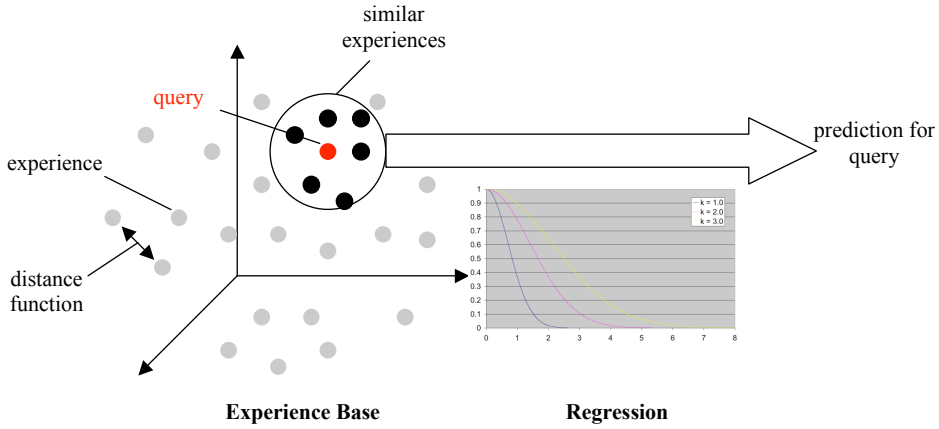

Figure 1. An overview of instance-based learning.

ized 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 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 [23] 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 the names of hosts, executables, users, and queues 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 approximately scales the distance to be between 0 and 1.

In our approach, we chose to return the N nearest neighbors to a query as the relevant experiences. An alternative would be to return all experiences within some distance of the query, but we found that using nearest neighbors generally resulted in slightly more accurate predictions. Therefore, when performing predictions, we must select a value for N.

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 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, 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 minimize the prediction error**. We then use the results of these searches when making future predictions.

Genetic algorithms [11] are a stochastic optimization technique that is particularly effective when the search space contains many local maxima, such as in our problem. It's 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 [14]. We have tried hill climbing and simulated annealing approaches to optimize our prediction parameters, but have had much better results using a genetic algorithm.

3. Predicting Execution Times

We use the instance-based learning techniques previously described to predict execution times. To perform these predictions, we need to define what information is in each experience, identify training and test workloads, and perform a search for the best prediction parameters. We describe the execution of each job using information that is available to workload managers that control the execution of jobs on parallel computers. The benefit of this approach is that it relies on information that is readily available from usage or accounting logs. The cost of this approach is that it does not include any application-specific information that could improve prediction accuracy. Table 1 shows the execution features of jobs submitted to the TACC lonestar system.

We evaluate our prediction technique using data from the TACC lonestar system. We create two workloads from the accounting logs: A training workload consisting of data from January and February of 2006 and an evaluation workload from March, 2006. Each workload consists of inserts of experiences into the experience base and requests for predictions. An insert is created each time a job completes execution and a prediction request is created each time a job is submitted to the workload manager. The training workload contains 16,892 insertions of experiences and the same number of prediction requests. The evaluation workload contains 10,361 insertions of experiences and again, the same number of prediction requests.

We optimized the predictor using the training workload and evaluated the performance of the resulting predictor by loading the training data into the experience base and predicting the jobs in the evaluation workload. The performance of our execution time prediction technique is shown in Table 2. This data shows that the prediction error on the evaluation data is nearly 72% of the mean execution time and is similar to the prediction error for the training data. The similarity of prediction error between the training and evaluation data indicates that the training data was a good representation of the evaluation data. Even though the prediction error is relatively high, it is much less than the nearly 246% error of the user-provided estimates. It may be possible to reduce the prediction error by using a larger set of training data, but it may also be the case that the execution times of jobs on lonestar are simply less predictable than those on other systems, which we will describe next.

Another way to measure the performance of our predictor is the amount of time it takes to form a prediction. We evaluate this using our execution time workload. We examine the amount of time it takes to predict our training workload of 16,892 prediction requests while we are searching for the best prediction parameters. On a 3 GHz Intel Xeon system, we find that the average prediction time varies between 3 and 8 milliseconds from experiment to experiment. We performed these experiments on a loaded computer system which we believe introduced the variability. In future work, we will more precisely quantify the amount of time it takes to form a prediction.

Table 1. The features in an execution experience for the lonestar system at TACC. This system is managed by the LSF workload manager.

Input Features		
Feature	**Type**	**Description**
User Name	String	The user that submitted the job.
Project Name	String	The project to charge to.
Queue Name	String	The queue the job was submitted to.
Number of CPUs	Integer	The number of CPUs requested.
Maximum Run Time	Integer	The user-specified maximum run time.
Time	Long	The prediction or insertion time.
Output Features		
Feature	**Type**	**Description**
Run Time	Integer	The amount of wall time used.

Table 2. Execution time prediction error on the TACC lonestar system.

	Our Predictions		User Estimates		
Workload	**Error (minutes)**	**% of Mean Run Time**	**Error (minutes)**	**% of Mean Run Time**	**Mean Run Time (minutes)**
Evaluation	122.44	71.59	420.23	245.70	171.04
Training	156.38	76.92	385.16	241.79	159.29

3.1. Related Work

There have been a number of previous efforts to predict the execution time of serial and parallel applications using statistical analysis, benchmarking, and modeling. Early work on predicting the execution times of serial applications was performed in [4] used both statistical analysis and modeling an application with a state machine. Kapadia [13] uses instance-based learning techniques to predict run times of serial application for a relatively small set of applications where information about the input data and configuration for each execution is known. Iverson [12] predicts the execution time of serial applications using nearest neighbors and kernel regressions. Predictions for the execution time of serial applications running on shared computer systems were investigated in [5] based on time series predictions of host load.

Several researchers, including the author of this work, have used a categorization approach to make predictions of parallel applications [6, 10, 19] where experiences are put into categories and all experiences in a category are used to form a prediction for a query that falls in that category. These categorization approaches turn out to be special cases of instance-based learning with a specific distance function.

An alternative approach to predicting execution times is to develop application and system models and then use these models to make predictions [18, 9]. Since this type of approach uses much more detailed information about applications and computer systems, it has the potential to be much more accurate. However, it is unlikely that performance models could be developed for all applications running on a general purpose distributed system such as the TeraGrid or Open Science Grid. Since our goal is to provide predictions for this type of distributed system, we did not choose the modeling approach.

In [19] it was shown that our previous categorization prediction technique has lower prediction error than the approaches of Downey and Gibbons, primarily due to our genetic algorithm searches to find the best prediction parameters. Table 4 shows a comparison of our previous categorization technique and our IBL technique. The four workloads that we use for comparison are described in Table 3. The results show that our IBL technique has lower error than our older categorization technique for 3 of the 4 workloads and has 10 percent lower prediction error on average. Furthermore, the prediction error for these workloads ranges from 32 to 58 percent of the mean run times, a significantly lower error than we obtained for the lonestar workload.

4. Predicting Queue Wait Times

We predict queue wait times using the same instance-based learning approach as when predicting execution

Table 3. Characteristics of comparison workloads.

Workload Name	System	Number of CPUs	Location	Time	Number of Jobs
ANL	IBM SP2	120	ANL	3 months of 1996	7,994
CTC	IBM SP2	512	CTC	11 months of 1996	79,302
SDSC95	Intel Paragon	400	SDSC	12 months of 1995	22,885
SDSC96	Intel Paragon	400	SDSC	12 months of 1996	22,337

Table 4. A comparison of our IBL execution time prediction technique to our previous categorization technique.

Workload	Error (minutes)			Mean Run Time (minutes)
	IBL	Categorization	User Estimate	
ANL (1996, 3 mo.)	41.27	38.48	104.35	97.08
CTC (1996)	105.71	106.73	222.71	182.49
SDSC (1995)	50.40	59.65	N/A	108.16
SDSC (1996)	53.25	74.56	N/A	166.85

times. The differences are in the features that make up each experience and the workloads. To predict queue wait times, we need to characterize the state of a workload manager and this is not straightforward. We begin by using the features specified in Table 5 that we selected based on our past experiences. These features fall in to several groups: information about the job, information about the running jobs, and information about the waiting jobs. A final input feature is the time the experience was observed (or the prediction is made) to capture any locality in time. The output feature is the wait time. The features are easily understood, except perhaps the three that have to do with work. We define work to be the number of CPUs multiplied by the amount of time the user estimates they will use. We have features that capture the amount of work for the job, the amount of work that running jobs have left to do, and the amount of work that waiting jobs will do. There are many other ways to characterize a wait time experience and we will explore them in future work.

We once again evaluate our prediction technique using data from the TACC lonestar system. We create a training workload from January and February of 2006 and an evaluation workload from March, 2006. An insert is created each time a job begins to execute and a predict is created each time a job is submitted. The training workload has 17,543 prediction requests and 15,469 insertions of experiences while the evaluation workload has 9,966 prediction requests and 8,817 insertions. We optimized our predictor using the same techniques we used to optimize our run time predictor. The performance of our wait time prediction technique is shown in Table 6. This data shows that the prediction error is 115 percent, relatively high, and somewhat higher than the 96 percent error on the training data. This difference in error may indicate that more training data is required to improve prediction accuracy and consistency.

4.1. Related Work

We previously used a categorization approach to predict queue wait times [20]. Other work predicts wait times using the same categorization technique as we used [15] and the same instance-based learning approach we present here [16]. A different approach is to provide an upper bound on wait times with some degree of confidence [3]. Comparison to this work is a little difficult because different information is being provided by each approach which makes it difficult to identify which approach has better performance.

An alternate approach that we have explored in the past is to perform simulations of the scheduling algorithm using run time predictions [20]. The advantage of this approach is that it can be more accurate than the one presented here. The disadvantages are that it requires detailed knowledge of the scheduling algorithm, which can be difficult to obtain, and that it is much more accurate when jobs submitted in the future have minimal affect on the start times of jobs that have already been submitted. This isn't always the case with fair share algorithms or user- or queue-based priorities.

5. Predicting File Transfers

Finally, we predict file transfer times using instance-based learning. In particular, we predict the amount of time it takes GridFTP [1] to transfer files. We select the features for file transfer experiences based on data from the

Table 5. The features in a wait time experience for the lonestar system at TACC.

Input Features		
Feature	**Type**	**Description**
Job Queue Name	String	The queue the job was submitted to.
Job # CPUs	Integer	The number of CPUs requested by the job.
Job Max. Run Time	Integer	The user-specified maximum run time.
Job Work	Integer	The amount of work the job will perform.
Running # Jobs	Integer	The number of jobs currently running.
Running Work	Integer	The remaining work of currently running jobs.
Waiting # Jobs	Integer	The number of waiting jobs.
Waiting Work	Integer	The amount of work to be done by jobs waiting to run.
Time	Long	The prediction or insertion time.
Output Features		
Feature	**Type**	**Description**
Wait Time	Integer	The amount of time the job waits to start.

Table 6. Wait time prediction error.

Workload	Error (minutes)	Percent of Mean Wait Time	Mean Wait Time (minutes)
Evaluation	365.83	115.27	317.38
Training	363.35	96.37	377.02

Grid FTP performance logs. After some experimentation, we currently use the features shown in Table 7. One thing to note is that we include a feature that is the domain of the source and destination hosts under the assumption that hosts in the same domain will have similar network performance to a remote system. In future work, we will further explore the best features to use including investigating if there are derived features that would be helpful (such as whether it is a work day and the time of day).

We evaluate our prediction technique using data from the TACC lonestar system. Our training workload is from October 22 to December 20 of 2006 and the evaluation workload is from December 21, 2006 to January 21, 2007. The training workload contains 7035 requests for predictions and the same number of insertions of experiences while the evaluation workload contains 3470 requests for predictions and the same number of insertions. For these workloads, an insert is created each time a transfer completes and a prediction request is created each time a transfer begins.

We optimized this predictor using the same techniques as we used to optimize our previous predictors. The performance of our execution time prediction technique is shown in Table 8. This data shows that our prediction error is 37 percent for the evaluation workload and this is lower than the 51 percent error of the training workload indicating that the training was effective.

5.1. Related Work

Predicting network performance is a relatively well studied field. One project that readers may be familiar with is the Network Weather Service [24]. This work differs in the prediction techniques used and that our approach requires no active probes of the network. Other work examined several different approaches to predict GridFTP performance [22] and reported significantly lower error than we attained for our lonestar workload. We examine this next to attempt to determine if their lower prediction error is a result of superior prediction techniques or different data sets.

The authors of [22] provided their data sets so that we could perform a comparison. The data set in this paper consisted of GridFTP transfers between the Information Sciences Institute (ISI) at The University of Southern California, the Lawrence Berkeley National Laboratory (LBL), and Argonne National Laboratory (ANL). The transfers were performed by the authors between ISI and ANL and between LBL and ANL with sizes ranging in size from 1 megabyte to 1 gigabyte. To improve prediction performance, the authors organized their data into 8 workloads. A transfer is assigned to a workload based on source and destination as well as size of the transfer. For each workload, the first 15 transfers were used as the training set and the rest of the transfers were the evaluation set.

Table 7. The features in a file transfer time experience for the lonestar system at TACC.

Input Features		
Feature	**Type**	**Description**
DestinationHost	String	The destination host of the file.
DestinationDomain	String	The destination domain of the file.
SourceHost	String	The source host of the file.
SourceDomain	String	The source domain of the file.
Size	Integer	The size of the file.
Streams	Integer	The number of parallel streams to use for the transfer.
Stripes	Integer	The number of stripes (servers) to use for the transfer.
Type	String	Whether the transfer is a store or a retrieve.
Time	Long	The prediction or insertion time.
Output Features		
Feature	**Type**	**Description**
Duration	Integer	The amount of time the transfer took.

Table 8. File transfer time prediction error.

Workload	Mean Error (seconds)	Percent of Mean Duration	Mean Duration (seconds)
Evaluation	8.50	36.57	32.29
Training	14.35	50.83	33.27

We choose to combine all of their data into a single training workload and a single evaluation workload since our approach benefits from having a larger training set. Our training workload contains the same 120 transfers as they used to train and the evaluation workload contains 762 transfers. The features in an experience for this workload are shown in Table 9 and we optimized our predictor using the same techniques as we used to optimize our previous predictors. The resulting performance is shown in Table 10.

The data shows that our mean prediction error is 16 percent of the mean bandwidth for the training workload and 13 percent of the mean bandwidth for the evaluation workload. This is significantly lower error than we attained for the GridFTP workload from lonestar. Since the same prediction technique was used, we attribute this difference to the data set. This data set is more structured with only 8 different transfer sizes and transfers from ANL to only 2 other sites. Our lonestar GridFTP data contains 390 different transfer sizes and transfers between lonestar and 76 other systems. There could also have been less contention on the networks used while taking this data, resulting in more consistent performance.

Table 10 also shows the percent error of the predictions, as was presented in [22]. We calculate percent error using

$$\frac{|bandwidth_m - bandwidth_p|}{bandwidth_m} * 100$$

where $bandwidth_m$ is the measured bandwidth and $bandwidth_p$ is the predicted bandwidth. Our approach resulted in a 18.38 percent error for the training data and a 14.61 percent error for the evaluation data. While [22] does not provide an overall percent error for each predictor, we calculate it from their downloadable raw performance data. Their predictor with the lowest error overall is AR5d that has a 15.79 percent error. Our approach has slightly lower prediction error so we can conclude that our approach is viable and that, once again, prediction performance is very dependent on the workload.

6. Prediction Services

We created several services to provide predictions and the architecture of these services is shown in Figure 2. These services are implemented with web services and have a Web Service Description Language (WSDL) interface definition and can be communicated with using the SOAP protocol. We currently host our services in a Globus version 4 [8] container and use the Globus tools to generate Java bindings for our WSDL service definitions.

The core service is the Learning Service that is very general and is simply a web service interface layered atop our instance-based learning prediction techniques. This service contains one or more predictors where each predictor has

Table 9. The features in a file transfer time experience for the data from [22].

Input Features		
Feature	**Type**	**Description**
Source	String	The source site of the file.
Destination	String	The destination site of the file.
Size	Integer	The size of the file.
Time	Long	The prediction or insertion time.
Output Features		
Feature	**Type**	**Description**
Bandwidth	Float	The bandwidth during the transfer took.

Table 10. File transfer time prediction error for data from [22].

Workload	Mean Error (KB/sec)	Percent of Mean Bandwidth	Mean Bandwidth (KB/sec)	Percent Error
Evaluation	822.03	13.33	6,166.22	14.61
Training	946.53	15.67	6,038.65	18.38

it's own experience schema, experience base, and prediction configuration. In this application, it has three predictors, one each for execution times, queue wait times, and GridFTP times. The interface to this service is relatively simple and allows a client to get a list of predictors, get information about each predictor, request a prediction, and insert an experience.

To keep the Learning Service general, we created a Scheduler Prediction Service and a GridFTP Prediction Service to perform functions specifically related to the types of predictions described in this paper. The GridFTP Prediction Service is relatively simple and simply passes prediction requests and new experiences through to the Learning Service.

It is also a relatively simple matter for the Scheduler Prediction Service to provide an execution time prediction. The client can either provide a description of a job or a job identifier. If a job identifier is provided, the service looks up the job description from the set of already submitted jobs. The job description is then passed to the Learning Service for a prediction to be formed. To insert an execution time experience, the Scheduler Prediction Service tracks the status of jobs using job state information sent to it by the Information Gatherer. When it notices that a job has finished, it inserts an experience for this job into the Learning Service.

To handle a wait time prediction request, the Scheduler Prediction Service receives a message from a client that contains information about a job to predict the wait time of. This information could describe a job that might be submitted or be the job identifier of a job that was already submitted. The Scheduler Prediction Service then combines job information with a description of the jobs that are running and waiting on the system (provided by the Information Gatherer) to form a query to pass to the Learning Service. The Scheduler Prediction Service also stores information about the conditions when every job is submitted to the system. When each job begins to execute, this information is used to form a wait time experience that is inserted into the Learning Service so that more information is available for wait time predictions.

The final service in our architecture is the Information Gatherer that gathers information from the local workload manager and GridFTP server. There are a number of ways that this service can get the information it needs. In the current design, it periodically executes a command provided by the workload manager to get the state of jobs in the system. This state is then passed on to the Scheduler Prediction Service to be used when providing start time predictions and when creating execution time and wait time experiences. The Information Gatherer simply watches the GridFTP performance log files to obtain file transfer experiences to send to the GridFTP Prediction Service.

6.1. Client Interfaces

The prediction service has a WSDL interface definition so it is straightforward to generate client side APIs in a number of programming languages. We are primarily working with Java and so are currently only generating Java APIs.

We have created prototype command line clients that use the generated Java client API. Two possible approaches are to have command line programs that are named generally or that are named to resemble the commands provided by the

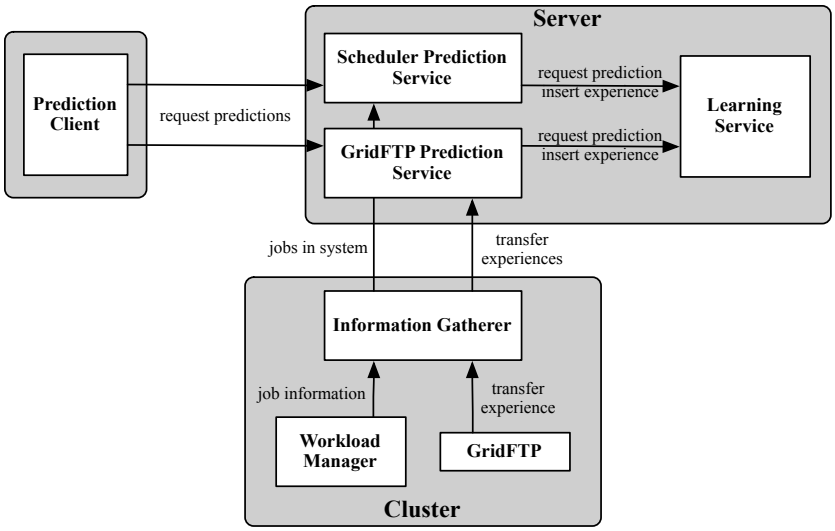

Figure 2. Architecture of our prototype prediction services.

local workload manager. Our target audience for the command line programs is users of a distributed system with many different clusters. Since there are a number of different workload managers available for clusters, we chose to name our command line programs in a general way. These programs are:

- `pred_runtime` is used to request a run time prediction. A system and job identifier can be supplied for an already submitted job or a description of a yet to be submitted job can be provided on the command line or in a file.

- `pred_waittime` is used to request a wait time prediction. Either a system and job identifier can be supplied or a description of a job.

- `pred_transtime` is used to request a transfer time prediction. The default and only protocol supported currently is GridFTP. Either a list of local files can be provided along with a destination host or a description of the files to be transfered can be provided along with the source and destination hosts.

7. Conclusions and Future Work

This paper presents our instance-based learning techniques to predict job execution times, batch scheduling queue wait times, and file transfer times. Our results show that the prediction errors for data obtained from the TACC lonestar system are 37% of mean transfer time for file transfer time, 115% of mean queue wait time, and 72% of mean execution time. We also found that our approach has lower prediction error on other workloads and has lower prediction error than previous approaches. In future work, we plan to examine what features should be used for wait time experiences and GridFTP experiences, as there are a number of derived features to explore. Finally, we wish to examine the performance of our prediction techniques on data gathered from other clusters in the distributed systems TACC participates in.

We also provide a description of the prototype prediction services that we have created. In the future, we will be gaining more experience managing these deployed services. We expect to refine our service interfaces and command line interfaces as well as provide one or more portlets so that our predictions are easy to incorporate into user portals and science gateways. If an information service become widely deployed, we will investigate gathering the data we need from that service rather than having our own information gathering daemon. We will also be investigating whether and how our services should automatically optimize their prediction parameters.

References

[1] William Allcock, John Bresnahan, Rajkumar Kettimuthu, Michael Link, Catalin Dumitrescu, Ioan Raicu, and Ian Foster. The Globus Striped GridFTP

Framework and Server. In *Proceedings of the SC05 Conference*, November 2005.

[2] Christopher Atkeson, Andrew Moore, and Stefan Schaal. Locally Weighted Learning. *Artificial Intelligence Review*, 11:11–73, 1997.

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

[4] Murthy Devarakonda and Ravishankar Iyer. Predictability of Process Resource Usage: A Measurement-Based Study on UNIX. *IEEE Transactions on Software Engineering*, 15(12):1579–1586, December 1989.

[5] Peter Dinda. Online Prediction of the Running Time of Tasks. In *Proceedings of the 10th IEEE International Symposium on High Performance Distributed Computing*, 2001.

[6] Allen Downey. Predicting Queue Times on Space-Sharing Parallel Computers. In *International Parallel Processing Symposium*, 1997.

[7] Enabling Grids for E-sciencE. http://public.eu-egee.org.

[8] Ian Foster. Globus Toolkit Version 4: Software for Service-Oriented Systems. 3779:2–13, 2005.

[9] Jonathan Geisler, Valerie Taylor, Xingfu Wu, and Rick Stevens. Using kernel coupling to improve the performance of multithreaded applications. In *Proceedings of the 16th International Conference on Parallel and Distributed Computing Systems*, 2003.

[10] Richard Gibbons. A Historical Application Profiler for Use by Parallel Scheculers. *Lecture Notes on Computer Science*, 1297:58–75, 1997.

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

[12] M. Iverson, F. Ozguner, and L. Potter. Statistical Prediction of Task Execution Times Through Analytic Benchmarking for Scheduling in a Heterogeneous Environment. In *Proceedings of the IPPS/SPDP'99 Heterogeneous Computing Workshop*, 1999.

[13] N. Kapadia, J. Fortes, and C. Brodley. Predictive Application Performance Modeling in a Computational Grid Environment. In *Proceedings of the 8th High Performance Distributed Computing Conference*, 1999.

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

[15] Hui Li, David Groep, Jeff Templon, and Lex 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.

[16] Hui Li, David Groep, and Lex 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.

[17] The Open Science Grid. http://www.opensciencegrid.org.

[18] Jennifer Schopf and Francine Berman. Performance Prediction in Production Environments. In *14th International Parallel Processing Symposium and the 9th Symposium on Parallel and Distributed Processing*, 1998.

[19] Warren Smith, Ian Foster, and Valerie Taylor. Predicting Application Run Times Using Historical Information. *Lecture Notes on Computer Science*, 1459:122–142, 1998.

[20] Warren Smith, Valerie Taylor, and Ian 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.

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

[22] Sudharshan Vazhkudai, Jennifer M. Schopf, and Ian Foster. Predicting the Performance of Wide Area Data Transfers. In *Proceedings of the 2002 International Parallel and Distributed Processing Symposium*, May 2002.

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

[24] Richard Wolski. Dynamically Forecasting Network Performance Using the Network Weather Service. *Journal of Cluster Computing*, 1:119–132, January 1998.