

Replica Selection Using Instance-Based Learning

Yu Hu

Department of Computer Science
University of Chicago
yuhu@cs.uchicago.edu

Jennifer M. Schopf

Mathematics and Computer Science
Division
Argonne National Laboratory
jms@mcs.anl.gov

Abstract

In many scientific applications, Grid technologies and infrastructures facilitate distributed resource sharing and coordination in dynamic, heterogeneous multi-institutional environments. Replication of data can enable scalable resource storage for Grid applications that involve large datasets. The selection of a replica can, however, significantly influence the efficiency of a replication scheme. Many current replica selection approaches assume that a significant amount of environment data is available, including network status information, log files of historical GridFTP file transfers, CPU status, and predictions. We propose a lightweight instance-based learning (IBL) algorithm to allow efficient replica selection when only a limited amount of environment data is available. We evaluate this approach in a Grid environment and demonstrate that using IBL to select replicas has reasonable accuracy with acceptable overheads when only limited data sources are available.

Keywords: Data Grid, Replica selection, Instance-based learning

1 Introduction

In many scientific applications, Grid technologies and infrastructures facilitate distributed resource sharing and coordination in dynamic, heterogeneous multi-institutional environments. In an increasing number of applications, large data collections are emerging as important community resources [CFK+01]. By creating copies, called *replicas*, of subsets of large datasets that are then copied to other sites, replication can enable better load balancing and can improve reliability [RF01]. The selection of a replica can, however, significantly influence the efficiency of a replication scheme and may not be straightforward in a Grid environment where components,

including networks, CPU, and disks, behave unpredictably. Current approaches to replica selection [FSW+99, SW02, VS03] can require large amounts of environment data, including network status information, transfer log files, and I/O disk throughput logging. When less environment data is available, these approaches lose their effectiveness [VS03].

In this paper we present a replica selection approach based on instance-based learning (IBL). IBL allows us to use only a limited amount of environment data to make replica decisions with modest overheads. We evaluate the IBL approach to replica selection in a real Grid environment and compare it to using a straightforward average of previous transfers, the method currently used in some systems. Experimental results show that the IBL prediction algorithm achieves a 5% to 10% improvement over the average approach with an overhead of only 1 to 2 seconds. These results demonstrate that the IBL replica selection algorithm can be an efficient approach to replica selection when only limited data sources are available.

The rest of the paper is organized as follows. In Section 2, we discuss related work and motivate our approach. In Section 3, we describe IBL algorithms and the decision points when using IBL for replica selection. In Section 4, we present our experiments. In Section 5, we briefly summarize our results.

2 Related Work

Two main approaches to selecting a replica exist: looking at individual components to construct a model and measuring end-to-end performance to make a prediction. For example, Shen and Choudhary [SC00] use individual components for network and disk access to predict I/O behavior. However, Geisler and Taylor [GT99] have shown the importance of understanding how components interact; in particular, since couplings between components can change behavior, considering components in isolation may miss significant performance effects.

Whole-system monitoring is a more common approach. Examples include the Network Weather Service [WSH99] and Iperf [HK98], each of which predicts network behavior from historical information for the end-to-end system by gathering data using probes of varying sizes. However, we have seen that probe behavior may not be strongly correlated to the achieved bandwidth for large file transfers [VS03]. Several statistical approaches to predicting file transfer times have also been evaluated: Faerman et al. [FSW+99] and Vazhkudai and Schopf [VS03] use regression models, and Swamy et al. [SW02] construct cumulative distribution functions to derive predictions of transfer times. While each of these achieves relatively good accuracy, the algorithms assume that a significant amount of environment data is available, including bandwidth logs, network variation information, file transfer logs, and disk throughput logs.

Instance-based learning (IBL) is widely used in many fields to make predictions, including high performance computing. Gibbons [G97], Smith et al. [SFT98], and Kapadia et al. [KFB99] all use techniques similar to IBL to predict application execution times in a cluster or Grid environment. Kroeger and Long [KL96] use an IBL technique to predict file systems events.

Our approach differs from previous work in two ways. First, while we use whole-system measurement as a basis, we assume that we have access only to file transfer logs and not more comprehensive environment data. Second, we use IBL to make a relative prediction, not an absolute one and predict only which server will be fastest, not its throughput value.

3 Instance-Based Learning

3.1 Definitions

To describe our IBL algorithm, we introduce three concepts: an instance, a distance metric, and a target function.

An *instance* is a set of data about an event, often represented as a vector of values. In our work, an event refers to a file transfer, and an instance of this event consists of the six pieces of data we

collect for each transfer: day and hour that the transfer occurs, file size, source, destination, and transfer time.

We use this set of data in our work because it was the common set available from several file transfer mechanisms, and likely to be the minimal set we thought we could expect to have available in all cases. In our experiments this data is made available through the simple logging service that is part of the GridFTP server.

A *distance metric* is a function to define the similarity between two instances, usually a function that operates on the vectors of the instance. A distance metric is used to select a set of similar previous instances to the current instance so that further (more expensive) evaluation can be done on the smaller set, called a *neighbor set*. Good performance of any IBL approach strongly depends both on a well-defined distance metric that can identify the right neighbor set and on a smart determination of how many similar instances to be kept for further evaluation.

In our approach we use a weighted Euclidean distance metric, one of the most common distance metrics, because it allows us to weight each element of the vector separately. For example, in our experiments we found that day and time of the transfer and file size played a larger role than, say, who was transferring the file or the name of the file (both of which significantly affected the predictions of run times [SFT98]).

Thus, the distance between two instances I_1 and I_2 is defined as

$$D(I_1, I_2) = \sqrt{(I_1f - I_2f)^2 * W_f + (I_1h - I_2h)^2 * W_h + (I_1d - I_2d)^2 * W_d}$$

where

$$W_f + W_d + W_h = 1$$

and

I_jf , I_jd , and I_jh are the attribute filesize, day, and hour of instance I_j , respectively, and

W_f , W_d , and W_h are weights of for filesize, day, and hour, respectively.

An instance includes three additional elements not explicitly used in the distance metric. The file destination is set *a priori*; we know where we want the data to go. The neighbor set is selected in order to determine which instances are similar and eventually which source should be used, so when the neighbor set is selected, and the distance function is defined, we do not know the source or the transfer time, therefore we do not use these elements in the distance metric.

A *target function* is a function that, given a set of neighbor instances selected using the distance metric, performs some additional analysis on the smaller set in order to make a prediction for the current instance. For example, given a set of similar file transfers, such as those listed in Table 1, we can determine which server to fetch the next replica from by picking the server that previously achieved the best transfer time for some previous transfer (Server A in our example) or by selecting the server that is used in the majority of the transfers in the neighbor set (Server B in our example). We call these approaches Top and Major, respectively.

Table 1. Sample transfer file for target function comparison.

Instance	Transfer Time (MB/s)	Replica Server
I_1	1.1	B
I_2	1.2	A
I_3	0.9	B

3.2 Algorithm

The IBL algorithm we use for replica selection consists of the following steps:

1. A set of past instances (previous file transfers) is stored in a history database.
2. A set of similar instances, the neighbor set, is selected. When a decision needs to be made about which server to retrieve the next replica from, the history database is examined, and the weight function is used to select those previous file transfers that are most similar to the one about to be made, using the weighted Euclidean distance metric, in terms of hour, day, and size of transfer.
3. A target function, either Top or Major, is applied to the neighbor set to decide what server should be used for the upcoming file transfer.

In order to use this algorithm, we need to define how many instances should be contained in the neighbor set, and the weightings used in the distance metric to select those neighbors. The pragmatic approach we take to this is described in the next section.

4 Experiments and Results

In this section we discuss the experiments and results for the IBL replica selection algorithm on our Grid testbed.

4.1 Experimental Setup

To be able to determine the best site from which we should transfer a file, we had two obvious choices for running experiments. The first choice was to run the experiments in a simulator where we had complete control and could observe the differences in the decisions our algorithm made versus those that an omnipotent selector would make. However, we felt strongly that simulated results would not reflect what a user would see in a real Grid environment. The second choice was to run on a real system, sharing the network and hosts with other users, and to determine which file was the right one to transfer a replica from by making a selection decision but performing the actual file transfers from all the server sites in order to see which one was the best in fact. This approach suffers severe contention on the receiving machine and I/O system, to the point where data is unavoidably skewed and results are nondeterministic.

We decided to try a third approach, namely, sending the files *from* the receiver machine *to* the server machines. This allowed us to run the experiments on a real system and yet avoid the

contention effects seen when receiving on a single host. Because our approach predicts only relative behavior (which machine is best) as opposed to an absolute prediction of file transfer time, we verified that the transfer times in the reverse direction behaved with the same relative performance as those in the original direction [HS04].

Our experiment platform includes one client and five remote servers, as shown in Table 2. We used GridFTP [ABB+02] for our file transfers and logging. Each site ran Linux and had version 2.4.3 of the Globus Toolkit® installed. We used eleven files as test files, ranging in size from 1 MB to 1024 MB in steps of order 2. We collected three log files (LearningSet, Data1, and Data2), each consisting of 10 days of file transfer data, approximately 110 file transfers, on this testbed. Each transfer sent a randomly selected test file from the client site (University of Chicago) to the remote five server sites. The time interval between file transfers was set as a random value between 1 and 10,800 seconds. For each transfer, we logged the day, hour, file size, source, destination, and transfer time.

Table 2. Experimental testbed.

Site	CPU	Architecture	Memory
University of Chicago	AMD 266 MHz	i686	128 MB
University of Illinois, Urbana Champaign	Intel 266 MHz	i686	256 MB
University of California, San Diego	AMD 1.7 GHz	i686	1GB
University of Wisconsin	Intel 900 MHz	ia64	1GB
University of Houston	Intel 900 MHz	ia64	1GB
University of Tennessee, Knoxville	Intel 1.7 GHz	i686	1GB

We performed our experiments on these log files post mortem. We read in the time, day, and file size from the log file for the next transfer; determined the best “source” for the data; and then added that transfer data to the database of transfer instances to use in making the next decision. The first transfer used a random selection of sources.

In real life, the only data available to an algorithm would be the single file transfer logs. However, we also wanted to study the affects of having omnipotent data about all sources in our work to better understand how having additional data affected the predictive ability of our algorithm. Towards this end, we evaluated two database update strategies: adding in just *One* file transfer data instance (chosen from the five available at random), and adding in *All* five instances we collected at one time.

In total, we have four variants of the IBL algorithm to consider in our experiments, each with a choice of Target Function (Section 3) and database update strategy: Update the database with all data and use the Top target function (AllTop), update the database with all data and use the Major target function (AllMaj), update the database with one data point and use the Top target function (OneTop), and update the database with one data point and use the Major target function (OneMaj), keeping in mind that only the last two are options in real settings.

4.2 Determination of Parameters

To use an IBL approach, we must define the size of the neighbor set selected by the distance metric and any weights used as part of the distance metric. To determine a good set of parameters to use, we must resolve a chicken-and-egg problem: in order to determine the best-sized neighbor set, we need the complete weight function parameterization; and in order to determine the weight function parameterization, we need a neighbor set size. In our work, we determined the right neighbor set size by fixing a set of weight parameterizations that spanned the space for our metric. Next, we examined all feasible values for the neighbor set size. Then, given the best choice for neighbor set size, we evaluated different choices of weight parameters for the distance metric.

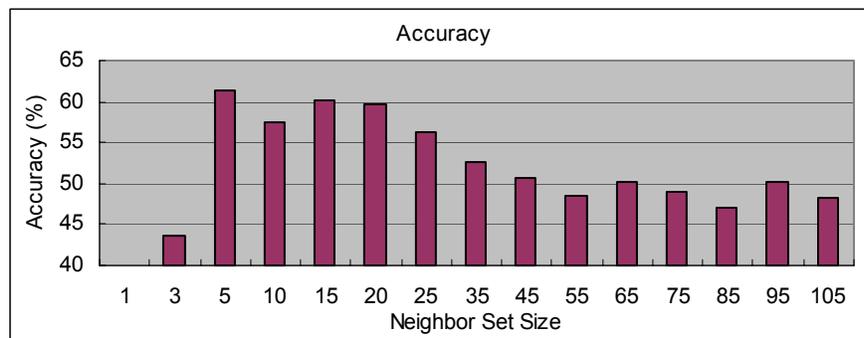
In this section we detail how these parameters were selected. All parameter selection was performed using the LearningSet log file. As part of the conclusions we evaluate how the accuracy of these parameters affect the outcome of the prediction technique for the Data1 and

Data2 log files. The approach we take is a general search of the space, a *priori*, as opposed to, say, using a genetic algorithm during the runs. We made this decision based on overhead concerns: the IBL approach is an expensive one, and we needed to reduce runtime costs in order to make it practical to use during runtime.

We use two metrics when we evaluate parameter selection: accuracy and overhead. *Accuracy* is defined as the percentage of time the fastest transfer source is selected. *Overhead* is the time, in seconds, it takes the algorithm to run.

4.2.1 The Neighbor Set Size Determination

To determine a good value to use for the neighbor set size, we selected four different weight parameterizations for the distance metric that cover the space possible, namely, $\langle 0.33, 0.33, 0.33 \rangle$, $\langle 0.8, 0.1, 0.1 \rangle$, $\langle 0.1, 0.8, 0.1 \rangle$, and $\langle 0.1, 0.1, 0.8 \rangle$ for the tuple $\langle \text{fileSize}, \text{day}, \text{hour} \rangle$. Note that these are not the weights that will be used in the final determination of the distance metric, but a simple spanning set to resolve the aforementioned chicken-and-egg problem. We then ran the IBL prediction approach using the LearningSet data to examine the accuracy and overhead to determine a good value for the neighbor set size for all four IBL variations: AllTop, AllMaj, OneTop, and OneMaj.



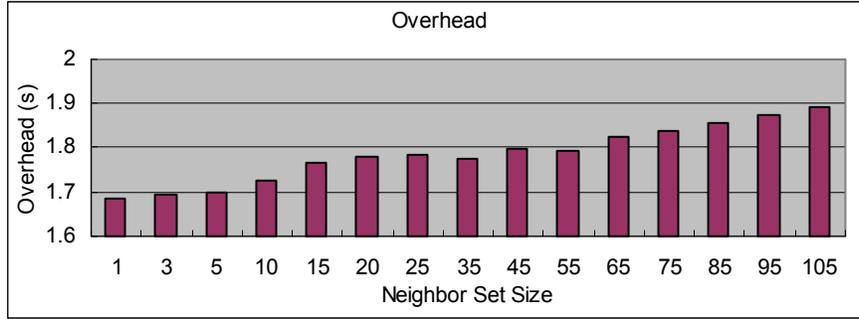


Figure 1. Neighbor set size evaluation for IBL variant AllTop.

Figure 1 shows the accuracy and overhead for the AllTop IBL variant the results for the other 3 variants were similar. The accuracy plot shows a smooth, mostly Gaussian curve. By increasing the number of neighbors, more useful historical information is included; but when too many neighbors are added, the similarity decreases and the accuracy is reduced, in this case around 15. The overhead of the algorithm in general grows linearly with the size of the neighbor set. The best values for the neighbor set size are given in Table 3 for the four IBL variants.

So we have now determined the size of the neighbor sets to use in step 2 of our algorithm (Section 3.2). We now need to examine the distance metrics and determine its weights in order to select those neighbors most qualified.

Table 3. Neighbor set size for four IBL variants.

IBL Variant	Best Neighbor Set Size
AllTop	15
AllMaj	20
OneTop	35

OneMaj	35
--------	----

4.2.2 Weighted Euclidean Distance Parameterization

Given a good value of for the neighbor set size, we can now determine what the weights of the weighted Euclidean distance function should be. We have three weights to determine, one each for the fileSize, day, and hour parameters.

When weighting the parameters used in the distance function we also have the problem of interdependencies. A distance function includes a parameter for each of the three aspects, and there are different local minima for each. Determining these weights is the most difficult aspect of the IBL approach. We determine values in isolation, and then examine several ways to combine them, finally selecting the weightings that give the best accuracy for our learning set of data. We use the learning set for this selection, but then apply those values to our other data sets in part to show that “close enough” values perform well, in fact, Table 6 shows that in several instances we see better predictions using the parameterization on a different data set than the original.

To evaluate the parameters, we first examine each weight in isolation for values of 0.1 to 0.9 in steps of 0.1, and set the weights of the other two parameters equal (the sum of the three weights was 1). This was done for all four IBL variants, and the best choices for each parameter in isolation are given in Table 4 (FileSize-Centric for file size in isolation, etc.). Note that occasionally two values for the weight had the same accuracy, in which case we use the average (for example, AllTop and Day-Centric).

Table 4. Best parameterization for the distance function for weights determined in isolation, for tuple $\langle W_f, W_d, W_h \rangle$.

IBL Variant	FileSize-Centric	Day-Centric	Hour-Centric
-------------	------------------	-------------	--------------

AllTop	<0.9, 0.05, 0.05>	<0.25, 0.375, 0.375>	<0.55, 0.225, 0.225>
AllMaj	<0.5, 0.25, 0.25>	<0.4, 0.3, 0.3>	<0.2, 0.4, 0.4>
OneTop	<0.9, 0.05, 0.05>	<0.3, 0.35, 0.35>	<0.4, 0.3, 0.3>
OneMaj	<0.8, 0.1, 0.1>	<0.4, 0.3, 0.3>	<0.5, 0.25, 0.25>

The next problem to solve is how to combine these choices in a sensible way. We examined four approaches. (1) Normalize the three weights proportional to their good values, called the *Ratio Set*. (2–4) Assign the weight of one parameter the best value in isolation, determined using the accuracy of selection with the learning set; then assign the other two parameters proportional to their values in isolation, called FSSet, DaySet, and HourSet, respectively. So for AllMaj and FSSet, the parameter W_f is set to 0.5, the best file size weight in isolation, and W_d and W_h are set according to their isolation weight ratio (0.4:0.2), so that the sum of the weights is one, and the full tuple is <0.5, 0.33, 0.17>. We then selected the parameter setting approach that was best for each of our four experiment cases using the LearningSet log file. These values are detailed in Table 5.

We have therefore determined a number of possible weightings for the distance metric. We then evaluated the accuracy of the selections offered by each of these weightings, using the learning set data, and the highest accuracy values are highlighted in Table 5.

Having selected good parameters using only the LearningSet log file, we used these values for the other datasets collected, Data1 and Data2. An evaluation of the effects of these parameters on the overall prediction accuracy is given at the end of Section 4.3.

Table 5. Candidate sets (W_f, W_d, W_h) for weighted Euclidian distance function for four IBL variants. Best candidate set is shaded gray.

IBL Variant	Candidate FSSet	Candidate DaySet	Candidate HourSet	Candidate Ratio Set
AllTop	<0.9, 0.03, 0.07>	<0.47, 0.25, 0.28>	<0.35, 0.1, 0.55>	<0.53, 0.15, 0.32>
AllMaj	<0.5, 0.33, 0.17>	<0.43, 0.4, 0.17>	<0.44, 0.36, 0.2>	<0.45, 0.36, 0.19>
OneTop	<0.9, 0.04, 0.06>	<0.48, 0.3, 0.22>	<0.45, 0.15, 0.4>	<0.56, 0.19, 0.25>
OneMaj	<0.8, 0.09, 0.11>	<0.37, 0.4, 0.23>	<0.33, 0.17, 0.5>	<0.47, 0.24, 0.29>

4.3 Comparison with the Average Approach

We ran a series of experiments in which we used the IBL approach to make replica selection decisions. These results were then compared to a simple averaging approach in terms of predictive ability and overhead. The averaging approach we used simply computed the throughput (total transfer time/number of bytes transferred) for each server for all previous transfers, and selected the server with the highest average throughput to transfer the next file from as is done in current approaches. This gave us six strategies to compare: the four IBL variants plus two using the averaging technique, AllAvg and OneAvg. Figures 2 and 3 show the accuracy of these six approaches. Summary statistics for overall accuracy are given in Table 6, and for overhead are given in Table 7.

Table 6. Summary statistics for overall accuracy.

IBL Variant	LearningSet	Data1	Data2
AllTop	65.25	70.0	65.45
AllMaj	67.27	69.06	61.82
AllAvg	54.77	58.18	50.09
OneTop	52.74	50.91	58.18
OneMaj	50.0	60.0	59.09
OneAvg	39.0	44.55	50.91

Table 7. Overhead for single replica selection.

Approach	Data1 Overhead (s)	Data2 Overhead (s)
AllTop	1.44	1.69
AllMaj	1.44	1.75
AllAvg	0.09	0.11
OneTop	0.1	0.14
OneMaj	0.12	0.11
OneAvg	0.09	0.09

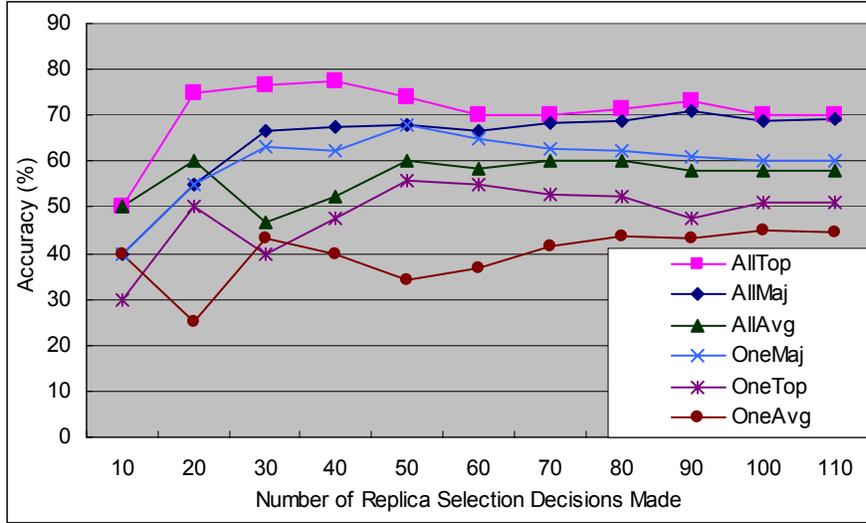


Figure 2. Prediction accuracy for four IBL variants and two average prediction techniques on Data1 log.

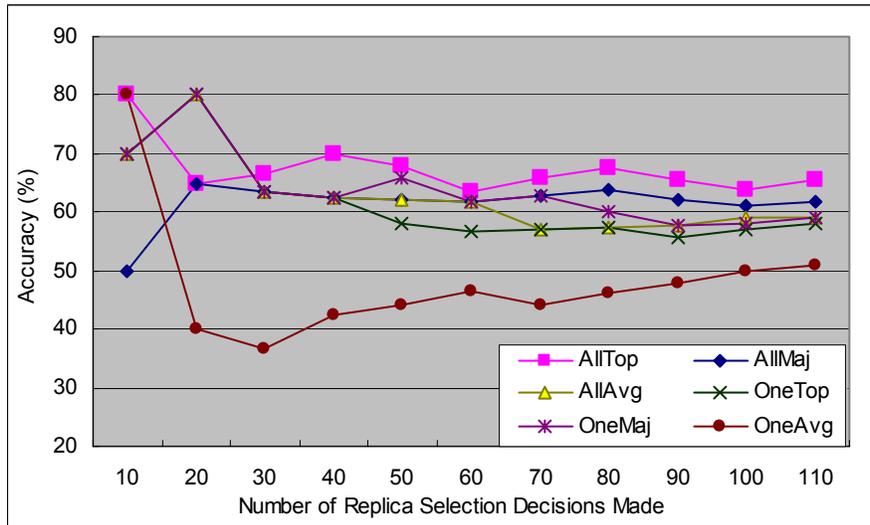


Figure 3. Prediction accuracy for four IBL variants and two average prediction techniques on Data2 log.

Figures 2 and 3 show that after an initial start-up phase (less than the first 30 decisions), the accuracy remains relatively constant for each approach. This is expected because decisions cannot be made without some data in the database. We also observe that a higher degree of accuracy is achieved by the approaches that add in all of the data transfer information instead of

only the one selected. Again, this is not surprising because there is more relevant data to work from. However, we note that in practical settings this additional data will not be available.

More important, we found that for a given update approach, both target functions (Top and Major) had a higher accuracy than the average approach, by approximately 10% in all cases. When we compare the summary statistics given in Table 6, we see that different IBL variants behave best for different datasets; hence, no conclusion can be made as to which is better for our setting. The overall accuracy of each of the IBL variants is at best 70%. This is comparable to Vazhkudai’s 15% to 30% error [VS03] when making absolute predictions and using multiple datasets in a similar setting. The IBL approach uses only limited environment data, and therefore improves on this approach. In addition, given the improvement of the IBL approach over the more common average approach, we feel that the IBL approach should be strongly considered.

As expected, collecting additional data with the All approach significantly improved the server selection accuracy. Because of this, if this approach were used in practice we would recommend an evaluation of additional “dummy” transfers to collect more environment data for use in the algorithm.

Looking at the overhead information described in Table 7, we see that the overhead of the average approach is smaller than that of the IBL approach. This was expected. However, the overhead of the IBL approach is only 0.1–1.7 seconds, which is quite acceptable in an environment where the transfers are large. For example, in our testbed the file transfer time for a 1 GB file averaged more than 260 seconds.

We include summary statistics for the learning set in Table 6 to help evaluate the need for weight selection based on a specific dataset instance. These results show that the IBL algorithms are not overly dependent on the exact selection of weights used. The weights were picked using only data from the LearningSet log, and yet the other data logs saw better predictions in several cases. Therefore, we believe approximate weights are more than adequate for good predictions.

5 Conclusion

In this paper, we described a lightweight instance-based learning approach to replica selection to be used when only limited data sources are available. We applied our algorithm using data collected from a Grid environment, and found that our approach achieved more accurate results for only slightly more overhead than the currently used averaging approach. Furthermore, approximate sets of weights were acceptable and did not adversely impact the accuracy.

Acknowledgments

We gratefully acknowledge support and assistance from Christopher Beckmann, Yong Zhao, and Xuehai Zhang. Thanks also to our colleagues within the GrADS project for providing access to testbed resources. This work was supported in part by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Advanced Scientific Computing Research, Office of Science, U.S. Department of Energy, under contract W-31-109-Eng-38.

References

- [ABB+02] Bill Allcock, Joe Bester, John Bresnahan, Ann L. Chervenak, Ian Foster, Carl Kesselman, Samuel Meder, Veronika Nefedova, Darcy Quesnel, and Steve Tuecke. Secure, Efficient Data Transport and Replica Management for High-Performance Data-Intensive Computing. In *Proceedings of the Eighteenth IEEE Symposium on Mass Storage Systems and Technologies*, page 13, 2001.
- [CFK+01] Ann Chervenak, Ian Foster, Carl Kesselman, Charles Salisbury, and Steven Tuecke. The Data Grid: Towards an Architecture for the Distributed Management and Analysis of Large Scientific Datasets. *Journal of Network and Computer Applications*, vol. 23, 187–200, 2001.
- [FSW+99] Marcio Faerman, Alan Su, Rich Wolski, and Fran Berman. Adaptive Performance Prediction for Distributed Data-Intensive Applications. In *Proceedings of SuperComputing '99*, 1999.
- [GT99] Jonathan Geisler and Valerie Taylor. Performance Coupling: Case Studies for Measuring the Interactions of Kernels in Modern Applications. In *Proceedings of the SPEC Workshop on Performance Evaluation with Realistic Applications*, 1999.

- [HK98] Chung-Hsing Hsu and Ulrich Kremer. IPERF: A Framework for Automatic Construction of Performance Prediction Models. In *Proceedings of the Workshop on Profile and Feedback-Directed Compilation (PFDC)*, 1998.
- [HS04] Yu Hu and Jennifer M. Schopf, IBL for Replica Selection in Data-Intensive Grid Applications, University of Chicago Computer Science Department Technical Report #TR-2004-03, April 2004. Available from <https://www.cs.uchicago.edu/research/publications/techreports/TR-2004-03>
- [G97] Richard Gibbons. A Historical Application Profiler for Use by Parallel Schedulers. In *Proceedings of the 3rd Workshop on Job Scheduling Strategies for Parallel Processing*, 1997.
- [KFB99] Nirav H. Kapadia, José A.B. Fortes, Carla E. Brodley. Predictive Application-Performance Modeling in a Computational Grid Environment. In *Proceedings of the Symposium on High-Performance Distributed Computing (HPDC)*, 1999.
- [KL96] Thomas M. Kroeger and Darrell D.E. Long. Predicting File System Actions from Prior Events. In *Proceedings of USENIX 1996 Annual Technical Conference*, 1996.
- [RF01] Kavitha Ranganathan and Ian Foster. Identifying Dynamic Replication Strategies for a High-Performance Data Grid. In *Proceedings of the International Grid Computing Workshop (Grid'01)*, 2001.
- [SC00] X. Shen and A. Choudhary. A Multi-Storage Resource Architecture and I/O, Performance Prediction for Scientific Computing. In *Proceedings of the Symposium on High-Performance Distributed Computing (HPDC)*, 2000.
- [SFT98] Warren Smith, Ian Foster, and Valerie Taylor. Predicting Application Run Times Using Historical Information. In *Proceedings of IPPS/SPDP '98 Workshop on Job Scheduling Strategies for Parallel Processing*, 1998.
- [SW02] Martin Swany and Rich Wolski, Multivariate Resource Performance Forecasting in Network Weather Service. In *Proceedings of SuperComputing '02*, 2002.
- [WSH99] Rich Wolski, Neil T. Spring, and Jim Hayes. The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing. *Journal of Future Generation Computer Systems*, vol. 15, 757–768, October 1999.
- [VS03] Sudharshan Vazhkudai and Jennifer M. Schopf. Using Regression Techniques to Predict Large Data Transfers. *International Journal of High Performance Computing Applications* (special issue on Grid Computing: Infrastructure and Applications), vol. 17, no. 3, August 2003.