IBL for Replica Selection in Data-Intensive Grid Applications

Yu Hu[1] Jennifer M Schopf[2]
Argonne National Laboratory[2]
Department of Computer Science, University of Chicago[1]
{yuhu}@cs.uchicago.edu[1] {jms}@mcs.anl.gov[2]

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 help enable high-throughput file transfer and scalable resource storage in scientific Grid applications that involve large data transfers. The selection of a replica can, however, significantly influence the efficiency of a replication scheme. Many current approaches assume that a significant amount of data is available, such as network status information, log files of historical GridFTP file transfers, and CPU status and predictions. We propose a lightweight instance-based learning (IBL) algorithm to allow efficient replica selection with much less required data. We implement the approach and evaluate it in a Grid environment. Our evaluation demonstrates that the IBL approach can be an efficient tool for replica selection when only limited data sources are available.

Keywords: Data Grid, Replica Selection, IBL, and GridFTP.

1 Introduction

Grid computing, the integration of a collection of distributed computing resources to offer performance unattainable by any single machine, has experienced a surprisingly fast evolution in many fields during the past several years. In an increasing number of data-intensive Grid applications, large data collections are emerging as important community resources [1]. For example, current projects focusing on data Grid [1] technology including GriPhyn [2], the Particle Physics Data Grid [3], the EU Data Grid [4], and the Taiwan Knowledge Innovation Grid [5] involve transfers of data in terabytes or even petabytes.

By creating multiple copies (replicas) of the huge datasets, replication in these data-intensive Grid applications can enable high-throughput file transfers. Moreover, it can help in load balancing and can improve reliability [6].

The selection of a replica is complicated, however. It involves using some application-specific criteria [7] to choose from among various replicas spread across the Grid on numerous devices whose performance could vary in unpredictable ways.

Current approaches [8][9][10][11][12] proposed for replica selection need large amounts of data (e.g., network status information, GridFTP logs, and I/O disk throughput). When less information is available—for instance, only GridFTP log files—such approaches may not apply. This paper addresses the problem of replica selection in these circumstances.

Our approach is based on instance-based learning (IBL), an Artificial Intelligence technique that makes a prediction based only on historic information. IBL is widely used for prediction in many applications [13][14][15][16]. This versatility, coupled with its modest information requirement, makes it well suited for our replica selection algorithm.

In this paper, we present a new replica selection approach, called IBL Replica Selection. Our approach has two novel aspects. First, it performs replica selection based only on historical GridFTP log files; thus it requires much less information than do other approaches. Second, in our algorithm, the replica site is picked using a *relative* metric. That is, we select a replica without determining the absolute values of transfer times. Hence, our IBL algorithm is more lightweight than most current methods.

We have implemented and evaluated the IBL approach in a real Grid environment. Specifically, we conducted some search experiments to find the proper parameters for our IBL algorithm. With these parameters, we compared the IBL approach with a simple and common replica selection approach called the *average approach*. Experimental results show that the IBL prediction algorithm achieves better accuracy (5% to 10% higher) than the *average approach* in both ideal and practical circumstances. In addition, the overhead (time to make a prediction decision) of our IBL algorithm is acceptable (1 to 2 second) since the file transfer time in data-intensive Grid applications is usually long. These promising results demonstrate that the IBL replica selection algorithm can be an efficient approach for replica selection when only limited data sources are available.

The rest of the paper is organized as follows. In Section 2, we discuss the data management issue in data-intensive Grid applications. In Section 3, we discuss some related work and previous approaches. In Section 4, we describe the IBL algorithm for replica selection and its implementation. In Section 5, we present our experiments and results. In Section 6, we draw conclusions and describe future work.

2 Data Management in data-intensive Grid Applications

Replica selection is a key component of data management in date-intensive Grid applications. In this section we therefore discuss data management issues. In general, the management of data involves four main components: a metadata catalog, a replica location service, a replica selection service, and a data access service.

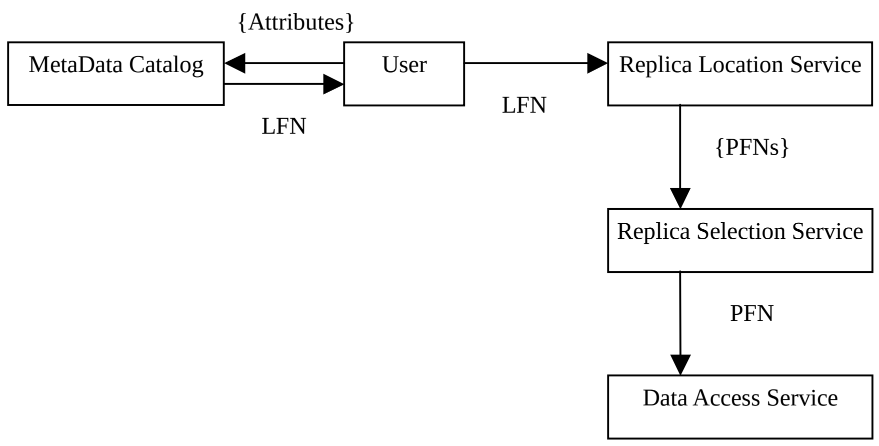


Figure 1 Management of data in data-intensive Grid applications.

Figure 1 shows the interactions of the four components. To access a dataset, a user first queries a metadata catalog by providing some attributes of the dataset. A logical file name (LFN) for the dataset is returned. With the LFN, the user calls the replica location service to retrieve a list of physical file names (PFNs) for that LFN. These PFNs are the physical locations of replicas. The replica selection service then selects one of the PFNs from the candidate set. Finally, the selected replica is accessed by a data access service, for example, by using GridFTP.

A metadata catalog is a service that maps attributes to logical file names. For instance, Chervenak et al. [17] introduce a prototype metadata catalog service MCS. The MCS includes a database in which some attributes of datasets, such as data type, creator, creation time, and modification time are stored. When a user queries MCS by providing a number of attributes, it will return the LFN of the dataset that possesses these attributes.

A replica location service is a registry service that maps the logical filename to a set of physical file names. Giggle is one example of a replica location service [18]. Whenever a user queries with a LFN, giggle searches the local replica catalog (LRC) and replica index (RLI) and outputs a set of PFNs that map to the LFN.

The replica selection service is a service that chooses a replica from among those spread across the Grid based on some criteria. A general criterion is to pick out the replica that has the fastest throughput. Since it is costly to transfer large files, there is much benefit in selecting the most appropriate replica. However, replica selection is complicated because it can involve several components, including networks, CPU, and disks in the end-to-end data path, each of which can experience many unexpected variations [10].

We address the replica selection problem in this paper. We propose a lightweight approach for a practical circumstance, in which much less data is available and current approaches [8][9][10][11][12] may not apply. Our approach uses the IBL technique to make a replica selection based only on historical GridFTP log files. Moreover, in our IBL algorithm, we pick the replica site using a simple relative metric. That is, instead of calculating absolute values of file transfer times for all replicas, we pick the site based on the relative rankings of their file transfer times.

3 Related and Previous Work

Since the replica selection problem is actually the prediction of the best replica site, a proper choice of prediction approach is crucial. Currently, a number of prediction approaches have been used in many fields.

One approach used in system prediction is to construct models for each system component and then combine these performance models for a prediction of the full system. For example, Shen and Choudhary [19] construct two component models, network and disk access, to predict I/O performance. This approach may not be practical, however, because in a shared environment the performance of the components may be dynamic [20]. Moreover, combining those models may omit some significant effects that happen between pairs of components [21].

Another approach for predicting system performance is to use observations on the whole system. With this approach, a prediction model can be constructed without detailed information about underlying components. Also such predictions can capture the interactive effects between components. Numerous tools have applied this approach. For example, the Network Weather Service [22] provides the TCP/IP throughput forecasts based on observations from past end-to-end network performance; and NetLogger [23], Web100 [24], and iperf [25] predict network behavior using historical information on the whole system. This approach of forecasting end-to-end behavior from historic performance of the entire system has been applied to predict file transfer time in some applications. For example, Faerman et al. [11] introduce an AdRM model that uses the NWS probe data and adaptive linear regression models to predict file transfer time in the Storage Resource Broker [26] and SARA [27] applications. Swany et al. [12] construct cumulative distribution functions of past history to derive predictions of transfer times. Vazhkudai and Schopf [9] apply a regression model on the combined current network load variation and end-to-end throughput observations from past GridFTP file transfers to predict the time of file transfer. In their later work [8][10] they present techniques to combine observations of network status information, end-to-end application behavior, and disk I/O throughput load data. They develop a set of regression models to derive predictions, and they achieve relatively good accuracy; however, their algorithms require large amounts of data (NWS network variations, GridFTP logs, and I/O disk throughput). Similar to these approaches, our IBL algorithm makes predictions based on historical performance of the entire system.

Instance-based learning (IBL) is widely used in many fields. For example, Atkeson et al. [28] use the IBL technique to train task models for control; Fuentes and Gulati employ IBL to determine atmosphere temperatures [29]; Telelis and Stamatopoulos solve combinatorial optimization problem based on IBL [30]; Lattner extracts information for the generation of metadata by using the IBL technique [31]. Furthermore, IBL is being increasingly used in many prediction applications and has attained promising performance. Richard Gibbons [15] uses IBL to implement a Historical Profiler to predict the execution times of parallel applications; Smith et al. adopt the IBL technique [13] to predict the run times of parallel applications; Kapadia et al. [14] describe an application of IBL for the prediction of run-specific resource-usage in a computational Grid environment; and Kroeger et al. [16] predict file systems events with the IBL technique. This wide range of applicability motivates us to use IBL in our approach.

4 IBL Algorithm for Replica Selection

In this section we introduce our IBL algorithm for replica selection. We first review instance-based learning (Section 4.1) and then address the implementation of our approach (Section 4.2).

4.1 IBL Algorithm for Replica Selection

In Section 4.1.1 we discuss instance-based learning algorithms. In the following sections (4.1.2 to 4.1.4), we elaborate on the instances, distance metrics, and target functions in our IBL algorithm. In Section 4.1.5 we discuss performance factors affecting our IBL algorithm.

4.1.1 IBL Algorithms

Learning, as defined in the field of artificial intelligence, is the process of developing principles for knowledge acquisition and improving performance over time. A learning algorithm should have three basic characteristics: (1) the ability to acquire new knowledge, (2) the ability to generalize the pattern of the knowledge, and (3) the ability to autonomously adjust in order to improve performance. Learning algorithms are used in different fields, for instance data mining, natural language processing, and machine learning.

We can divide all learning algorithms into two categories: lazy learning algorithms and eager learning algorithms. A lazy learning algorithm stores new data without further analysis and defers processing the data until a request is made. Thus, a lazy learning algorithm learns a function that is tuned to a specific request. On the other hand, eager learning algorithms process the data immediately to learn a function that is independent of the type of new requests.

IBL [32] is a lazy learning algorithm. A novel aspect to an IBL algorithm is that its prediction is based on the similarities between the request and past events. (An *event* is defined as an observation such as a file transfer, a program run, or a resource selection, and that defines the completion of a specific task.)

Compared with other algorithms, IBL algorithms offer several advantages. Since IBL requires only past events for making predictions, it can be used when only limited data, such as historical file transfer log files, is available. By measuring the similarities and extracting information from similar past events, IBL can attain better performance (because it is believed that similar events have similar behaviors [32]). Moreover, IBL may be a suitable solution in domains that are not completely understood [33], for example, the replica selection problem. These advantages make IBL applicable in our replica selection approach.

To describe an IBL algorithm, we introduce two concepts: an instance and a target function.

An *instance* is a set of data about an event. For example, with respect to file transfers, an instance is a set of data about these transfers, such as destination, source, transferred file, and transfer time. An instance is often represented as a vector of values. The vector can be used to measure the similarity between instances, as described in Section 4.1.3.

A *target function* is a function that, given a set of instances, calculates some properties of these instances and uses these properties to make a prediction. For example, given a set of file transfer instances between two fixed points, a target function might calculate the average time of these file transfers to predict the time of future transfers.

The following briefly describes the steps of an IBL algorithm:

a. A set of past instances is stored in a history database.
b. When the prediction of a query instance (the current instance to be predicted) is being made, the IBL algorithm examines the similarities between the query instance and all past instances based on a specific metric. The similarity is called *distance* and the metric is called a *distance metric*.
c. The algorithm then picks a set of nearest *neighbor instances* (based on the distance) since these are likely to be similar to the query instance.
d. A target function of the nearest neighbor instances is then applied to make a prediction for the query instance.
e. After prediction, the query instance is stored in history database.

Figure 2 briefly shows the relations among these instances: past instances, query instance, and neighbor instances. The five-point star symbol represents the query instance. A neighborhood (the circle) can be found based on calculated distances. To those neighbor instances (four-point star symbols) a target function can be applied to achieve prediction.

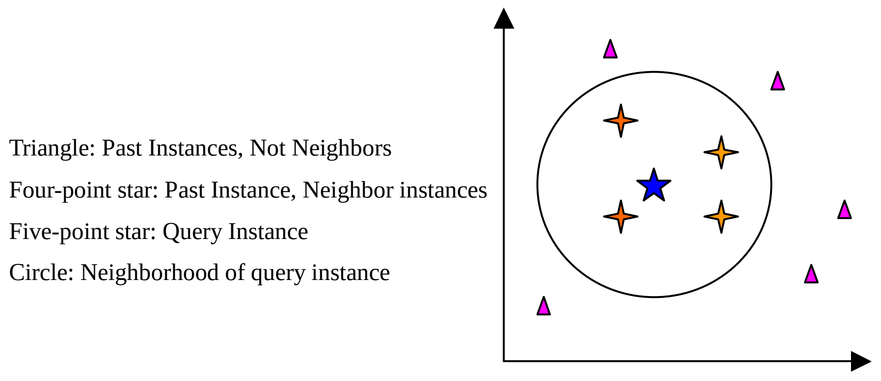

Triangle: Past Instances, Not Neighbors
Four-point star: Past Instance, Neighbor instances
Five-point star: Query Instance
Circle: Neighborhood of query instance

Figure 2 Illustration of relations among instances.

4.1.2 Instances

As defined in Section 4.1.1, an instance in an IBL algorithm is a set of data describing an event. In our IBL algorithm, the event refers to a file transfer. Thus, we represent the instance as the vector $(happenday, happentime, filesize, source, dest, transfertime)$ where each feature is one attribute of a file transfer. The value *happenday* is the day when the file is transferred; the value *happentime* is the time when the file transfer is started; *filesize* is the size of transfer file; *source* is the source site of the transfer; *dest* is the destination site of the transfer; and *transfertime* is the total time of the file transfer. These features are the major factors involved in a file transfer and can be easily recorded by logging GridFTP features [34]. In our IBL algorithm, we use the features *happentime*, *happentime*, and *filesize* to decide the distance between instances.

4.1.3 Distance Metrics

In this section, we define the distance metrics used in our IBL algorithm. Generally, a good distance metric should accurately reveal the relationship between two instances: similarity or dissimilarity. In particular, three major issues are considered by a distance function: what attributes of instances should be included to determine the similarity between two instances; how much each attribute affects the similarity; and how one can integrate the differences of all attributes to achieve an accurate aggregate distance. These provide us the criteria to choose an appropriate distance metric for our IBL algorithm.

A possible distance metric is the Chebychev Distance, which is defined as

$$D(x, y) = \max_{i=1}^{m} |x_i - y_i| \ ,$$

where *x* and *y* are two input vectors (*x* typically represents a past instance and *y* the query instance to be predicted), *x* has the same number of attributes as *y*, and *m* is the number of input variables (attributes) of the vector *x* (or *y*). This metric is simple: although it considers all features, it bases the distance on only one feature, the one that has the biggest difference between the two vectors. For instance, assume there are three vectors v_1, v_2, and v_3, each of which has three features. Specifically, v_1 is (3, 4, 5), v_2 is (6, 4, 5), and v_3 is (6, 5, 7). The Chebychev Distance between the two vectors v_1 and v_2 is the same as the

Chebychev Distance between v_1 and v_3, namely, 3. Clearly, v_2 is more similar to v_1 than is v_3, a fact that isn't reflected in the calculated Chebychev Distances (both are equal). Thus the Chebychev Distance is not an appropriate distance metric for cases when all these features of the instance should be included in the distance function.

Another distance function is the Manhanttan Distance, which is defined as:

$$D(x, y) = \sum_{i=1}^{m} |x_i - y_i| \; .$$

Different from the Chebychev Distance, the Manhanttan Distance includes all the features of a vector. But it omits the possible combined effect between pairs of features by simply adding the differences of all features together. Thus for the cases in which the combined effects are important, the Manhanttan Distance is not a proper choice.

Another distance metric, the Euclidean Distance function is defined as:

$$D(x, y) = \sqrt{\sum_{i=1}^{m} (x_i - y_i)^2} \; .$$

The Euclidean Distance is widely used in many applications. For instance, Gvenir et al. [35] calculate Euclidean Distances to make financial prediction, Faloutsos et al. [36] apply Euclidean Distance metric in data mining, and Yi Lu Murphey et al. [37] use Euclidean Distances in their image retrieval project. Furthermore, the Euclidean Distance avoids the shortcomings of Chebychev Distance and the Manhanttan Distance: it considers effects of all features and takes the combined effect into consideration.

The Euclidean Distance metric has a transformed form, Weighted Euclidean Distance, which is defined as

$$D(x, y) = \sqrt{\sum_{i=1}^{m} (x_i - y_i)^2 * w_i} \quad \text{Where } \sum w_i = 1.$$

Here w_i is the weight of each attribute. Besides possessing the same advantages as Euclidean Distance metric, the Weighted Euclidean Distance may be more appropriate because it can embody the features' *relative* importance to the distance by assigning them distinct weights.

In our IBL algorithm, (1) all attributes of instances affect the similarity and should be included in distance function; (2) the combined effects between pairs of attributes are useful; and (3) each attribute of the instance contributes differently. We, therefore, chose the Weighted Euclidean Distance. Thus, in our IBL algorithm, the distance between two instances I_1 and I_2, $D\ (I_1, I_2)$ is defined as follows:

$$D(I_1, I_2) = \sqrt{(I_1f - I_2f)^2 * W_f + (I_1t - I_2t)^2 * W_t + (I_1day - I_2day)^2 * W_{day}} \quad Where\ W_f + W_t + W_{day} = 1.$$

Here I_1f refers to the attribute *filesize* of instance I_1, I_1day refers to the attribute *happenday* of I_1, I_1t refers to the attribute *happentime* of I_1, and so on. W_f, W_{day}, and W_t are weights of these attributes. Once the metric is chosen, a set of nearest neighbor instances can be selected with calculated distances.

4.1.4 Target Functions

The target function is also a significant component of our IBL algorithm. In this section, we discuss different choices of target functions.

To make our algorithm less computationally intensive, we examined two simple options for the target function, *pick top ranked* and *pick major ranked.* The process of *pick top ranked* is as follows: Pick the nearest neighbor instance having the highest throughput, and choose the replica server of this instance. The process of *pick major ranked* is as follows: Pick a number of nearest neighbor instances with the highest *n* throughput and choose the most frequent replica server associated with these instances. In both cases, we define the *throughput* as *total transfer time/file size*.

An example of these two target functions is as follows. Assume we have the sorted list of nearest neighbor instances as Table 1. Since instance I_1 has the largest transfer throughput, the *pick top ranked* method will select I_1 and choose the replica server of I_1, A, as best replica server. On the other hand, *pick major ranked* will pick a number of instances from the list, which ranges from *1* to *n*. For instance, when $n = 3$, it may choose instances I_1

(ranked first), I_2 (ranked second), and I_3 (ranked third). Then among these three instances, it finds the most frequent server, B, to be the best replica server.

Table 1 An example of two target functions.

Neighbor Instances	Throughput (MB/s)	Replica Server
I_1	1.2	A
I_2*eighbortancess that our IBL prediction he three data*□□□	1.1	B
I_3	0.9	B
…	…	…
I_n	0.1	E

4.1.5 Performance Choices

In general, the performance of our IBL algorithm dependents mainly on the following choices:

1. Proper weights for the Weighted Euclidean Distance function, since different weights can result in distinct distances and lead to large variation in performance.
2. A proper number of nearest neighbor instances, since this can greatly affect the accuracy and the overhead of the IBL prediction algorithm.

Making proper choices is a complicated task in general because we have no idea of the relative importance of each attribute to the Weighted Euclidean Distance, and we don’t know how the number of neighbor instances affects the accuracy. To find good choices, we conducted some search experiments, discussed in Section 5.3.

4.2 Implementation of the IBL Algorithm for Replica Selection

The implementation of our IBL algorithm comprises four components: *initialization of cases_base* (the database storing past instances), *parameter setting*, *IBL replica selection*, and *update cases_base*. Figure 3 shows the interactions of these four components.

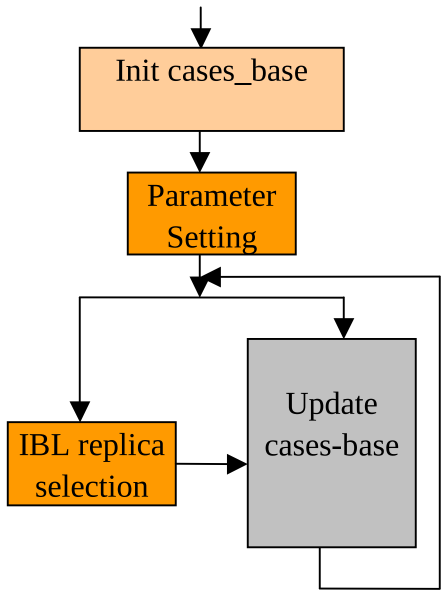


Figure 3 Implementation of our IBL algorithm.

Component: Init cases_base

Init cases_base is the first step in our model. In this component, a database, namely, cases_base, is initialized by storing a number of past instances. As described in Section 4.1.2, each instance includes the following information: *happenday, happentime, file size, source, dest,* and *total transfer time.* The implemented program, Initcasesbase, reads the GridFTP log files of historical file transfers and stores the extracted information into cases_base. We use perl as the implementation language and PGI as the interface to access the database. Table 2 gives a sample of the cases_base we stored.

Table 2 A sample of cases_base stored.

Day	Time	Source	File Size, MB	Destination	Transfer Time, s
Wed	5:00	capitol.cs.uchicago.edu	128	Bmajor.cs.uiuc.edu	112
Tue	18:00	cirque.ucsd.edu	512	capitol.cs.uchicago.edu	642

Component: Parameter Setting

The parameter setting component assigns the parameters required by the IBL replica selection component. These parameters include the weights in the Weighted Euclidean Distance function (i.e. weights of *happenday*, *happentime*, and *file size*), the number of nearest neighbor instances (defined as an integer *K*), and the choice of target functions (*pick top ranked* or *pick major ranked*). Currently, we assign them manually.

Component: IBL replica selection

This IBL replica selection component is the core part of our IBL implementation. This component involves three steps: when the replica selection decision is being made, (1) compute the Weighted Euclidean Distances between the current instance and each past instance of cases_base; (2) pick *K* nearest neighbor instances from cases_base based on calculated distances; and (3) apply the target function to find best replica server.

We implement it as a perl program that takes the following inputs: the query instance (a set of data about current file transfer), the name of cases_base, weights of all features in distance function, the number of nearest neighbor instances *K*, and the choice of target function (*pick top ranked* or *pick major ranked*). The program outputs the predicted replica server.

Component: Update cases_base

After replica selection is made, this update cases_base component is activated to record the information on current file transfer into cases_base using some specific update strategy. In addition the update component can also be activated manually to execute its other functionality: check cases_base and delete stale past instances. For instance, we can run a shell program, for example, *Delete –days 10*, to check cases_base and delete the past instances that happened ten days before.

5 Experiments and Results

To validate our IBL approach, we applied it on the GridFTP log files collected on a real Grid platform. In Section 5.1, we introduce the setup of the platform and the process of collecting the GridFTP log files. The steps of applying the IBL approach to the collected log data are also presented in this section.

As described in Section 5.1, the collected GridFTP log data contains information about a number of file transfers. Each of these file transfers actually involves five simultaneous transfers (sends or receives) between the client and five remote servers. When the client receives files from the five remote servers simultaneously, however, the recorded transfer times of these receives may be useless because of the possible contentions in the client machine. On the other hand, since there is no such contention in the client machine when sending files to those remote replica servers simultaneously, the recorded transfer times of these sends are useful: we can apply the IBL algorithm only to the send log files. If we can verify that the send process has the same relative performance as receive in our test bed, we can validate the effectiveness of the IBL algorithm only by the results on send log data. Therefore, we present in Section 5.2 a preliminary experiment we conducted to compare the relative performance of send and receive.

As noted in Section 4.1.5, some parameters need to be pre-assigned for our IBL algorithm. In Section 5.3, we describe search experiments we conducted to find the proper set of parameters.

In Section 5.4, we compare our IBL approach with a simple and common approach (the so-called average approach) on two collected GridFTP send log data files. We find that in both cases the IBL approach achieves higher accuracy (about 5% to 10% higher) than the average approach. Also, the overhead (time to make a prediction) of our IBL approach is acceptable. These promising results demonstrate that the IBL approach can be an efficient tool for replica selection.

5.1 Preparations

In this section, we describe some preparations for the following experiments. Specifically, in Section 5.1.1, we introduce the Grid platform on which the GridFTP log files are collected; in Section 5.1.2, we present the process of collecting GridFTP log data; and in Section 5.1.3, we elaborate on the steps of applying the IBL algorithm to the collected log data.

5.1.1 Platform

The experiment platform included one client and five GridFTP servers. GridFTP was the communication medium. For the client, we selected host capitol.cs.uchicago.edu, where replica requests were made. We chose five remote hosts as replica servers: bmajor.cs.uiuc.edu, cirque.ucsd.edu, beak.cs.wisc.edu, i2-53.cs.uh.edu, and torc17.cs.utk.edu. All these hosts were workstations installed with Linux and the Globus 2.4. Table 3 shows additional information about them.

Table 3 Information about the hosts in our test bed.

Name	Site	Brief Description
Bmajor.cs.uiuc.edu	University of Illinois at Urbana Champaign	CPU: Intel 266MHz, Platform: i686, Memory: 256MB
Cirque.ucsd.edu	University of California, San Diego	CPU: AMD 1.7GHz, Platform: i686, Memory: 1GB
I2-53.cs.uh.edu	University of Houston	CPU: Intel 900MHz, Platform: ia64, Memory: 1GB
beak.cs.wisc.edu	University of Wisconsin	CPU: Intel 900MHz, Platform: i686, Memory: 512MB
torc17.cs.utk.edu	University of Tennessee at Knoxville	CPU: Intel 1.7GHz, Platform: i686, Memory: 1GB
Capitol.cs.uchicago.edu	University of Chicago	CPU: AMD 266 MHz, Platform: i686, Memory: 128MB

We also generate eleven files as test files, which ranged from 1MB to 1024MB in steps of order 2. Before any experiment, these test files were copied to all the servers.

5.1.2 Collecting GridFTP Data

On the platform, to collect a set of GridFTP log files, we conducted 110 GridFTP file transfers, which lasted for ten days. In each file transfer, a randomly selected test file was sent from the client to five remote servers simultaneously (the reason will be discussed in Section 5.2). The time interval between sequential file transfers was set as a random value between 1 and 10,800 seconds.

After each file transfer, some of its information was logged. The logged data comprised two parts: property information and transfer information. Property information included happen day, happen time, file size, and transfer source; transfer information included five pairs of (transfer destination, transfer time) corresponding to each remote server and its transfer time. In Table 4, we show an example of the logged information for one file transfer.

Table 4 Sample of logged information for one file transfer.

Property Information				Transfer Information	
Day	Time	Source	File Size, MB	Dest	Transfer Time(s)
Wed	3:00am	capitol.cs.uchicago.edu	128	Bmajor.cs.uiuc.edu	123
				Cirque.ucsd.edu	211
				Torc17.cs.utk.edu	176
				I2-53.cs.uh.edu	179
				Beak.cs.wisc.edu	100

Similar information was collected for all 110 transfers. We call this set of log files as the “collected GridFTP log data”.

5.1.3 Applying the IBL Prediction Approach to Log Data

The collected log data comprised a set of log files, each corresponding to one file transfer. Before applying the IBL algorithm, we sorted these log files into an ordered list by the beginning time of their corresponding file transfers. The head log file of the list corresponded to the file transfer that occurred earliest.

The following steps briefly show how we applied the IBL prediction approach to the log data: (1) init the history database to be empty; (2) take the head log file from the sorted list; (3) with set parameters, make replica selection for the head log file (for the first selection decision, we chose a random server because the history database was empty), and check the correctness of the prediction; (4) update the history database with some update strategy; (5) repeat steps 2–4 until all log files are handled; and (6) calculate the prediction accuracy (*accuracy = number of correct prediction /total number of predictions*) and the overhead (*overhead = total time to make all the predictions/number of predictions*).

The correctness of the replica selection can be checked as follows. When the replica selection decision is made, a server is selected as predicted best replica server. We know from the preceding section that the log file of a file transfer records five transfer times, each of which corresponds to one server. With these transfer times, a real best replica server, which has the fastest transfer throughput in this file transfer, can be found. By comparing the predicted best replica server and the real best replica server, we can judge this replica selection correct or not.

For the update strategy, we examined two choices: *Addall* and *Addone*.

Addall: After making a replica selection for a file transfer *l*, five entries are added into the history database (cases_base). These five entries have the same attributes' values (day, time, source, and file size), which are set as the corresponding values of *l*. The only differences among the five entries are the destination and transfer time. For each entry, the attribute destination is set as one of the five servers, and the attribute transfer time is set as the corresponding transfer time of that server in the file transfer *l*. With this update strategy, this history database is an ideal one: for any file transfer, all information including transfer times of all five servers is kept. Using these transfer times, we can accurately judge which server is best for any file transfer. These judgments of historical file transfers help to make an effective replica selection. Thus, to examine the performance of our IBL approach in an ideal case, we applied this strategy.

Addone: Instead of storing five entries, only one entry is added into cases_base. Of this entry, the values of day, time, source, file size are set as the corresponding values of *l*. Meanwhile, the attribute destination of this entry is set as a server that is randomly selected from the five remote servers, and the attribute transfer time is set as the corresponding transfer time of the selected server. Obviously, with *Addone*, the cases_base is not as ideal as *Addall*: for any file transfer, much less data—only the information of one server—was kept. Nevertheless, we examined this strategy because it is more similar to practical circumstances: each historical file transfer is related only to one server, which can be any one of the five remote servers.

Besides these two update strategies, there exists another possible choice: *Addbest*. Similar to *Addone*, it adds only one entry into cases_base, and its attributes (day, time, source, and file size) are set as the corresponding values of *l*. However, the attribute destination of the entry is not set as a randomly selected server, as in *Addone*; instead it is set as the server that has the fastest throughout in file transfer *l*. We did not apply this strategy because it is almost identical to *Addall* for our IBL algorithm. For instance, when target function is *pick top ranked*, our IBL algorithm will pick a server that has fastest throughput (best server) from similar historical file transfers. Even though *Addall* stores five entries for each file transfer, the only useful information is the entry that corresponds to the best server. In other words, the actual useful information stored by *Addall* is almost the same as the information stored by *Addbest*.

5.2 Preliminary Experiment: Relative Performance of Send and Receive

As discussed in Section 5.1.2, to verify our IBL algorithm, we need to log the information (including transfer times) of simultaneous file transfers (sends or receives) between one client and several remote replica servers. With these transfer times, we can judge the relative ranking of all servers for one file transfer. Based on this, we can make a replica selection decision.

When we receive files from five remote servers simultaneously, however, the recorded transfer times may not be useful because of the possible contention in the client machine. The process of receiving a file from several remote servers at the same time is actually several simultaneous sends from those servers to the client machine. These send processes are controlled by different remote servers, which know nothing about each other. Since there is a lack of centralized control on these send processes while the servers keep sending files to the client machine, some contentions on the client machine may happen. This contention may affect the sends in an unexpected way, and lead to the distorted transfer times. In such cases, the ranking information of servers is not useful. Therefore we did not attempt to validate our IBL algorithm by conducting experiments on the log files of these receives.

On the other hand, when we send files from the client to several remote servers, all the simultaneous send processes are controlled by one machine (the client). Hence, there are no previous contentions on the client. Without these contentions, the ranking information of the servers is reliable. Thus, to verify the effectiveness of our IBL approach, we chose to conduct experiments only on GridFTP send log files. This decision is justified as long as sends have the same relative performance as receives for our test bed. Experiments in this section provide evidence in support of this methodology.

5.2.1 Experimental Methodology

On the platform introduced in Section 5.1.1, we ran the first experiment as follows: with a test file *f*, (1) send *f* from client to server 1 (bmajor.cs.uiuc.edu) and then receive *f* from server 1 to the client; (2) send *f* from client to server 2 (cirque.ucsd.edu) and then receive *f* from server 2 to client; (3)–(5) send and receive for the rest three servers (beak.cs.wisc.edu, i2-53.cs.uh.edu, torc17.cs.utk.edu). We conducted the successive transfers, from the client to one sever and from the server to the client, in order to record the send and receive performances at almost the same time. We repeated the loop (1)–(5) ten times for statistical reliability; after these ten runs, we averaged the transfer times. We conducted this experiment for all eleven test files, f_1 (1 MB) to f_{11} (1 GB).

5.1.2 Results and Analyses

Table 5 shows the average transfer times and the rankings of five servers (based on their locations, we name them as UCSD, UH, UTK, UIUC, and WISC) for all test files (1MB to 1GB). Specifically, for each test file, we list the send times of servers, receive times of servers, ranking of servers according to their send times, and ranking of servers according to their receive times.

Table 5 Relative Performance of send and receive for eleven test files.

Files	Transfer Time (s)	UCSD	UH	UTK	UIUC	WISC	Rankings (leftmost server has longest transfer time)				
1MB	Send	4.91	2.61	1.98	2.04	1.02	UCSD	UH	UIUC	UTK	WISC
	Receive	3.60	2.72	1.80	1.84	0.99	UCSD	UH	UIUC	UTK	WISC
2MB	Send	4.76	3.33	2.65	2.15	1.51	UCSD	UH	UTK	UIUC	WISC
	Receive	4.56	4.09	2.55	2.35	1.64	UCSD	UH	UTK	UIUC	WISC
4MB	Send	6.76	5.17	3.94	2.95	1.81	UCSD	UH	UTK	UIUC	WISC
	Receive	6.59	5.75	3.47	2.68	1.61	UCSD	UH	UTK	UIUC	WISC
8MB	Send	11.8	8.01	5.17	4.47	3.06	UCSD	UH	UTK	UIUC	WISC
	Receive	11.31	7.74	5.07	3.78	2.66	UCSD	UH	UTK	UIUC	WISC
16MB	Send	21.3	14.06	9.41	6.95	5.25	UCSD	UH	UTK	UIUC	WISC
	Receive	19.95	15.25	8.71	6	4.4	UCSD	UH	UTK	UIUC	WISC
32MB	Send	41.38	28.65	19.7	14.43	12.03	UCSD	UH	UTK	UIUC	WISC
	Receive	37.63	26.45	16.17	10.55	7.8	UCSD	UH	UTK	UIUC	WISC
64MB	Send	78.42	55.78	45.46	29.32	25.12	UCSD	UH	UTK	UIUC	WISC
	Receive	71.61	51.34	31.66	20.24	17.14	UCSD	UH	UTK	UIUC	WISC
128MB	Send	148.35	128.73	115.99	96.21	84.37	UCSD	UH	UTK	UIUC	WISC
	Receive	139.04	115.74	86.55	77.27	70.95	UCSD	UH	UTK	UIUC	WISC
256MB	Send	318.13	237.03	173.6	114.05	92.32	UCSD	UH	UTK	UIUC	WISC
	Receive	325.37	228.28	120.78	78.52	61.75	UCSD	UH	UTK	UIUC	WISC

512MB	Send	644.32	500.06	356.21	230.57	183.94	UCSD	UH	UTK	UIUC	WISC
	Receive	593.94	406.81	242.15	151.81	121.36	UCSD	UH	UTK	UIUC	WISC
1GB	Send	1287.9	977.91	715.28	452.98	396.45	UCSD	UH	UTK	UIUC	WISC
	Receive	1168.5	744.6	478.03	276.82	260.99	UCSD	UH	UTK	UIUC	WISC

From this table, we see that for each test file, although the absolute performance of the sends and receives are different, the rankings of servers for the two processes are the same. This means that for our test bed send and receive have the same relative behavior. This finding assures us that we can verify our IBL algorithm only on GridFTP send log files in the following experiments.

5.3 Experiment 2: Search for Proper Parameter Values

To evaluate the proper parameters to be used in our test experiments, in this section we describe a number of search experiments on a set of GridFTP send log files. The GridFTP log data, *Run1*, was collected by using the method described in Section 5.1.2. In Section 5.3.1 we present the search experiments for the number of nearest neighbor instances *K;* in Section 5.3.2 to Section 5.3.4 we present the search experiments for the three weights of the Distance Function individually (file size, day, and the weight of attribute hour); finally, in Section 5.4.5 we decide the proper set of parameters by incorporating the values of each parameter that are found in previous sections (Section 5.3.1 to Section 5.3.4).

5.3.1 Number of Nearest Neighbor Instances *K*

We begin with our search for the number of nearest neighbor instances.

Search Approach

To describe the search approach, we first define the *performance* of one *K* (number of nearest neighbor instances). With a fixed *K*, one can use different sets of the three weights (file size, day, and hour) that may result in different accuracies. Since we had no idea of the proper set of weights, we crudely choose four sets of weights (w_1 to w_4, corresponding to four typical cases) and averaged their accuracies and overhead as the performance of the fixed *K*. The four sets were w_1 = (0.33, 0.33, 0.33), w_2 = (0.8, 0.1, 0.1), w_3 = (0.1, 0.8, 0.1), w_4 = (0.1, 0.1, 0.8).

We performed the following steps to compute the performance of one *K*: (1)set *K* as one value and set the target function; (2) set the set of three weights; (3) as described in section 5.1.3, applied IBL prediction approach to the set of GridFTP send log file *Run1* and got the accuracy and overhead; (4) repeated steps 2 and 3 by setting the weights as the four sets w_1 to w_4; and (5) with the four accuracies and overheads, computed the average values of them and output them as performance of this *K*.

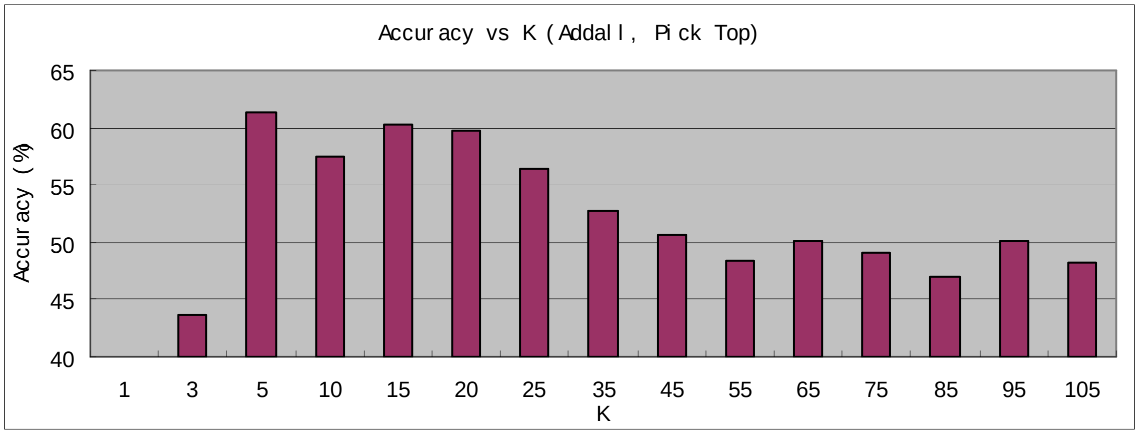
The search approach was as follows: We examined the performance of a set of *Ks* (candidates), and choose the good value of *K* that achieves *good* performance. Here we define *good* performance as best accuracy or good accuracy with acceptable overhead.

As discussed in Section 4.1.4 and Section 5.1.3, there are two choices of target functions (*pick top ranked* and *pick major ranked*) and two choices of update strategies (*Addall* and *Addone*). Because the two update strategies will result in different historical information and the two target functions correspond to different ways to process the historical information, four combinations of these choices—(*Addall, pick top ranked*), (*Addall, pick major ranked*), (*Addone, pick top ranked*), (*Addone, pick major ranked*) — refer to four distinct cases. Thus, we ran four sets of search experiments, each corresponding to one of the cases.

For the case (*Addall, pick top ranked*), after we ran experiments on the set of *Ks* (1, 3, 5, 15, 25, …, 105), we observed several phenomena: (1) the accuracy kept decreasing when *K* ranged from 20 to 105; (2) the overhead kept increasing; and (3) the average accuracy when *K*=105 (48%) was much smaller than the achieved maximum accuracy (about 60%). Based on these findings, we considered it unnecessary to choose *K* bigger than 105. Therefore, we choose (1, 3, 5, 15, 25, …, 105) as the candidate set of *Ks* for the case (*Addall, pick top ranked*). Similarly, we choose (1, 3, 5, 15, 25, …, 105) as the candidate set of *Ks* for (*Addall, pick major ranked*). On the other hand, for the cases (*Addone, pick top ranked*) and (*Addone, pick major ranked*), since the size of cases_base is at most 110, we selected a set of *Ks* from [1, 110], namely, (1, 3, 5, 15, 25, …, 105).

Results and Analyses

The following four plots show the results of the four sets of experiments. Specifically, Figure 4 is the result for the case (*Addall*, *pick top ranked*); Figure 5 is result for the case *(Addall*, *pick major ranked)*; Figure 6 is for *(Addone, pick top ranked)*; and Figure 7 is for *(Addone*, *pick major ranked)*. Each of the figures includes two plots: the plot *accuracy vs K* and the plot *overhead vs K*. In the plot *accuracy vs K*, the *y* axis is the average value of the four accuracies for the sets (w_1 to w_4) and the *x* axis is the value of *K*. Similarly, in the plot *overhead vs K*, the *x* axis is the value of *K*, and the *y* axis is the overhead.



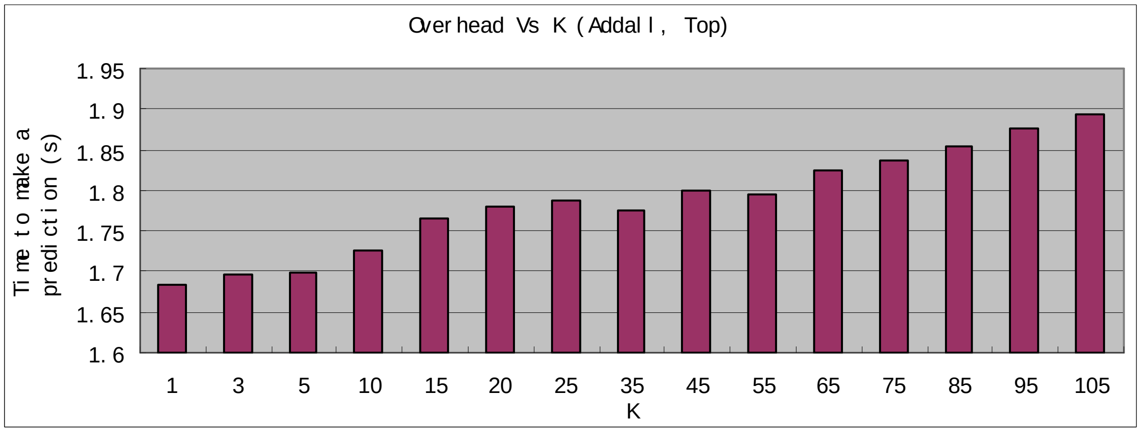
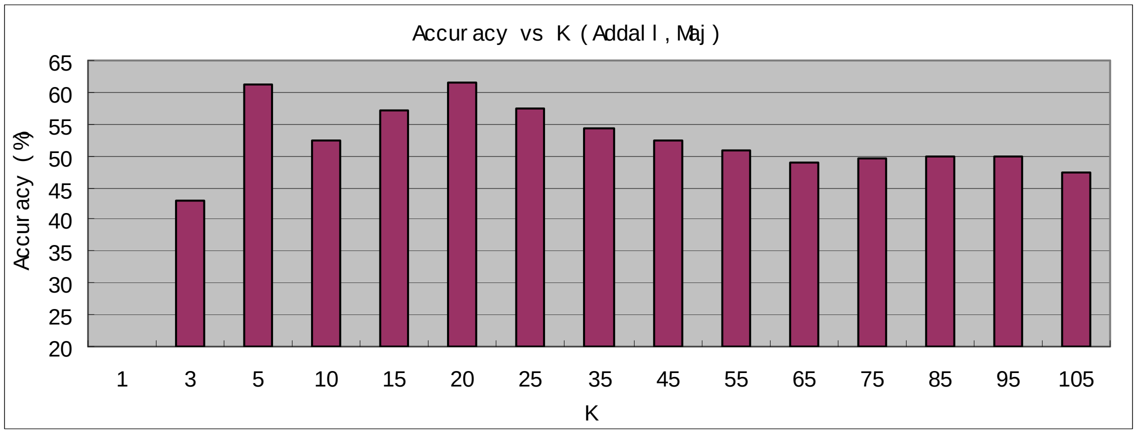


Figure 4 Results of *K* experiment for the case (Addall, Top).



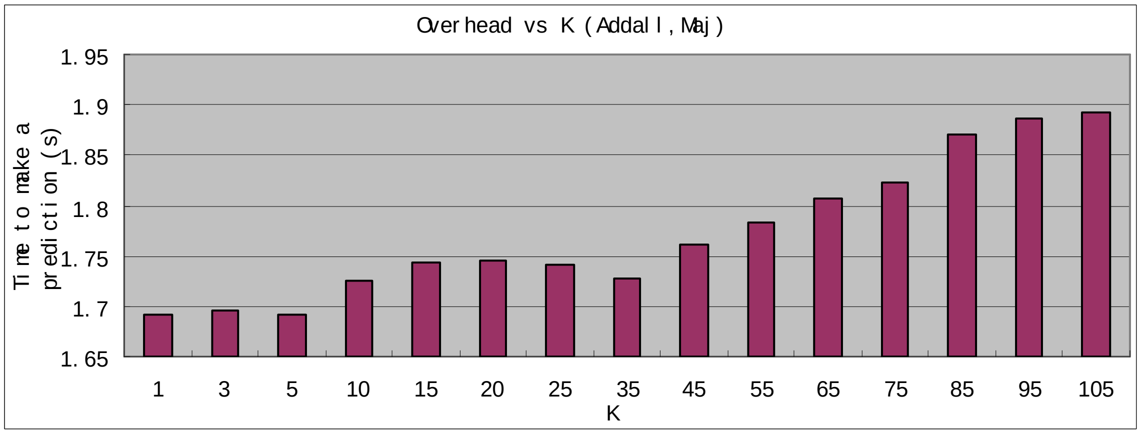
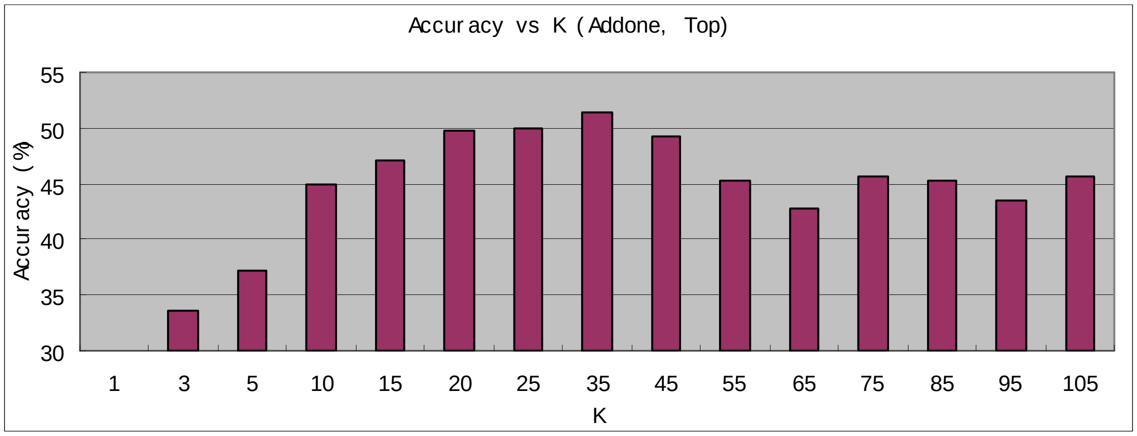


Figure 5 Results of *K* experiment for the case (Addall, Maj).



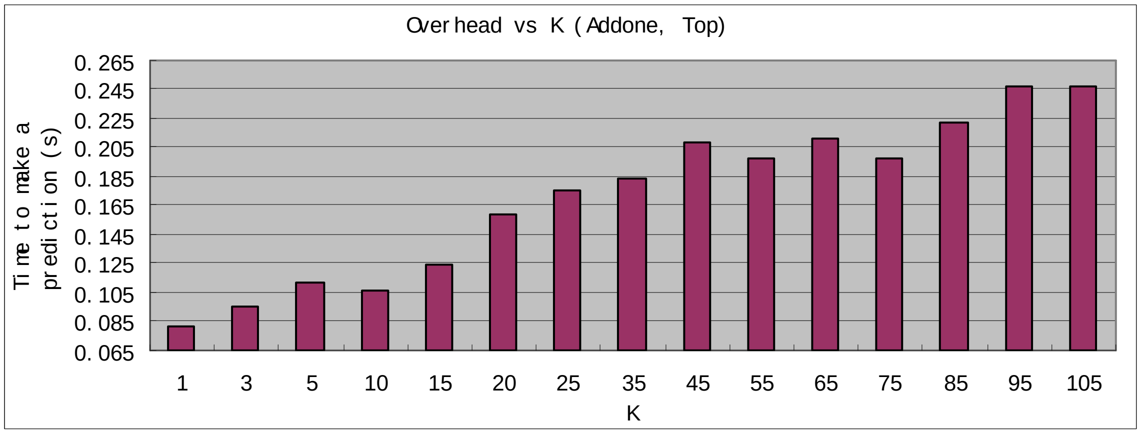
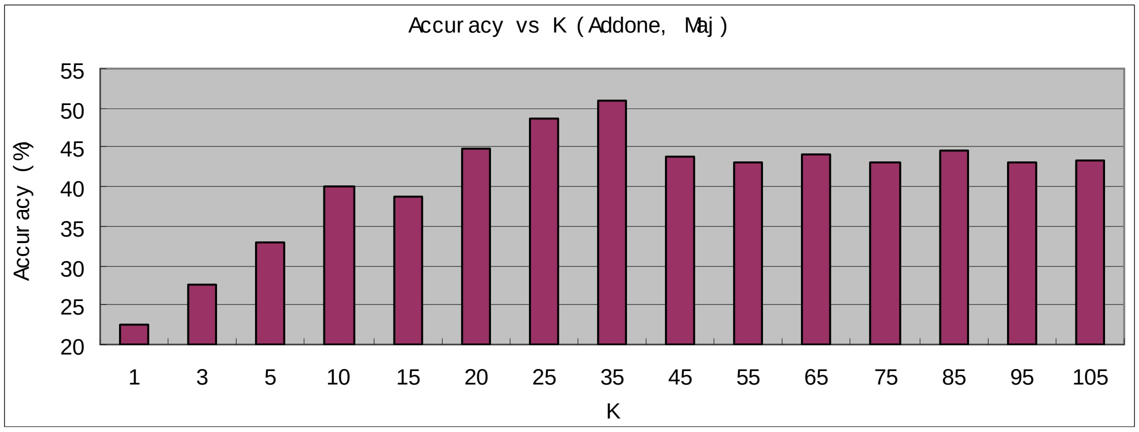


Figure 6 Results of *K* experiment for the case (Addone, Top).



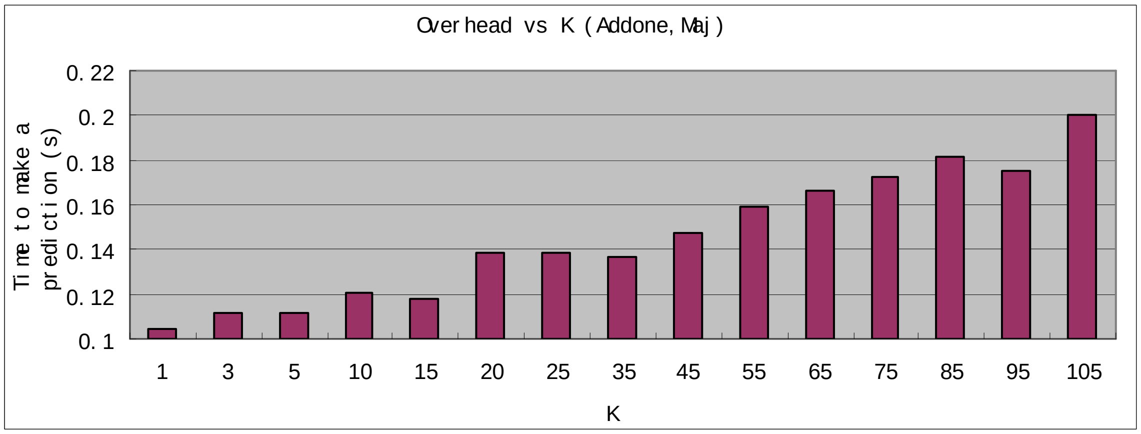


Figure 7 Results of *K* experiment for the case (Addone, Maj).

From these figures, we noted two phenomena. First, all the accuracy curves look like Gaussians except at two points (*K=5* in Figure 4 and Figure 5): as *K* becomes bigger, the accuracy keeps increasing until reaching its maximum, and then the accuracy degenerates. Second, the overhead (time to make a prediction) becomes greater with *K*. These two findings agree with our expectations. By increasing *K*, more useful historical information (the information on similar past instances) can be included, which may result in more effective prediction. When *K* is too big, however, the information of dissimilar historical instances, if included, reduces the accuracy of prediction. Therefore, there exists a good value of *K* that leads to best accuracy. The reason why the overhead increases as *K* is straightforward: the more historical instances that are considered, the more data dealt with by IBL algorithm, and the longer time needed.

As for the two points (*K=5* in Figure 4 and *K=5* in Figure 5), both achieve very high accuracies, a result that is not compatible with the general trend addressed before. Except the two points, however, all the others agree with the Gaussian curve very well. These two points may be due to statistical fluctuations. Thus, we neglect the results of these two exceptional points.

In addition, we also observe that the overhead doesn't change much in all cases although it does increase. Therefore, we focus only on accuracy as the performance measurement of *K*.

We choose those *Ks* that attained best accuracies: 15 for *(Addall, pick top ranked)*, 20 for *(Addall, pick major ranked)*, 35 for *(Addone, pick top ranked)*, and 35 for *(Addone, pick major ranked)*. Obviously, these good values of *Ks* are different for four cases. We explain the difference as follows. *Addall* and *Addone* store different historical information. The distinct patterns of historical information will lead to different good values of *Ks*. And the two target functions represent distinguishing methods to deal with the selected historical data. It is reasonable to expect various performances of them, which would lead to different good values of *Ks*.

5.3.2 Weight of the Attribute File Size

Our next experiment focused on the weight of file size.

Search Approach

The weight of file size in the Weighted Euclidean Distance metric refers to the importance of attribute file size. Different values of them can result in distinct accuracies with our IBL algorithm. Thus, our search approach in this section is as follows: examine accuracies of the IBL prediction when the weight of file size is assigned as all possible values, and choose as the good value of the weight of file size the value that achieves best accuracy.

According to the definition of the Weighted Euclidean Distance for our IBL algorithm in Section 4.1.3, the weight of file size can range from 0 to 1. From this range, we choose nine sample values, (0.1, 0.2,…, 0.9), as representatives of all possible values. We applied the IBL algorithm with the nine weights of file size and choose a good one based on their accuracies.

Specifically, we took the following steps to find the good value: (1) set *K*; (2) set the weight of file size as one value, (3) set the remaining two weights as an equal value (1-weight of file size)/2 (we did so because we had no idea of the proper way to assign these two weights and thus simply set them equal); (4) applied IBL prediction approach to the log data *Run1* and got the accuracy; (5) repeated steps 2—4 with increasing file size weight gradually (i.e., 0.1, 0.2, …, 0.9).

Similar to previous section, we ran four sets of tests for the four cases: *(Addall, pick top ranked)*, *(Addall, pick major ranked)*, *(Addone, pick top ranked)*, and *(Addone, pick major ranked)*. For each case, the *K* has been selected in Section 5.3.1, particularly, 15 for *(Addall, pick top ranked)*, 20 for *(Addall, pick major ranked)*, 35 for *(Addone, pick top ranked)*, and 35 for *(Addone, pick major ranked)*. Then, in each set of test, we assign *K* as the previous selected value for that case.

Results and Analyses

We plot the results in Figure 8 to Figure 11. Each figure corresponds one case: Figure 8 refers to *(Addall, Pick Top ranked)*, Figure 9 refers to *(Addall, Pick Major ranked)*, Figure 10 refers to *(Addone, Pick Top ranked)*, and Figure 11 refers to *(Addone, Pick Major ranked)*. In these plots, the *x* axis is the weight of file size and the *y* axis is the prediction accuracy.

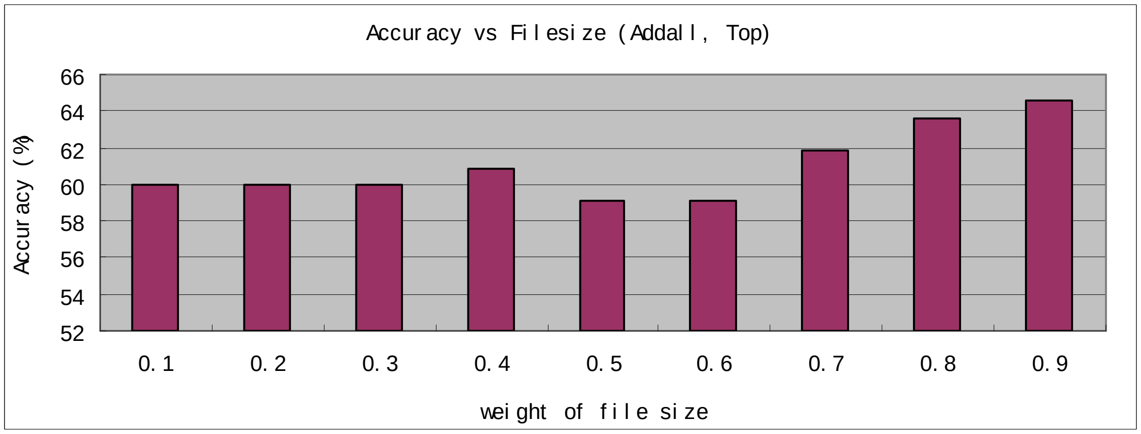


Figure 8 Results of weight of file size experiment for the case (Addall, Top).

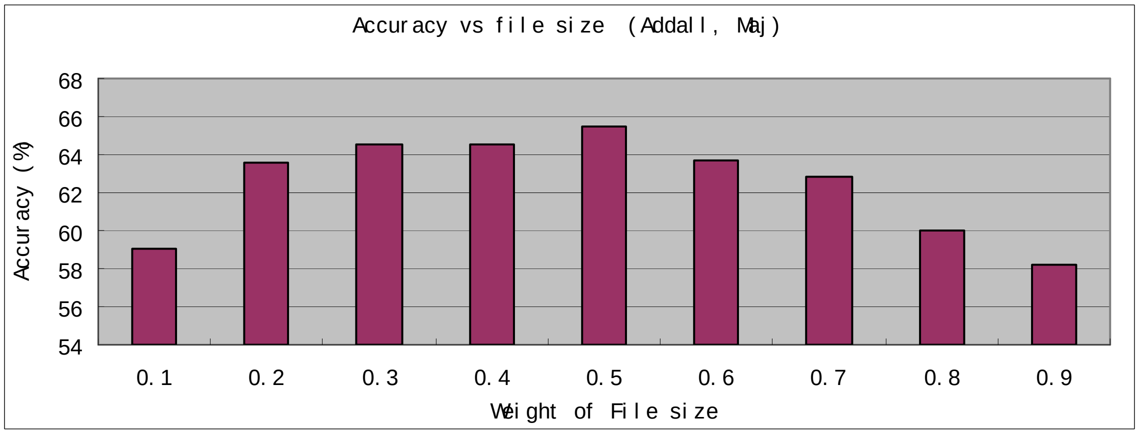


Figure 9 Results of weight of file size experiment for the case (Addall, Maj).

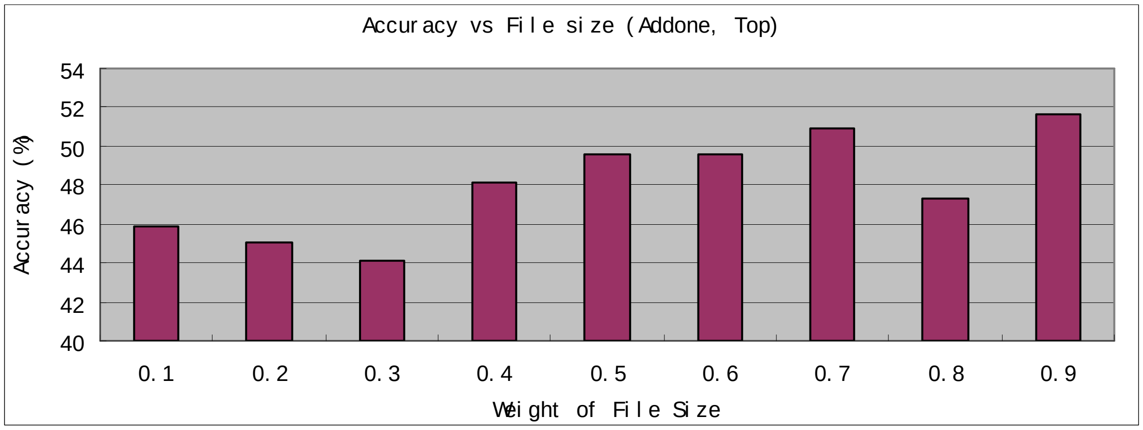


Figure 10 Results of weight of file size experiment for the case (Addone, Top).

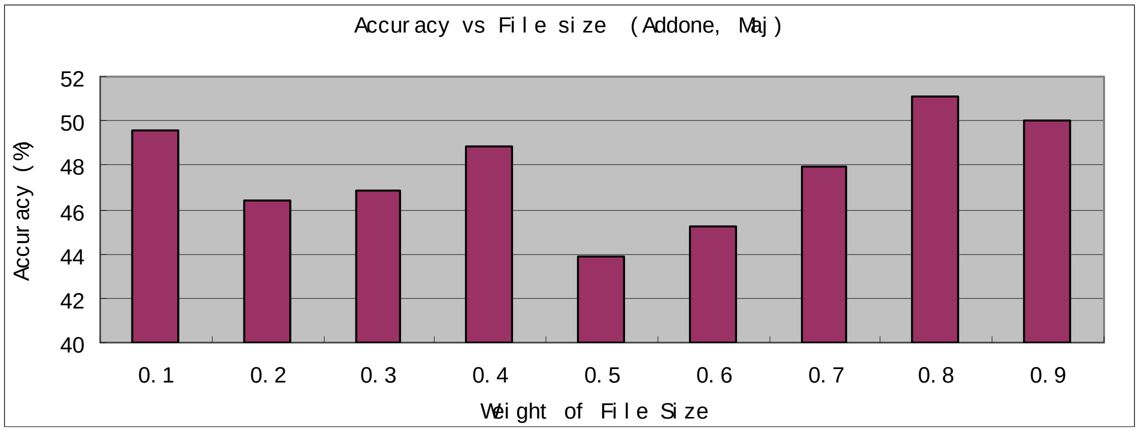


Figure 11 Results of weight of file size experiment for the case (Addone, Maj).

In each plot, the accuracy reaches its maximum when the weight of file size equals to a specific value (0.9 in Figure 8, 0.5 in Figure 9, 0.9 in Figure 10, and 0.8 in Figure 11). According to the metric of this experiment (choose the value that achieves best accuracy as the good value), we therefore chose these four values as the good values of the weight of file size. We noted that these values are relatively large, and hence the attribute file size needs to be considered heavily when making a replica selection. We explain this finding as follows: when transferring files of different sizes, the performance is probably dependent on different major factors. Thus it could turn out that distinct servers are best for file transfers of different sizes. For instance, given two servers A and B, A has a smaller latency than B, while B has bigger available bandwidth than A. Then, when transferring a file of small size, A is likely to be better than B because the transfer time is mainly correlated with the latency in this case. On the other hand, when transferring a file of large size, B is likely to be better than A because the transfer time is mainly dependent on the bandwidth. Consequently, when making a replica selection decision, one should pay particular attention to the file size.

In addition, we also find that different cases have different good values of the weights of file size. We explain this result same as in Section 5.3.1.

5.3.3 Weight of the Attribute Day

In this experiment, we examine the weight of day.

Search Approach

Our approach in examining the weight of day was similar to that described in Section 5.3.2. That is, we examining the accuracies of the IBL prediction when the weight of day is assigned as all possible values, and we chose the value that achieved best accuracy. Specifically we carried out the following steps to find the good values of the weight of day: (1) set *K*; (2) set the weight of day as one value, (3) set the remaining two weights as an equal value (1-weight of day)/2; (4) applied the IBL prediction approach on the log data *Run1* and got accuracy; (5) repeat steps 2—4, gradually increasing the day weight (i.e. 0.1, 0.2, …, 0.9).

We ran four sets of tests for the four cases: *(Addall, pick top ranked), (Addall, pick major ranked), (Addone, pick top ranked),* and *(Addone, pick major ranked).* In each set of tests, we assigned *K* as the selected values in Section 5.3.1.

Results and Analyses

We plot the results in Figure 12 to Figure 15. Each figure corresponds one case: Figure 12 refers to *(Addall, Pick Top ranked)*, Figure 13 refers to *(Addall, Pick Major ranked)*, Figure 14 refers to *(Addone, Pick Top ranked)*, and Figure 15 refers to *(Addone, Pick Major ranked).* In these plots, the *x* axis is the weight of day, and the *y* axis is the prediction accuracy.

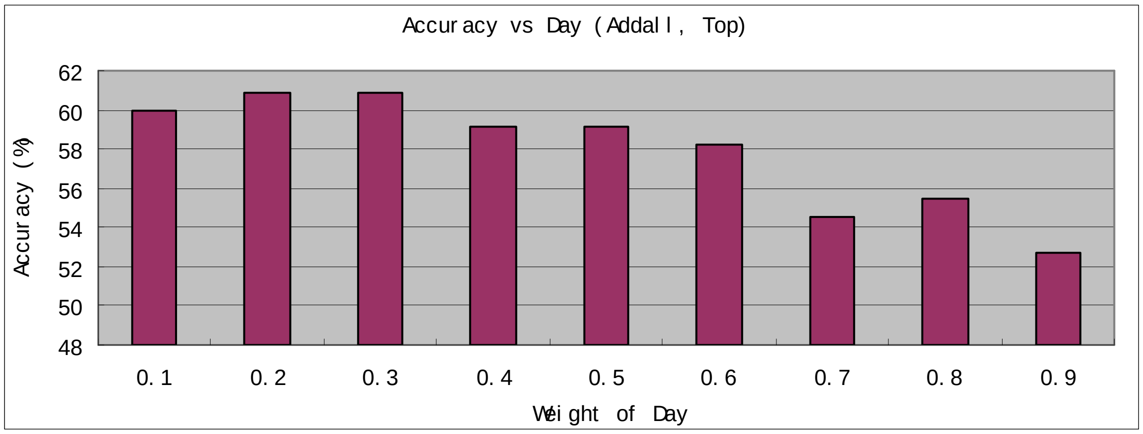


Figure 12 Results of weight of day experiment for the case (Addall, Top).

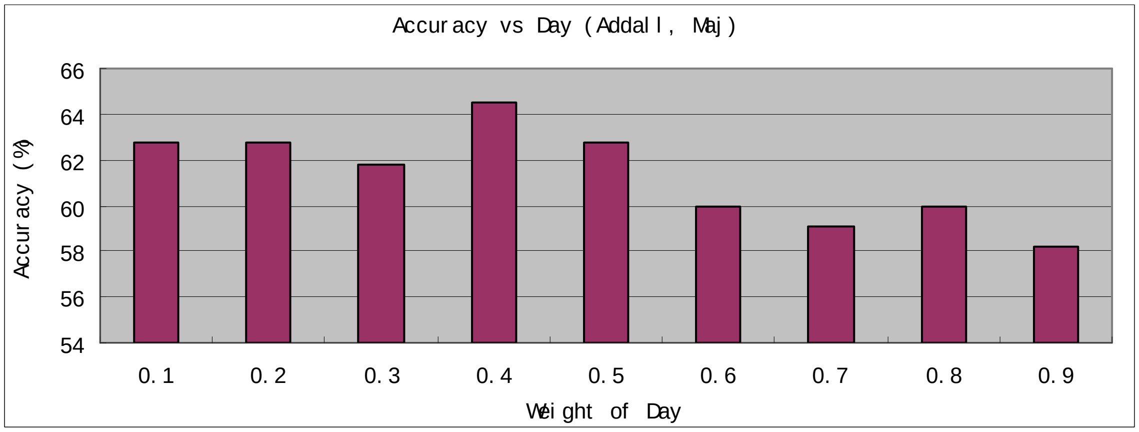


Figure 13 Results of weight of day experiment for the case (Addall, Maj).

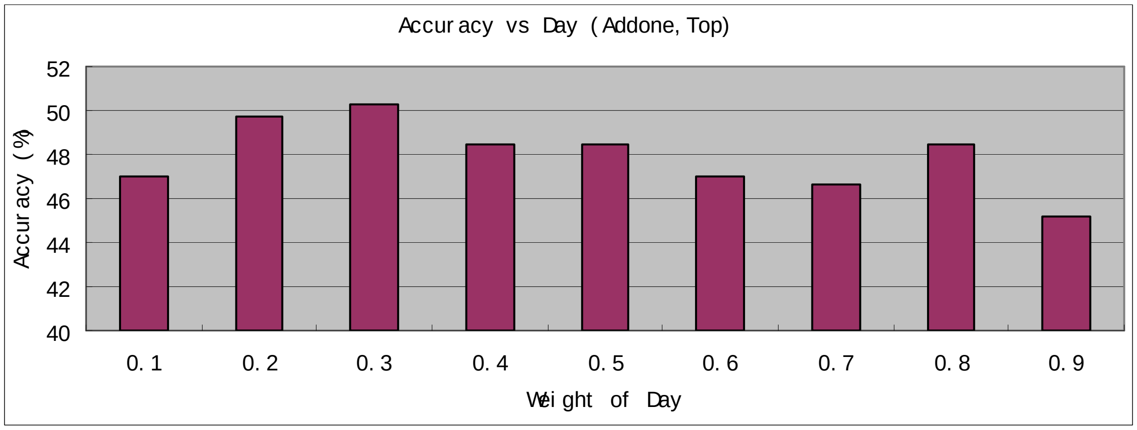


Figure 14 Results of weight of day experiment for the case (Addone, Top).

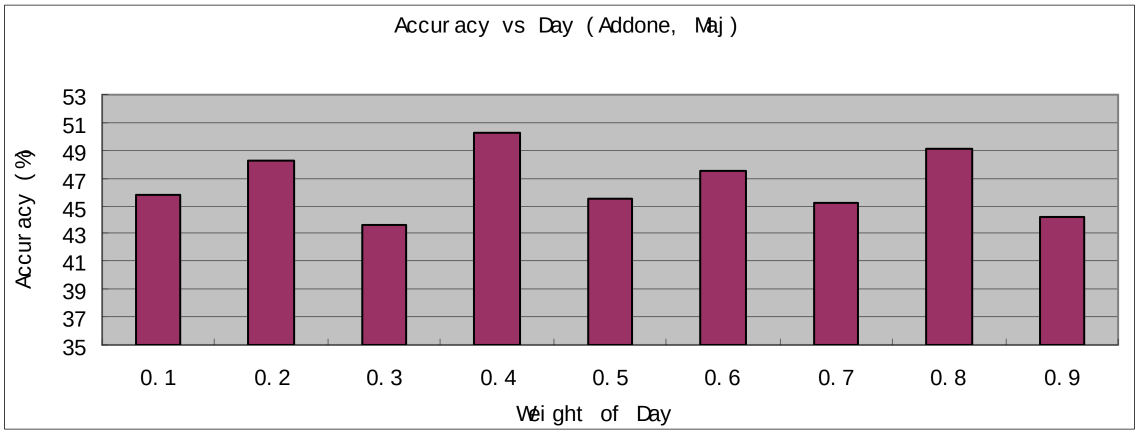


Figure 15 Results of weight of day experiment for the case (Addone, Maj).

Similar to Section 5.3.2, we selected the good values of the weight of day according to the metric "choose the value that achieves best accuracy". Specifically, we choose four values for four cases: 0.25 for the case *(Addall, Pick Top ranked)*, 0.4 for *(Addall, Pick Major ranked)*, 0.3 for *(Addone, Pick Top ranked)*, and 0.4 for *(Addone, Pick Major ranked)*. Actually, in Figure 12, two sequential values (0.2 and 0.3) achieve same best accuracy. Since we think there is no variation in accuracy for all the values in this range [0.2, 0.3], we choose the middle value of the range 0.25 as the good value for that case.

We noted that the four good weights of day are different. The reason is same as that given in Section 5.3.1.

5.3.4 Weight of the Attribute Hour

The next parameter we examine is “the weight of hour”.

Search Approach

We used the search approach described in Section 5.3.2 and Section 5.3.3. That is, we examined the accuracies of the IBL prediction when the weight of hour is assigned as all possible values, and we chose one value that achieved the best accuracy. Specifically we carried out the following steps: (1) set *K*; (2) set the weight of hour as one value, (3) set the remaining two weights as an equal value (1-weight of hour)/2; (4) apply IBL prediction approach on the log data *Run1* and got accuracy; and (5) repeat steps 2—4, gradually increasing the hour weight (i.e. 0.1, 0.2, …, 0.9).

We ran four sets of tests for the four cases: *(Addall, pick top ranked), (Addall, pick major ranked), (Addone, pick top ranked),* and *(Addone, pick major ranked).* In each set of tests, we assigned *K* as the selected values in Section 5.3.1.

Results and Analyses

Four figures show results, each corresponding to one case: Figure 16 refers to *(Addall, Pick Top ranked),* Figure 17 refers to *(Addall, Pick Major ranked),* Figure 18 refers to *(Addone, Pick Top ranked),* and Figure 19 refers to *(Addone, Pick Major ranked).* In these plots, the *x* axis is the weight of hour, and the *y* axis is the prediction accuracy.

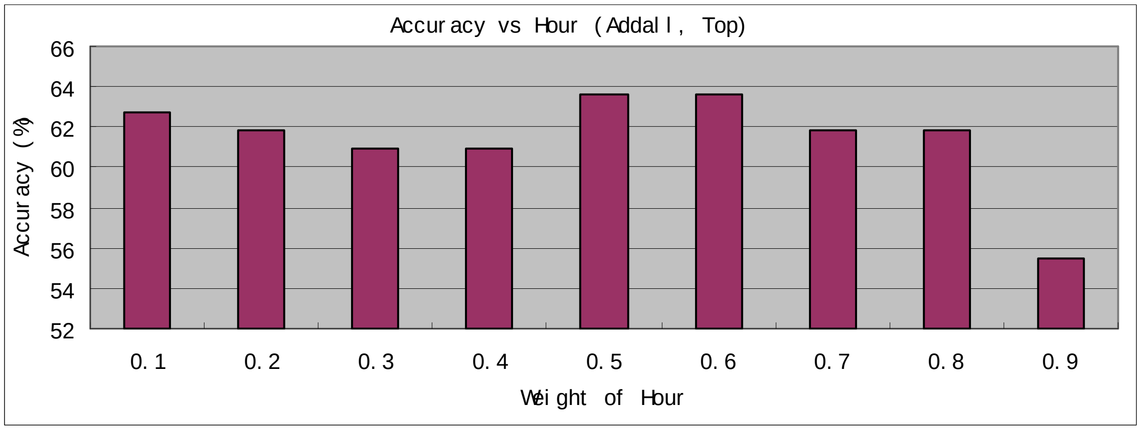


Figure 16 Results of weight of hour experiment for the case (Addall, Top).

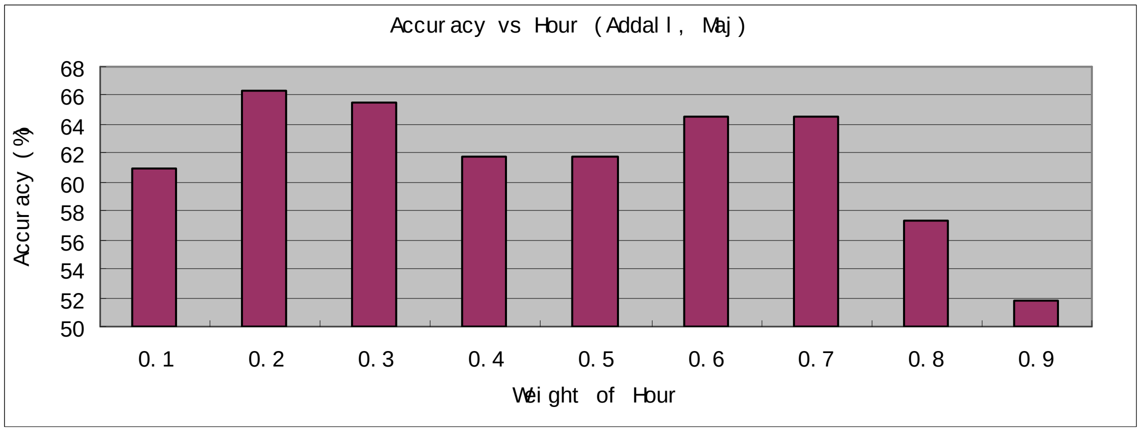


Figure 17 Results of weight of hour experiment for the case (Addall, Maj).

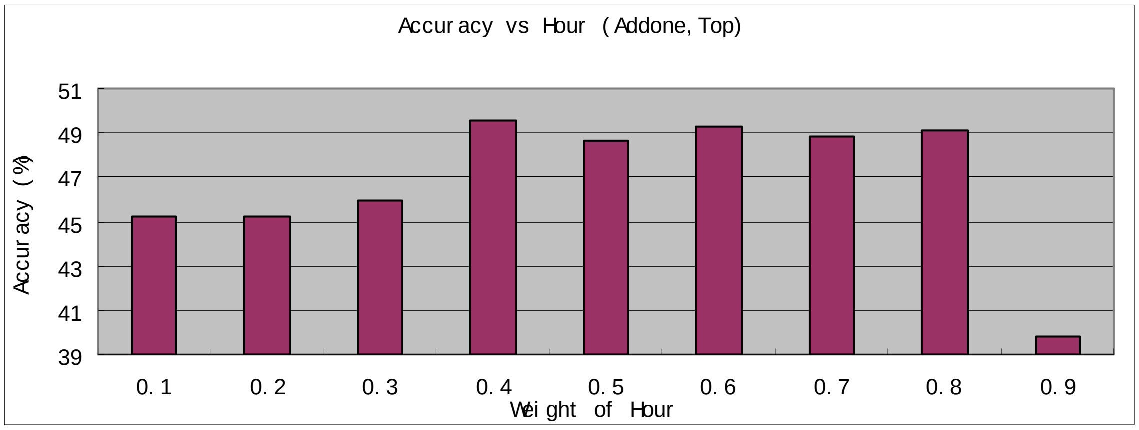


Figure 18 Results of weight of hour experiment for the case (Addone, Top).

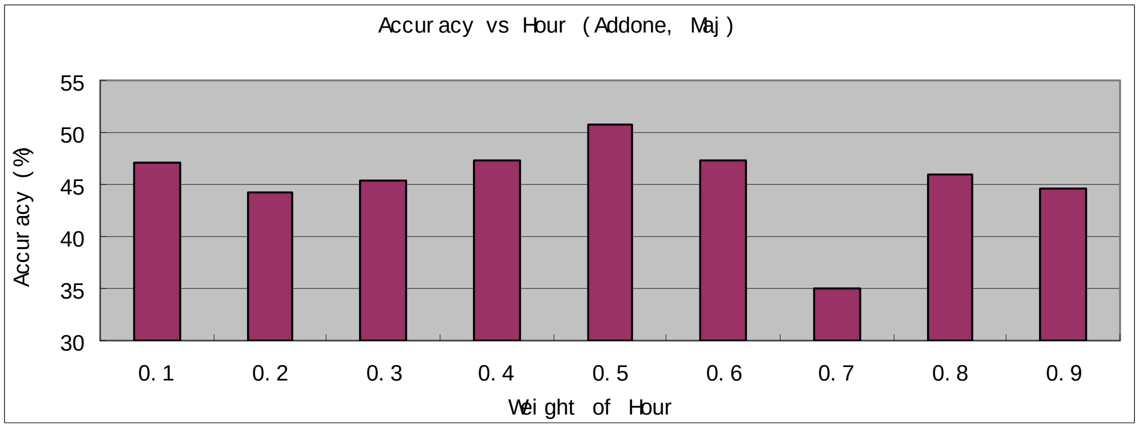


Figure 19 Results of weight of hour experiment for the case (Addone, Maj).

Similar to Section 5.3.2 and Section 5.3.3, we selected the good values of the weight of hour as follows: 0.55 for the case *(Addall, Pick Top ranked)*, 0.2 for *(Addall, Pick Major ranked)*, 0.4 for *(Addone, Pick Top ranked)*, and 0.5 for *(Addone, Pick Major ranked)*. Actually, in Figure 16, two sequential values (0.5 and 0.6) achieve same best accuracy. Similar to Section 5.3.3, we choose the middle value of the range 0.55 as the good value for that case.

We also noted that the four good values are different; the reason is the same as that given in Section 5.3.1.

5.3.5 Set of Parameters

In Sections 5.3.1 to 5.3.4, we evaluated the good values for each parameter. When these values were incorporated into the set of parameters, the sum of the three weights was not equal to 1 (which is not satisfied with the definition in Section 4.1.3). This result is reasonable, however, because when we search the good value for each weight, we don't consider the sum constraint (sum of the good values of three weights should equal to 1). However, since the sum constraint must be satisfied when incorporating these weights, we can't combine them directly. Therefore, we need to figure out a way to incorporate these weights. This is the problem that we now address.

One simple approach is to normalize the three weights proportional to their good values. For example, suppose the found good values are 0.9 (weight of file size), 0.3 (weight of day) and 0.6 (weight of hour), then we can incorporate them into the set *(0.5, 0.17, 0.34)*. We call this set as *Ratio set*.

Besides the *Ratio set*, other choices exist. One is to assign the *weight of file size* as its good value and then assign the remaining two features (*weight of day* and *weight of hour*) proportional to their good values. For instance, suppose the found good values are 0.9 (weight of file size), 0.3 (weight of day), and 0.6 (weight of hour). First, we can assign the *weight of file size* as 0.9, and then we assign the feature *weight of day* as *(1-0.9)*(1/ (1+2)*) *=0.03*, and the feature *weight of hour* as *(1-0.9)*(2/ (1+2)*) *=0.07*. We call this set as *FS set*.

Similarly, we can define a *Day set* (assign the feature *weight of day* as its good value) and an *Hour set* (assign the feature *weight of hour* as its good value).

With the four candidate sets (*Ratio set*, *FS set*, *Day set*, and *Hour set*), we applied the IBL algorithm to *Run1* and got four accuracies. We then compared these four accuracies and chose one set that attained the best accuracy.

We then ran four experiments for the four cases: *(Addall, pick top ranked), (Addall, pick major ranked), (Addone, pick top ranked),* and *(Addone, pick major ranked).*

In Table 6, we list the four candidate sets (*Ratio Set*, *FS set*, *Day set, Hour set*) for each case. In Figure 20 to Figure 23, we show the accuracies of the four candidate sets. In these plots, the *x* axis refers to the four sets that are defined in Table 6, and the *y* axis refers to the prediction accuracy.

Table 6 Candidate sets of Parameters for four cases.

Case	Ks	Candidate Sets
(Addall, Pick Top ranked)	15	FS set: (0.9, 0.03, 0.07), Day set: (0.47, 0.25, 0.28) Hour set:(0.35, 0.1, 0.55), Ratio set: (0.53, 0.15, 0.32)
(Addall, Pick Maj ranked)	20	FS set:(0.5, 0.33, 0.17), Day set: (0.43, 0.4, 0.17) Hour set:(0.44, 0.36, 0.2), Ratio set: (0.45, 0.36, 0.19)
(Addone, Pick Top ranked)	35	FS set: (0.9, 0.04, 0.06), Day set: (0.48, 0.3, 0.22) Hour set: (0.45, 0.15, 0.4), Ratio set: (0.56, 0.19, 0.25)
(Addone, Pick Maj ranked)	35	FS set: (0.8, 0.09, 0.11), Day set: (0.37, 0.4, 0.23) Hour set: (0.33, 0.17, 0.5), Ratio set: (0.47, 0.24, 0.29)

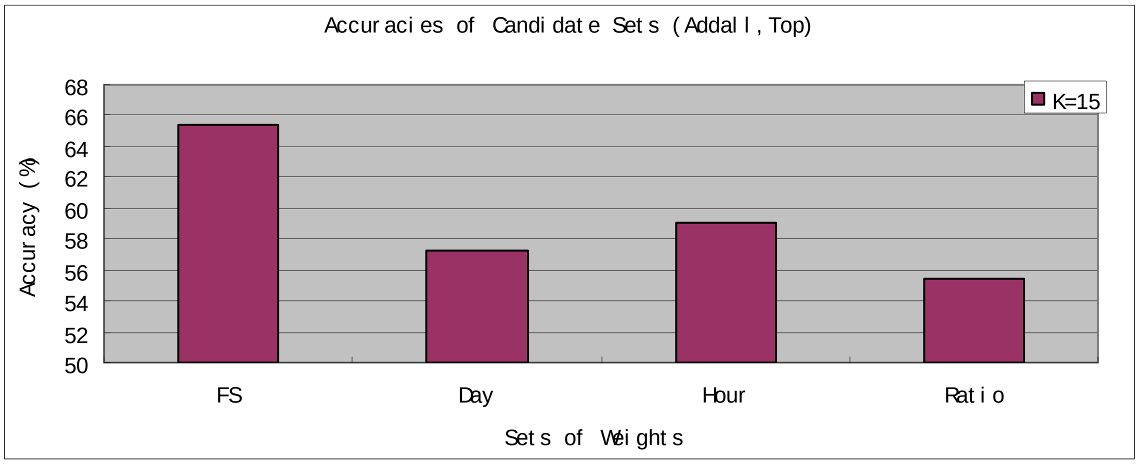


Figure 20 Accuracies of four candidate sets for the case (Addall, Top).

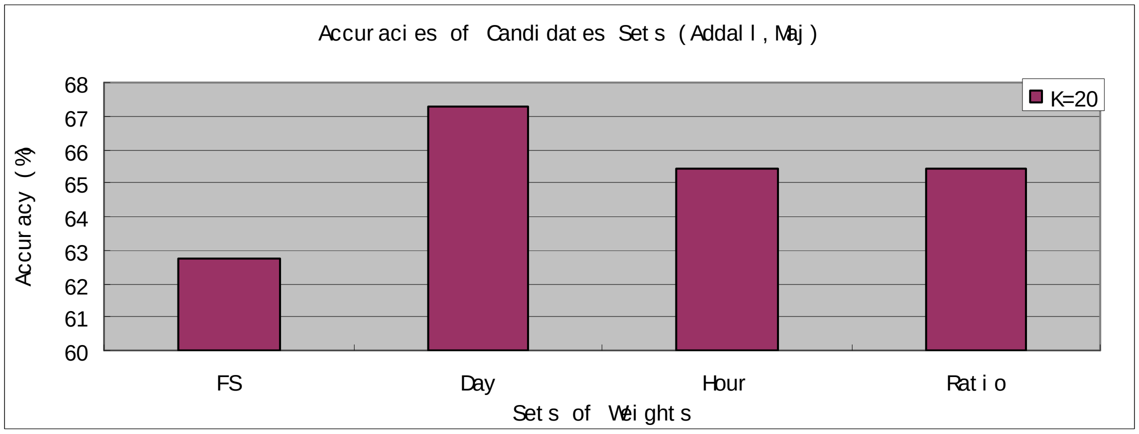


Figure 21 Accuracies of four candidate sets for the case (Addall, Maj)

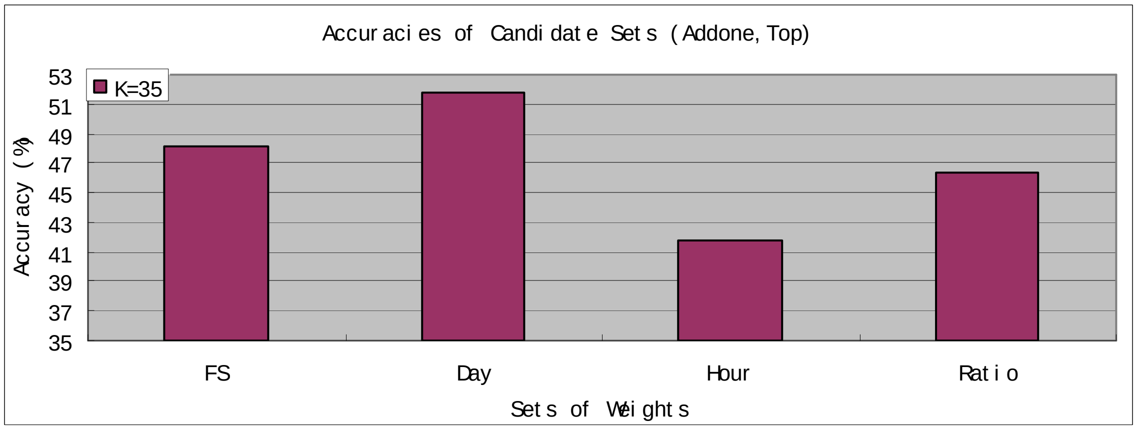


Figure 22 Accuracies of four candidate sets for the case (Addone, Top)

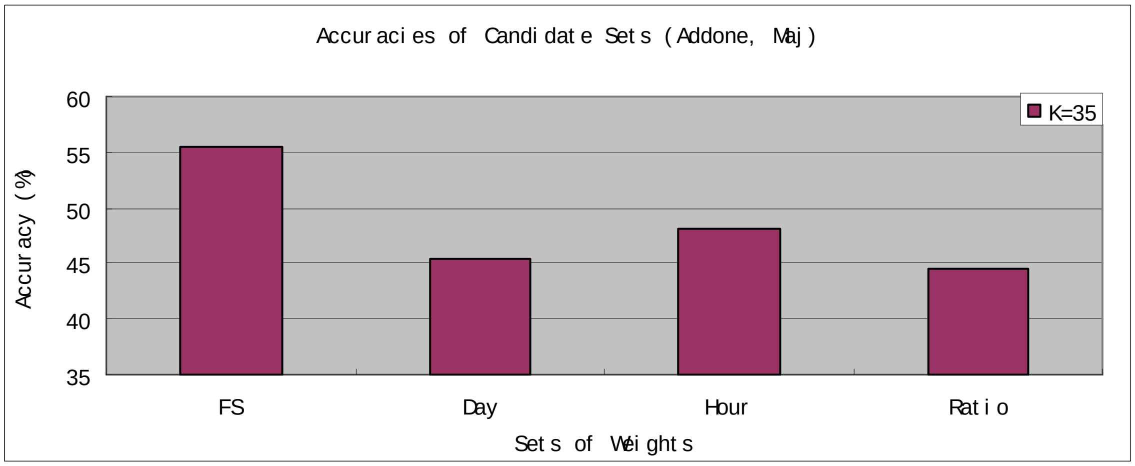


Figure 23 Accuracies of four candidate sets for the case (Addone, Maj)

From these plots, we pick out the proper set of weights for each case and list them in

. With these chosen parameters, we can define four choices for our IBL algorithm. The approach *AllTop* refers to the IBL algorithm with parameters: three weights (0.9, 0.03, 0.07), *K* (15), target function (*pick top ranked*), and update strategy (*Addall*). Similarly, the approaches *AllMaj*, *OneTop* and *OneMaj* are defined and listed in

.

Table 7 Decided Sets of Parameters and their corresponding approaches.

Case	Decided *Ks*	Decided set	IBL Approach
(Addall, Pick Top ranked)	15	FS set: (0.9, 0.03, 0.07)	*AllTop*
(Addall, Pick Maj ranked)	20	Day set: (0.43, 0.4, 0.17)	*AllMaj*
(Addone, Pick Top ranked)	35	Day set: (0.48, 0.3, 0.22)	*OneTop*
(Addone, Pick Maj ranked)	35	FS set: (0.8, 0.09, 0.11)	*OneMaj*

5.4 Experiment 3: Comparison with the Average Approach

In our next experiment, we compared our IBL approach with the so-called *average approach,* on two sets of log data files. These two sets of log files, *Run2* and *Run3*, were collected by using the method described in Section 5.1.2. Experiments show that the IBL approach achieves better accuracy than does the average approach on both data and that its overhead (time to make a prediction) is acceptable. These promising results verify that the IBL approach can be an efficient way for replica selection when only limited data sources are available.

5.4.1 Average Approach

When limited data (e.g. only GridFTP log files) is available, the average approach is a simple and common method for replica selection: it computes the average throughput of each server in historical file transfers, and it selects the server, which has fastest historical average throughput, as the best replica server. For example, with log files of historical file transfers (t_1 to t_n), we can calculate the average historical throughput of server A as follows:

$$\text{Average throughput of Server A} = \sum_{i=1}^{n} ServerAthroughputIn(t_i)/n\,.$$

After the average historical throughputs of all servers are computed, we can choose the server that has the fastest average historical throughput. Specifically, we can apply the *average approach* on the collected log data according to the following steps: (1) sort the log files into an ordered list by the beginning time of their corresponding file transfers; (2) init the history database to be empty; (3) take the head log file from the sorted list; (4) with all log files in the history database, compute the average throughput of each server and choose the server that has fastest throughput as best server; (5) compare the chosen best server with the real best server, which can be found with information of the head log file, and check the correctness of this prediction; (6) update the history database using some update strategy; (7) repeat steps 3–6 until all log files are handled; and (8) calculate the prediction accuracy (*accuracy = number of correct predictions /total number of predictions*) and the overhead *(overhead = total time to make all the predictions/number* *of predictions*).

Since there are two possible update strategies (*Addall* and *Addone*), we have two choices for the average approach: *average-all* and *average-one*. The *average-all* corresponds to the *average* approach with *Addall* as update strategy, while the *average-one* refers to the one with *Addone* as update strategy.

5.4.2 Comparison of Methods

To compare our IBL approach with the average approach, we collected two sets of GridFTP log files using the method described in Section 5.1.2. We call them *Run2* and *Run3*. We applied the four approaches of our IBL algorithm (*AllTop* to *OneMaj*) to both log data (*Run2* and *Run3*), taking almost the same steps as in Section 5.1.2 except that we made additional accuracy measurements: instead of measuring the accuracy after all 110 replica selection decisions, we measured the accuracy after every 10 decisions. Consequently, eleven measurements of accuracy were output. Finally, the overhead was output.

We also applied the two *average* approaches (*average-all* and *average-one*) on the two log data using the method described in Section 5.4.1. Similarly, we modified the accuracy measurement: eleven accuracies were reported.

5.4.3 Results and Analyses

Experimental results are reported in Figure 24 and Figure 25 and in Table 8. In the two figures, the accuracies of six approaches on the two data files (*Run2* and *Run3*) are shown. In each figure, the *x* axis is the number of replica selection decisions made, the *y* axis is the prediction accuracy, and the six lines represent six different approaches (*average-all*, *average-one*, *AllTop*, *AllMaj*, *OneTop*, and *OneMaj*). In addition, in Table 8, the overhead is listed.

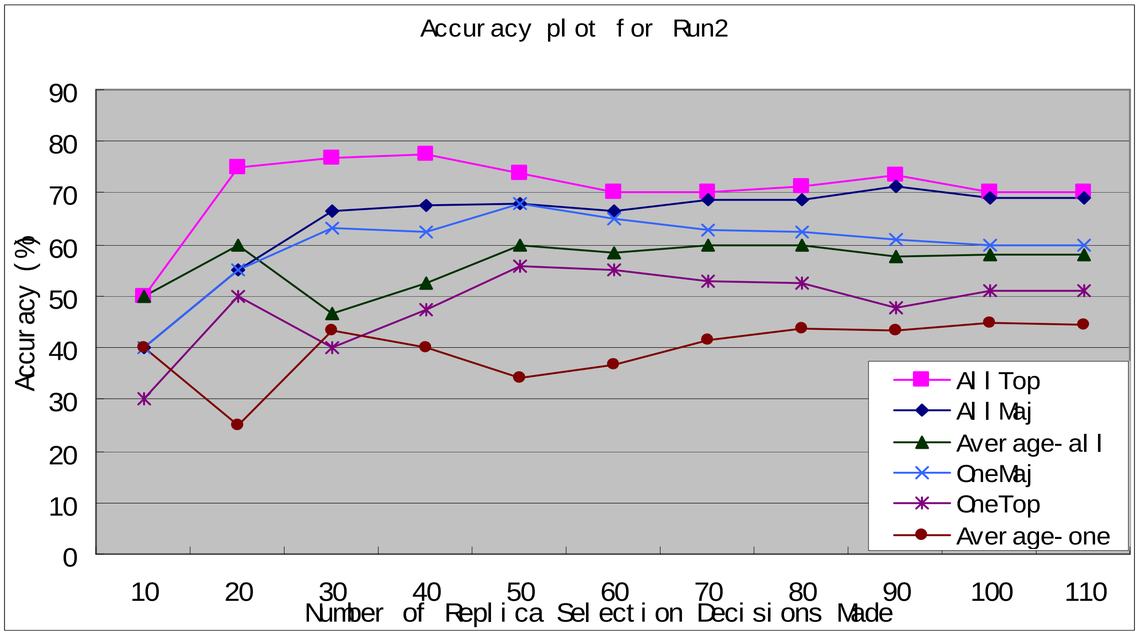


Figure 24 Accuracies of six approaches on data Run2.

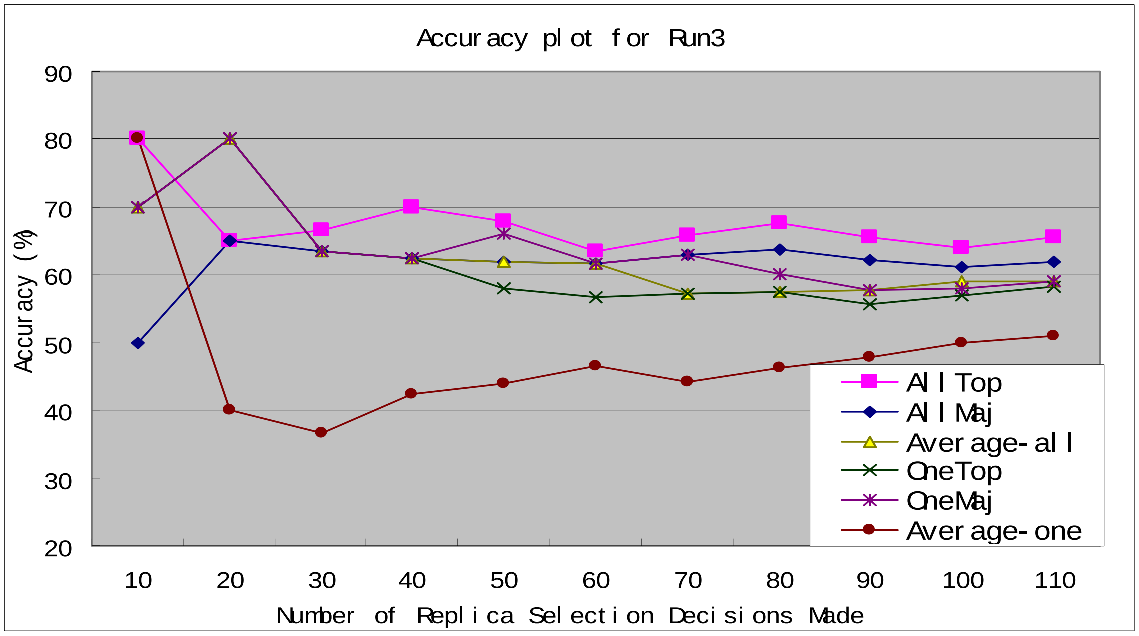


Figure 25 Accuracies of six approaches on data Run3.

Table 8 Overhead of six approaches on two log data Run2 and Run3.

Comparisons of Overhead		
	Run2	Run3
Average-All	0.09 s	0.11s
Average-One	0.09s	0.09s
AllTop	1.44 s	1.69 s
AllMaj	1.44 s	1.75 s
OneTop	0.1 s	0.14 s
OneMaj	0.12 s	0.11 s

From Figure 24 and Figure 25, we observe that the accuracy of each approach varies substantially for the first 30 decisions, and then stays almost constant. After the accuracies become stable, *AllTop* and *AllMaj* are always better (at least 10% for *Run2* and 5% for *Run3* higher) than *average-all*, whereas *OneTop* and *OneMaj* attain higher accuracy (at least 5% for *Run2* and 10% for *Run3* higher*)* than *average -one*.

We explain the first finding as follows. For the first 30 replica selection decisions, the history database stores very little historical information, a situation that can result in inaccurate predictions and hence yield high variations in accuracy. After a number of decisions are made, however, the history database has stored enough information to enables these approaches to make stable prediction decisions and keep the accuracy almost constant.

As for the second finding, we think the reason is that since our IBL algorithm exploits the similarity property (similar file transfer can have similar behavior) to extract more accurate information, our IBL approach can attain higher accuracies than does the average approach.

In addition, from the second finding we see that for both the update strategies (*Addall* and *Addone*), our IBL approach is always better than the average approach. When the update strategy is *Addall*, *AllTop* and *AllMaj* exceed *average-all* by 5% to 10%; when the update strategy is *Addone*, *OneTop* and *OneMaj* exceed *average-one* by 5% to 10%. As noted in Section 5.1.3, the two update strategies correspond to two different circumstances; hence, we can conclude that our IBL approach can attain better accuracy than does the average approach in both ideal and practical circumstances.

Furthermore, in Table 8, another phenomenon is obvious: the overhead of the average approach is smaller than that of our IBL approach. For instance, for data *Run2*, our IBL algorithm takes 0.1 to 1.44 second; the *average approach* takes only about 0.1 second. However, since data-intensive Grid applications always involve transfers of huge datasets (e.g., terabytes), the file transfer time in these Grid applications is usually long (e.g., in our test bed the file transfer time of 1GB is about 300s to 1300s). Compared with these long file transfer times, the overhead of our IBL algorithm (1–2s) is acceptable.

We also see that the accuracy of our IBL approach is not very high (at most 65% to 70%). The reason is that the selection of a replica depends on more factors (e.g., replica servers' CPU load, network status) than what we consider in our IBL algorithm. By neglecting these factors, our IBL algorithm loses prediction accuracy. When only a limited data sources are available, however, we don't have information on those factors for the replica selection. So, in such cases, the accuracy attained by our IBL algorithm is acceptable.

6 Conclusions and Future Works

In this paper, we proposed a lightweight IBL approach for the replica selection in a practical circumstance in which only limited data sources are available. In this approach, we applied the IBL technique to make a replica selection based only on the GridFTP log files. To validate our IBL approach, we collected some GridFTP log files in a real Grid environment. With these log data files, we conducted some search experiments to evaluate the good values of the parameters for our IBL algorithm. Finally, we compared the IBL approach with a simple and common replica selection approach known as the average approach. Experimental results demonstrate that our IBL approach can be an efficient tool for replica selection in data-intensive Grid applications when only limited data is available.

Since this work is at an early stage, there are some aspects that can be improved. First, we will conduct more experiments to verify the accuracy and tolerance of the parameters we find. Second, instead of conducting numerous search experiments, we will exploit some statistical tools to analyze log data and evaluate the parameters. Third, besides the two simple target functions (*pick top ranked* and *pick major ranked*), we plan to apply more sophisticated target functions to attain better accuracy.

Acknowledgements

We gratefully acknowledge much help from Christopher Paul Beckmann and Yong Zhao. We also appreciate the support and advice from Sudharshan Vazhkudai, Xuehai Zhang and other colleagues.

Reference

[1]. Ann Chervenak, Ian Foster, Carl Kesselman, Charles Salisbury, Steven Tueck. The Data Grid: Towards an Architecture for the Distributed Management and Analysis of Large Scientific Datasets. Journal of Network and Computer Applications, Page 187-200, 2001.

[2]. GriPhyN – Grid Physics Network. http://www.griphyn.org/.

[3]. Particle Physics Data Grid. http://www.ppdg.net/.

[4]. The Data Grid Project. http://www.eu-datagrid.org/.

[5]. Taiwan Knowledge Innovation Grid. http://motif.nchc.gov.tw/DataGrid/.

[6]. Kavitha Ranganathan and Ian Foster. Identifying Dynamic Replication Strategies for a High-Performance Data Grid. Proceedings of The International Grid Computing Workshop (Grid'01), Page 75 – 86,2001.

[7]. Sudharshan Vazhkudai , Steven Tuecke , and Ian Foster. Replica Selection in the Globus Data Grid. Proceedings of the International Workshop on Data Models and Databases on Clusters and the Grid (DataGrid 2001), Page 106, 2001.

[8]. Sudharshan Vazhkudai, Jennifer M. Schopf. Using Disk Throughput Data in Predictions of End-to-End Grid Data Transfer. Proceedings of Grid 2002, Page 291-304, 2002.

[9]. Sudharshan Vazhkudai, Jennifer M. Schopf. Predicting Sporadic Grid Data Transfers. Proceedings of HPDC-11, Page 188, 2002.

[10]. Sudharshan Vazhkudai and Jennifer M. Schopf. Using Regression techniques to predict large Data transfers. The International Journal of High Performance Computing Applications (IJHPCA) special issue on Grid Computing: Infrastructure and Applications, August 2003.

[11]. Faerman, M., A. Su, R. Wolski, and F. Berman. Adaptive Performance Prediction for Distributed Data-Intensive Applications. Proceedings of the 1999 ACM/IEEE conference on Supercomputing (CDROM), 1999.

[12]. Swany, M., and Wolsiki R. Multivariate resource performance forecasting in Network Weather Service. University of California Santa Barbara Computer Science Technical Report 2002-12, 2002.

[13]. Smith, W., I. Foster, and V. Taylor. Predicting Application Run Times Using Historical Information. Proceedings of IPPS/SPDP '98 Workshop on Job Scheduling Strategies for Parallel Processing. 1998.

[14]. Nirav H. Kapadia, José A.B. Fortes, Carla E. Brodley. Predictive Application-Performance Modeling in a Computational Grid Environment. Proceedings of the Eighth IEEE International Symposium on HPDC, 1999.

[15]. Richard Gibbons. A Historical Application Profiler for Use by Parallel Schedulers. Proceedings of the 3rd Workshop on Job Scheduling Strategies for Parallel Processing, 1997.

[16]. Thomas M. Kroeger, Darrell D.E. Long. Predicting File System Actions from Prior Events. In Proceedings of the USENIX 1996 Annual Technical Conference, Page 329-328, 1996.

[17]. A. Chervenak, E. Deelman, C. Kesselman, L. Pearlman, and G. Singh. A Metadata Catalog Service for Data Intensive Applications. GriPhyn Technical Report 2002-11, 2002.

[18]. Ann Chervenak, Ewa Deelman1, Ian Foster, Leanne Guy, Wolfgang Hoschek. Giggle: A Framework for Constructing Scalable Replica Location Services. Proceedings of Supercomputing 2002, 2002.

[19]. Shen, X. and A. Choudhary. A Multi-Storage Resource Architecture and I/O, Performance Prediction for Scientific Computing. Proceedings of 9th IEEE Symposium on HPDC, 2000.

[20]. Jennifer M. Schopf. and F. Berman. Performance Predictions in Production Environments. Proceedings of IPPS/SPDP'98. 1998.

[21]. Geisler, J. and V. Taylor. Performance Coupling: Case Studies for Measuring the Interactions of Kernels in Modern Applications. Proceedings of SPEC Workshop on Performance Evaluation with Realistic Applications 1999, 1999.

[22]. Rich Wolski, Neil T. Spring, Jim Hayes. The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing. The Journal of Future Generation Computer Systems Page 757-768, October 1999.

[23]. B. Tierney, W. Johnston, B. Crowley, G. Hoo, C. Brooks, and D. Gunter. The

Netlogger methodology for high performance distributed systems performance analysis Proceedings of the Seventh IEEE International Symposium on HPDC 7, pages 260-267, 1998.

[24]. The Web100 Project. http://www.web100.org, 2002.

[25]. Chung-Hsing Hsu, Ulrich Kremer. IPERF: A Framework for Automatic Construction of Performance Prediction Models. Proceedings of PFDC"1998, 1998.

[26]. Baru, C., R. Moore, A. Rajasekar, and M. Wan. The SDSC Storage Resource Broker. Proceedings of IBM Centers for Advanced Studies Conference 1998, 1998.

[27]. SARA: The Synthetic Aperture Radar Atlas. http://sara.unile.it/sara/.

[28]. Chris Atkeson and Andrew Moore and Stefan Schaal. Locally Weighted Learning for Control. Journal of AI Review, Page 75-113, 1997.

[29]. Olac Fuentes. Automatic Determination of Stellar Atmospheric Parameters Using Neural Networks and Instance-Based Machine Learning. Journal of Experimental Astronomy, Page 21-31, 2002.

[30]. Orestis Telelis, Panagiotis Stamatopoulos. Combinatorial Optimization through Statistical Instance-Based Learning. Proceedings of ICTAI 2001, page 203, 2001.

[31]. Lattner AD, Herzog, O. Instance-Based Learning and Information Extraction for the Generation of Metadata. Proceedings of I-KNOW '03, Page 472-479, 2003.

[32]. D. W. Aha, D. Kibler, and M. Albert. Instance-based Learning algorithms. Machine Learning, Page 37-66, 1991.

[33]. Janet Kolodner. Case-Based Reasoning. Page 25, 1993.

[34]. Sudharshan Vazhkudai, Jennifer M. Schopf, and Ian Foster. Predicting the Performance of Wide Area Data Transfers. In Proceedings of the 16 Int'l Parallel and Distributed Processing Symposium (IPDPS 2002), 2002.

[35]. H. A. Gvenir, S. Altngvde, I. Uysal, and E. Erel. Bankruptcy Prediction Using Feature Projection Based Classification. Proceedings of SCI/ISAS'99, pages 108-113, 1999.

[36]. C. Faloutsos and K. Lin. FastMap: A Fast Algorithm for Indexing, Data Mining and Visualization of Tranditional and Multimedia Datasets. Proceedings of ACM SIGMOD, pages 163-174, 1995. s

[37]. Automatic Image Indexing and Retrieval.http://www.engin.umd.umich.edu/ceep/tech_day/Archives/2000/reports/ECEreport2/ECEreport2.htm.