Predicting Indexer Performance in a Distributed Digital Library

Naomi Dushay
Dept. of Computer Science
Cornell University
Ithaca, NY 14853-7501
naomi@cs.cornell.edu

James C. French
Dept. of Computer Science
University of Virginia
Charlottesville, VA 22903
french@cs.virginia.edu

Carl Lagoze
Dept. of Computer Science
Cornell University
Ithaca, NY 14853-7501
lagoze@cs.cornell.edu

Abstract

Resource discovery in a distributed digital library poses many challenges, one of which is how to choose search engines for query distribution, given a query and a set of search engines. This paper focuses on search engine performance as a criterion for search engine selection and defines two measurements of search engine performance: *availability* – will the search engine respond within a time limit and *response time* – how quickly will the search engine respond, given that it responds at all. We predicted both of these performance characteristics with a variety of algorithms, all of which required little computation time and combined past performance data for each search engine into a succinct record. We used operational data from the NCSTRL distributed digital library to make and evaluate predictions, and we found that simple prediction methods performed as well as more complex methods and that prediction accuracy was closely related to data consistency.

1 Introduction

It has been said that the Internet, and the wide range of products and services made available there, has created a culture of instant gratification among networked computer users. In particular, Internet users expect swift and accurate responses to their search requests, but it can be difficult to fulfill these expectations in the rapidly expanding World Wide Web. Virtually all Web resource discovery tools are based on a centralized architecture, in which a central service creates and deploys a master index, possibly replicated for localized network access. While the utility of this architecture has been proven, there are inherent constraints to the centralized approach, including scalability, lack of domain specificity and intellectual property restrictions [14].

One approach offering promising solutions to these problems is distributed searching, in which query processing is distributed among a set of decentralized search engines. Rights management issues can be addressed via licensing agreements pertaining to specific search engines or sets of documents. Individual search engines can cater to the unique needs of a collection or a user community. Also, queries can be processed in parallel, reducing scalability issues.

A key question for distributed searching is: *given a query and a set of search engines, which ones should be selected?* Selection can be based on information content of the search engines and other factors including load, cost, licensing agreements, network latency and server reliability. These selection criteria require sophisticated mechanisms to work properly and reliably. For example, there may be tradeoffs between selection criteria, or search engines may have disparate methods for determining whether or not to grant access privileges. Once the full set of useful servers has been identified, we would like to select the most reliable and fastest of them to process our query.

In this paper, we focus on search engine performance with respect to query distribution: in particular, we tried to predict search engine performance. Accurate performance predictions could be used to choose among search engines that index the same information, thereby reducing the time a user waits for search results.

The paper is structured as follows. Section 2 describes the logical components of distributed searching architecture and our approach to the query distribution problem. Section 3 describes the algorithms used to make predictions of search engine performance. Section 4 presents the efficacy of our predictions regarding whether or not a search engine will respond before a time limit. Section 5 presents the efficacy of our predictions regarding how quickly a search engine will respond, given that it responds before the time limit. Concluding remarks are presented in section 6.

2 An Approach to Distributed Searching

We have been investigating distributed resource discovery issues in the broader context of federated digital library architecture [17]. This architecture builds digital libraries from distinct sets of individual services, each with defined functionality and the ability to communicate among themselves and to each other using defined protocols. The advantages of this architecture include scalability, easy extension of functionality, and its support of semi-autonomous management of distributed services: participants retain autonomy over their components while using the common protocol to communicate with other services in the digital library.

Three of the services in this model are: *user interfaces*, performing digital library functions pertaining directly to user interaction; *repositories*, which store and access digital documents; and *indexers*, which index information (metadata or full text) for digital documents and process queries on that information.

The information indexed at a particular indexer may be a replica of that at other indexers, may be completely disjoint, or it may overlap the information at other indexers in various ways. How a federated digital library apportions information among indexers depends on a mixture of administrative decisions, rights management issues and fault tolerance concerns. This creates a need for a mechanism to select indexers for query distribution based on content (choosing indexers that have information relevant to the query) as well as other factors such as cost or performance (choosing among multiple indexers indexing the same information).

Other researchers have investigated a variety of issues relevant to distributed searching. For example, the distributed database community has a long history of investigating the optimal distribution of indexing information across LANs and controlled WANs [4]. Research areas in the digital library community include query translation [3], content summarization for query routing [13] [2], content routing [9] [11], collection formation via sharing of metadata and index information [18] and protocols for meta-searching and metadata collection [1] [12]. In addition there have been a number of comparative studies of database selection algorithms [10] [8].

Our research focuses on the selection of indexers with respect to the performance of networks and of the indexers themselves. In [7], we introduced the *query mediator* as a digital library service which functions as an intermediary between user interfaces (UIs) and indexers. Specifically, query mediators (QMs) are responsible for translating queries to indexer protocols, choosing indexers for query processing, routing queries to those indexers, adaptively reacting to operational conditions, and aggregating query results. If a QM performs well, it will rapidly deliver complete search results to the UI. If a QM chooses indexers poorly, or makes poor adaptations to operational conditions, then the digital library user could receive slow or incomplete search results.

In [6], we found that on average QMs spend 44-54% of their time waiting for indexers to respond. We would like to improve the QM mechanism for choosing among overlapped or replicated indexers in order to reduce wait time for indexers, but determining which indexers fit that description is difficult. Indexer reliability and processing speed depend on factors such as indexer hardware characteristics, size of the indexes and current CPU load. In [7], we showed that an indexer's performance does not appear the same to a QM as it appears to the indexer itself. From the perspective of a QM, or the *QM-view*, indexer performance depends on additional factors that are hard to predict, such as network connectivity and network loads. If we could accurately predict QM-view indexer performance, then we could improve the QM's choice of indexers, reducing QM wait time and thus, user wait time.

Our predictions focused on two key aspects of indexer performance from the QM-view:

- ***Availability****: will the indexer respond within a time limit* (such as a search timeout)? Whether or not an indexer responds to a QM is dependent on whether the indexer is currently running and on the network, how long the QM listens for a response, in addition to network conditions, indexer CPU load, etc. If we could accurately predict QM-view indexer availability, then QMs wouldn't waste time waiting for responses from indexers that will never respond.

- ***Response time****: how quickly will the indexer respond, given that it responds at all?* This is the elapsed time between the moment the query is sent from the QM and the moment results are received at the QM from the indexer. If we could accurately predict QM-view indexer response time, then the QM could direct queries to the fastest indexers, and could choose judicious search timeouts as well.

2.1 NCSTRL – a Distributed Digital Library Testbed

The distributed digital library on which we base our research is the Networked Computer Science Technical Reference Library[1] (NCSTRL - pronounced "ancestral"). NCSTRL is an operational digital library employing a distributed, component-based architecture. The NCSTRL collection is globally distributed and made available through the Dienst [15] federated digital library architecture. Dienst is an open architecture and protocol [5] for distributed digital libraries that was developed as part of the DARPA-funded Computer Science Technical Reports Project[2]. These characteristics – global distribution, open interface, and production availability – make NCSTRL an ideal testbed for distributed digital library research (indeed, NCSTRL is one of the collections in the DARPA-funded Distributed Integration Testbed[3]).

The NCSTRL collection consists of institutions, or publishing organizations, each of which (at a minimum) provides a repository of digital documents and descriptive metadata [16] for those documents. These institutions are a combination of Ph.D. granting computer science departments, ePrint repositories, electronic journals, and research institutions. At the time of

[1] http://www.ncstrl.org
[2] http://www.cnri.reston.va.us/cstr.html
[3] http://www.cnri.reston.va.us/integration-testbed.html

publication of this paper, there were over 100 NCSTRL repositories and approximately 50 NCSTRL indexers worldwide.

The Dienst architecture specifies the operational characteristics of semi-autonomous core digital library services, as well as describing an open, extensible protocol for communicating among and with these digital library services. Core services include repositories, indexers and user interface gateways, as well as *collection services* which provide the mechanisms for federating these and other services into a digital library. While not formally defined in Dienst as a separate digital library service, the functionality of the QM is present in NCSTRL.

Each Dienst server, which implements and provides protocol access to a set of services, maintains logs containing operational and statistics messages. We obtained QM-view indexer performance data by analyzing Dienst logs from the following five NCSTRL servers for the period from March 1, 1997 through April 30, 1997:

1. NCSTRL – the home page of NCSTRL, located at Cornell University.

2. CS-TR – the Cornell University Department of Computer Science Dienst server.

3. LITE – the Dienst server at the University of Virginia.

4. BERKELEY – the University of California at Berkeley Dienst server.

5. FORTH – the Institute of Computer Science, Foundation for Research and Technology – Hellas (ICS-FORTH) Dienst server.

Full details of how the logs were analyzed can be found in [6].

3 Algorithms for Predicting Indexer Performance

As mentioned above, our goal is to improve the QM mechanism for choosing among overlapped or replicated indexers. If we could accurately predict indexer performance from the QM-view, then we could optimize the choice of indexers made by the QM, reducing QM wait time and thus, user wait time. QM-view indexer predictions need to address the following questions:

1. *Availability* – will the indexer respond before the search timeout?

2. *Response time* – how quickly will the indexer respond, given that it responds at all?

Answering these predictive questions accurately would be easy if the indexer performance data followed an observable pattern. Unfortunately, we were unable to discern clear overall patterns for indexer availability or response time in our data. Since we couldn't perceive patterns in the data, we applied a variety of predictive methods to QM-view indexer data. All methods we used combined past performance data for a given indexer at a particular QM into a succinct record and required little QM processing time.

Two of the predictive methods we used averaged previous observations:

Running average. The prediction is the average of all previous observations. *A running average* can also be limited the *k* most recent observations; we refer to this as a window of size *k*. In this paper, we are using the *running average* with the maximum window size: *k* is always as large as possible.

Single last observation. The last observation is used as a prediction. This is equivalent to a *running average* with a window size of one.

Our other predictive methods decayed old observations, as we believed that more recent data would be a better predictor of indexer behavior than older data.

Low pass filter. The prediction is the average recent behavior of an indexer. Old observations are decayed exponentially with the following formula:

Equation 1 - low pass filter formula

$$V_n = (m \bullet V_{n-1}) + ((1 - m) \bullet X)$$

V_n is the new value of the *low pass filter* (as well as the prediction)

V_{n-1} is the old value in the filter (before the most recent observation)

X is the most recent observation

m is a weighting parameter between 0 and 1

If m = 0, then the *low pass filter* is the same as the most recent observation – its predictions would be the same as the *single last observation* method. If m = 1, then the filter never changes – all predictions would be the initial filter value, assigned before the first observation.

Ideally, we want to optimize m so that the *low pass filter* predictions are as accurate as possible. We used m = 0.95 for our *low pass filter*, based on the work of Vingralek, Breitbart, Sayal and Scheuermann in [19].

While a *low pass filter* weights recent data more heavily than older data, it has a flaw: the weight on each observation decreases exponentially by the number of observations, rather than by the time elapsed between observations. We addressed this problem with *timed low pass filters*.

3.1 The Timed Low Pass Filter

Since queries are not routed to indexers at regular intervals, we wanted a prediction formula that weighted previous observations according to the freshness of the data: if the most recent observation was very old, we wanted to lessen its weight, and if it was very recent, we wanted to weight it heavily.

Like a *low pass filter,* a *timed low pass filter* gives an average of recent indexer behavior with old observations decayed exponentially, except the weighting depends on the time elapsed between filter updates. *Timed low pass filters* are updated via the following formula:

Equation 2 – timed low pass filter update formula

$$V_n = (m^D \bullet V_{n-1}) + ((1 - m^D) \bullet X)$$

V_n is the new value of the *timed low pass filter* (but not the prediction)

V_{n-1}, X, and m are as defined for Equation 1

D is the elapsed time since the last filter update.

Again, if m = 0, then the *timed low pass filter* is the same as the most recent observation. If m = 1, then the value in the filter never changes.

As D increases, m^D decreases. So as time passes, the old filter value contributes less to the new filter value -- the filter memory decreases.

Note that Equation 2 is for updating the *timed low pass filter*, not for making predictions. That's because the prediction time is not the same as the filter update time. When a QM receives a response from an indexer, it applies Equation 2 to update the value in the *timed low pass filter*. However, when a QM is choosing among overlapped or replicated indexers, it applies one of the formulas in Equation 3 to predict indexer performance using the *timed low pass filter*.

Equation 3 – timed low pass filter prediction formula

Method A: $P = (m^D \bullet Q) + ((1 - m^D) \bullet V)$

or

Method B: $P = (m^D \bullet V) + ((1 - m^D) \bullet Q))$

P is the prediction

V is the *timed low pass filter* value

Q is some predictive value, such as an initial prediction or the *running average*

m is a weighting parameter between 0 and 1 which we determine empirically.

D is the elapsed time since the *timed low pass filter* was last updated.

Our choice of m for the *timed low pass filter* was decided empirically and is illustrated in section 3.3 for our example.

Equation 3 has two prediction formulas, *method A* and *method B*, because we were unsure whether the *timed low pass filter* value or some other value was a better approximation of the "most recent" value when making a *timed low pass filter* prediction. That is, *method B* is very similar to Equation 2: V in *method B* is analogous to V_{n-1} in Equation 2 and Q in *method B* is analogous to X in Equation 2, the "most recent" observation. *Method A* is the same as *method B* with Q and V swapped – in *method A*, we view V as the "most recent" observation.

3.2 Making Predictions – an Example

As we indicated above, we are interested in predicting indexer behavior from the perspective of a QM by using past QM-view indexer performance data; these predictions would inform QM indexer choices when processing queries. Our approach is to simulate predictions for the indexer performance data we gathered from NCSTRL logs in [6] in order to assess which predictive methods are most effective.

Table 1 - Example of QM-view indexer response time data

obs time (in seconds elapsed since time 0)	13	17	19	28	39	45	49	51
obs value (indexer response time in seconds)	3	6	7	11	5	7	3	5

Table 1 contains example indexer response times for a particular indexer from the view of a particular QM. Note that we have the response times of indexers and when these response times were recorded at the QM, but we must extrapolate the following information if we are going to simulate predictions for the observed data:

- *Time of predictions*. The QM needs to make predictions before it chooses indexers and sends them queries, so predictions must occur not only before the observed data is recorded, but before the query was sent to the indexer. In the case of the *timed low pass filter* predictions, the time of the prediction affects the predicted value. We approximated the time of prediction as follows:

  Equation 4 - time of prediction

  $T_p = T_o - R - x$

  T_p is the time of prediction (in seconds elapsed since time 0)

  T_o is the time of the observation (in seconds elapsed since time 0)

  R is the indexer response time (in seconds)

  x is some constant representing the overhead time for the QM to compute predictions, choose indexers, and send the queries to the indexers (in seconds).

  For example, for the first observation in Table 1, T_o is 13 and R is 3. If we let x = 0.7, then the time of prediction for the first observation is (13 – 3 – 0.7), or 9.3.

- *Time of data structure updates*. We assumed the QM updated data structures soon after it recorded indexer performance data; we used a constant to represent the amount of time the QM took between recording indexer performance and updating filters. For example, if the constant was 0.5 seconds, then the data structures would have been updated at 13.5 seconds to incorporate the first observation in Table 1, at 17.5 seconds to incorporate the second observation, etc. The timing of data structure updates affects the values placed in the *low pass filter* and the *timed low pass filter*.

- *Initial values for the data structures*. We set the initial values of all data structures to the mean of the data and in the case of the *low pass filter* and the *timed low pass filter*, set the time of the initial filter updates to zero. The mean of the data is unknown at time zero, but since the methods decay old information quickly, the initial value has little effect on any but the first few predictions.

- *Initial prediction values*. We used the mean of the data for the initial prediction for all methods except the *timed low pass filter* predictions. *Timed low pass filters* compute an initial prediction based on the time of the prediction, the value in the timed low pass filter, the value of Q (for which we used the *running average*) and the value of m (which we set to 0.95 in this example). The mean of the data is unknown at time zero, but all of our prediction methods decay old information quickly enough so that the initial value has little effect on any but the first few predictions.

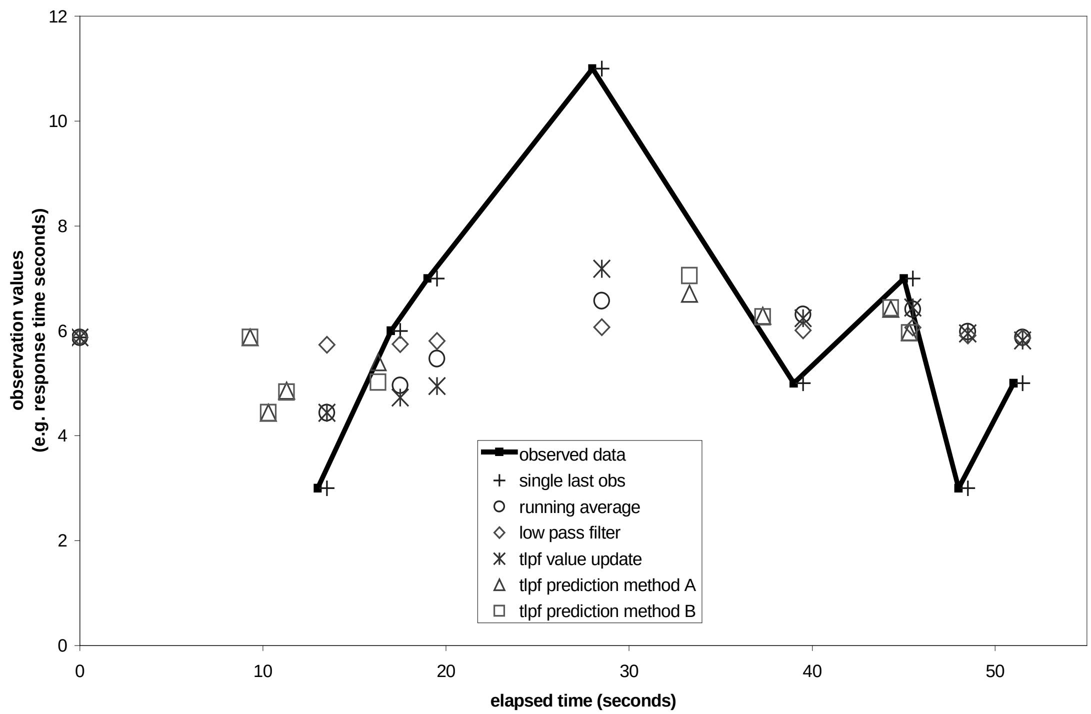

Figure 1 - data structure updates, predictions and observations as they occur in time

Figure 1 shows the observations in Table 1, updates of prediction data structures and *timed low pass filter* (*tlpf*) predictions as they occur in time. Recall that we used the *running average* for Q in the *tlpf* methods (Equation 3), and set m to 0.95. The *single last observation*, the *running average*, the *low pass filter* and the *tlpf* are all initialized at time zero to the mean of the observed data, 5.9 seconds. All four of these are updated at the *time of data structure updates* (see above), or just after the observations are recorded. (In this example, we assumed filter updates occurred 0.5 seconds after observations were recorded.) For example, soon after the observation occurring at time 28, the *single last observation* was updated to the observed value of 11 seconds, the *running average* was updated to 6.6 seconds, the *low pass filter* was updated to 6.1 seconds and the *timed low pass filter* was updated to 7.2 seconds.

The *single last observation*, the *running average* and the *low pass filter* methods all predict the next observation based on whatever value is in them at prediction time. For example, in Figure 1, at time 35 the *single last observation* predicts a response time of 11, the *running average* predicts a response time of 6.6, and the *low pass filter* predicts a response time of 6.1. These predictions will stay the same until these data structures are updated with the next prediction; in our example, the prediction of these methods is the same at time 30 as it is at time 37.

In the case of the *timed low pass filter* predictions, the time of prediction is a factor in the predicted value, and we approximated the prediction time according to Equation 4. For example, in Figure 1, at time 39, there is an observed value of 5, and we have set the prediction time constant x in Equation 4 to 0.7 seconds. So the *tlpf* predictions for the observation occurring at elapsed time 39 are prepared at time (39 – 5 – 0.7) or at elapsed time 33.3.

Note that the *running average*, *low pass filter* and *timed low pass filter value* all smooth the data over time: the highest and lowest observed values are mitigated by these methods.

Since it's hard to tell from Figure 1 which predictions are for which observed data points, we aligned the predictions from all methods with the observed values they were attempting to predict in Figure 2. For example, the *single last observation* method predicted a value of 7.0 for the observation that occurred at time 28; the *running average* method predicted 5.5, the *low pass filter* predicted 5.8, *tlpf method A* predicted 5.4 and *tlpf method B* predicted 5.0.

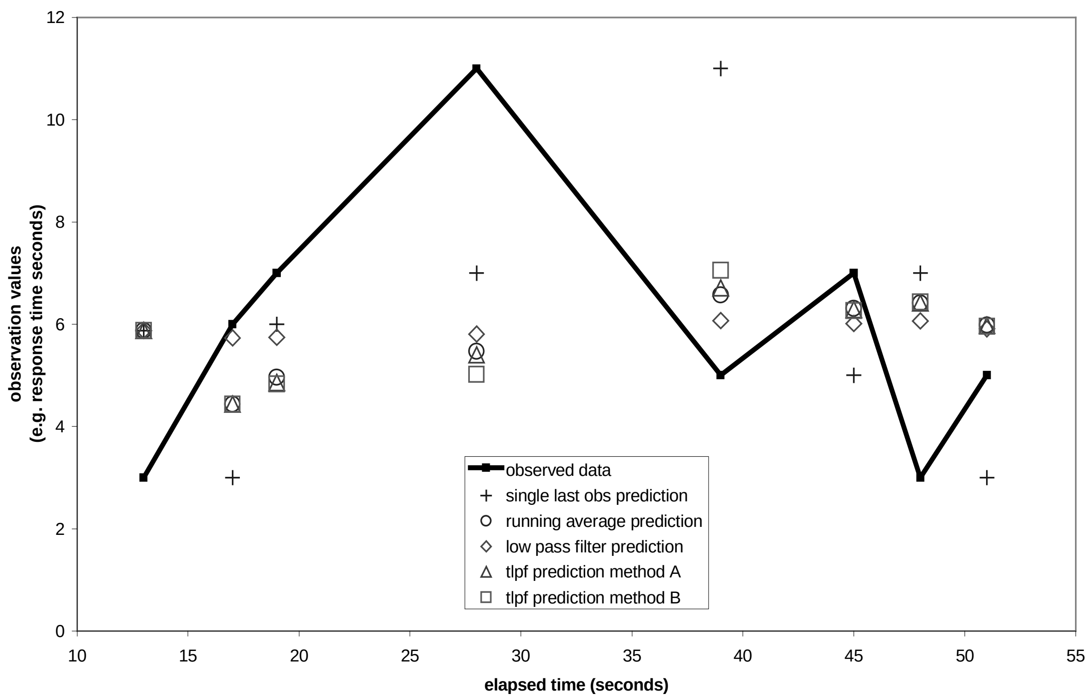

Figure 2 - predictions aligned with observed values

3.3 Tuning the Timed Low Pass Filter

In the example in section 3.2, we used a value of 0.95 for m in the *timed low pass filter* updates (Equation 2) and predictions (Equation 3). We would like to choose a value of m that makes the *tlpf* predictions as accurate as possible. We chose a constant for m empirically for our experiment, rather than by solving Equation 2 and Equation 3 for m using data from [6].

First we predicted indexer behavior with various m values in the *tlpf* equations, and then we evaluated the accuracy of the generated predictions associated with each m value by comparing predictions to observed data using mean square error (MSE). In the case of the example in section 3.2, the m values and their corresponding MSE for *tlpf prediction methods A* and *B* are shown in Table 2.

Table 2 - MSE for tlpf prediction methods A and B for different m values for section 3.2 example

m value	MSE for tlpf prediction method A	method B
0	11.78	7.63
0.1	11.79	7.62
0.2	11.82	7.61
0.3	11.86	7.57
0.4	11.87	7.51
0.5	11.80	7.46
0.6	11.58	7.48
0.7	11.08	7.69
0.8	10.11	8.29
0.9	8.65	9.08
0.99	7.53	6.58
0.999	7.61	5.93
0.9999	7.63	5.87
0.99999	7.63	5.86
0.999999	7.63	5.86
1	7.63	5.86

We can see from Table 2 that in our example, m value 0.99 gives the lowest MSE for *tlpf prediction method A*, while m values 0.99999, 0.999999 and 1.0 all give the lowest MSE of 5.86 for *tlpf prediction method B*. So the m value that minimizes the MSE for *method A* is not necessarily the m value that minimizes MSE for *method B*.

In our example, we predicted behavior for one indexer. But QMs send queries to many indexers, so m values must be chosen for each indexer to which the QM might send a query. In our experiment, we computed the MSE for each m value shown in Table 2 for each indexer contacted by the QM and picked the m value with the smallest MSE to use when comparing *tlpf* methods with other predictive methods. When we combined all indexer predictions at a QM for the *tlpf* methods, we chose the best m value for each indexer: there is not one single m value for a QM, but a separate m value for each indexer contacted by the QM.

It is important to note that our method of choosing m values allows for no cross training: m values are optimized on the same data that we used to make and evaluate predictions.

4 Predictions of Indexer Availability

We've said before that QMs could reduce user wait time if they selected reliable indexers when choosing among overlapped or replicated indexers for query distribution. We also stated that one predictive question QMs need to address is:

Availability: will the indexer respond to this QM before the search timeout?

Availability data was recorded as a binary measurement: either the indexer responded before the timeout (value 1) or it did not (value 0). When we applied our predictive algorithms to this data, they produced a number between 0 and 1, which we then rounded in order to get a predictive value. For example, an availability prediction of 0.3 meant we predicted the indexer would not respond before the search timeout; a prediction of 0.7 meant we predicted the indexer would respond before the timeout.

We predicted indexer availability using all methods delineated in section 3:

1. *single last observation*
2. *running average*
3. *low pass filter* (with m = 0.95, per [19])
4. *timed low pass filter predictive method A*
5. *timed low pass filter predictive method B*.

As in the example in section 3.2, for *timed low pass filter methods A* and *B* we used the *running average* for Q in Equation 3 and we initialized data structures at elapsed time zero to the mean of the observations. We approximated the overhead time for the QM to compute predictions (x in Equation 4) as 2 seconds, and we assumed the data structure updates occurred in the same moment (within one second) that the indexer performance data was recorded in the logs.

In addition to the five methods described in section 3, we also made predictions using:

6. ***the timed low pass filter value itself***. Since we had this information available, we used it as a predictive method, primarily to compare it with the *low pass filter* predictions and the *timed low pass filter* predictive method results.
7. ***the mean of all the data***. By "all the data", we mean all the observations we recorded. Since the *mean of all the data* could not be known at prediction time, we viewed this method as a sort of control or point of reference for our results.

In the remainder of this section, we examine our choice of m for *tlpf* methods (4, 5 and 6 above). We then analyze the accuracy of the availability predictions for all seven methods, determining which methods are superior. Last, we compare the prediction accuracy to the consistency of the availability data itself.

4.1 Tuning the Timed Low Pass Filter

As noted in section 3.3, for the timed low pass filter algorithms, each QM must choose an m value for each indexer to which queries could potentially be distributed. We chose m values empirically: for each QM in our study, we ran all three of the *tlpf* algorithms (methods 4, 5 and 6 above) with 16 particular values for m for each of the indexers the QM contacted.

Table 3 - MSE of tlpf method B availability predictions for CS-TR QM by m value

m value	A	B	C	D	E
0	0.01	0.40	0.12	0.40	0.31
0.1	0.01	0.40	0.12	0.40	0.31
0.2	0.01	0.40	0.12	0.40	0.31
0.3	0.01	0.40	0.12	0.40	0.31
0.4	0.01	0.40	0.12	0.40	0.31
0.5	0.01	0.39	0.12	0.40	0.31
0.6	0.01	0.39	0.12	0.40	0.31
0.7	0.01	0.39	0.12	0.40	0.31
0.8	0.01	0.39	0.11	0.39	0.31
0.9	0.01	0.39	0.11	0.40	0.30
0.99	0.01	0.39	0.12	0.39	0.25
0.999	0.02	0.41	0.15	0.40	0.14
0.9999	0.01	0.56	0.12	0.46	0.17
0.99999	0.99	0.61	0.88	0.56	0.32
0.999999	0.99	0.61	0.88	0.39	0.32
1	0.99	0.61	0.88	0.39	0.68

A = cs-tr.cs.cornell.edu:80
B = cs.nyu.edu:80
C = ncstrl.cc.vt.edu:8080
D = ncstrl.cc.vt.edu:8081
E = www.cc.gatech.edu:81

Table 3 shows the MSE for *tlpf method B* availability predictions for different m values for five indexers contacted by the CS-TR QM. The MSE for the 16 different m values for these five indexers illustrate some of the patterns we saw when choosing "best" m values for a given indexer (for a particular *tlpf* algorithm for a particular QM). For example, indexer E's MSE distribution is unimodal, with the lowest MSE of 0.14 occurring at m value 0.999, while the MSEs shown for indexers C and D are multimodal.

Given information like that in Table 3, we chose the "best" m value – the value, or one of the values, producing the lowest MSE – for each *tlpf* method for each indexer contacted by a QM. We used that "best" m value when comparing *tlpf* predictive methods against the other predictive methods for each indexer contacted by each QM.

When we looked at the MSE for a *tlpf* method for "all indexers" contacted by a QM, we chose the best m value for each indexer. In other words, the *tlpf* method for "all indexers" does not use one single m value for the QM, but a separate m value for each indexer contacted by the QM in this combined data.

4.2 Availability Prediction Results

Predictions are evaluated by comparing them to observations using mean square error (MSE). Since both the availability data and the availability predictions had binary values of 0 and 1, the error for any observation could be 0, 1, or –1. This means that the MSE is the same as the mean of the absolute value of the error: a MSE of .10 implies that one out of ten predictions was incorrect.

Table 4 shows the MSE for each of the availability predictive methods for QM CS-TR. For all but two indexers contacted by the CS-TR QM, any predictive method has a MSE within 0.05 of any other predictive method. For example, indexer *cs.nyu.edu:80* has MSEs ranging from a low of 0.36 for the *tlpf value* method to a high of 0.40 for the *running average* method. The two indexers that have more widely varying MSE values across different predictive methods are *lite.ncstrl.org:3803*, with MSE values ranging from 0.29 to 0.47, and *www.cc.gatech.edu:81*, with MSE values ranging from 0.03 to 0.32. Since results are similar for all predictive methods, we surmise that any pattern, or lack thereof, in the availability data affects all predictive methods similarly. In fact, the *single last observation* method and the *low pass filter* method (with m = 0.95) have identical MSE values for all indexers. Similarly, the *mean of all data* and the *running average* MSE values are equal for all but three indexers (*cs.nyu.edu:80*, *ncstrl.cc.vt.edu:8081*, and *www.cs.utah.edu:80*), for which the *running average* MSE is 0.01 greater than the *mean of all data* MSE.

Table 4 – availability prediction MSE for QM CS-TR

indexer	no. obs	mean of all data	single last obs	running average	low pass filter	timed low pass filter: tlpf value	timed low pass filter: method A	timed low pass filter: method B
cs-tr.cs.cornell.edu:80	14,868	0.01	0.01	0.01	0.01	0.01	0.01	0.01
cs.nyu.edu:80	1,869	0.39	0.37	0.40	0.37	0.36	0.37	0.39
dri.cornell.edu:80	14,172	0.00	0.01	0.00	0.01	0.01	0.00	0.00
ei.cs.vt.edu:8090	13,653	0.01	0.02	0.01	0.02	0.01	0.01	0.01
lite.ncstrl.org:3803	10,370	0.47	0.29	0.47	0.29	0.29	0.29	0.40
ncstrl.cc.vt.edu:8080	16,957	0.12	0.12	0.12	0.12	0.12	0.12	0.11
ncstrl.cc.vt.edu:8081	1,115	0.39	0.39	0.40	0.39	0.39	0.40	0.39
ncstrl.cc.vt.edu:8090	15,824	0.05	0.07	0.05	0.07	0.05	0.05	0.05
ncstrl.cs.cornell.edu:8090	13,057	0.13	0.17	0.13	0.17	0.17	0.13	0.12
www.cc.gatech.edu:81	3,076	0.32	0.03	0.32	0.03	0.03	0.03	0.14
www.cs.dartmouth.edu:80	2,510	0.13	0.15	0.13	0.15	0.13	0.13	0.13
www.cs.uiuc.edu:80	2,785	0.02	0.02	0.02	0.02	0.02	0.02	0.02
www.cs.umass.edu:80	13,993	0.11	0.15	0.11	0.15	0.15	0.11	0.11
www.cs.umd.edu:80	2,322	0.38	0.35	0.38	0.35	0.35	0.35	0.37
www.cs.utah.edu:80	910	0.06	0.06	0.07	0.06	0.06	0.07	0.06
www.icase.edu:80	13,761	0.09	0.14	0.09	0.14	0.14	0.09	0.09
www.ics.forth.gr:7000	5,162	0.11	0.06	0.11	0.06	0.06	0.06	0.10
www.tc.cornell.edu:80	13,988	0.01	0.00	0.01	0.00	0.00	0.00	0.01
all indexers	160,392	0.10	0.10	0.10	0.10	0.09	0.08	0.09

The MSE of availability predictions for the other four QMs in our study are similar to those presented in Table 4; we present the MSE for the combined data of all polled indexers for each of the QMs in Table 5.

Table 5 – MSE for all availability predictions

QM	no. obs	mean of all data	single last obs	running average	low pass filter	timed low pass filter: tlpf value	timed low pass filter: method A	timed low pass filter: method B
cs-tr	160,392	0.10	0.10	0.10	0.10	0.09	0.08	0.09
ncstrl	113,511	0.09	0.09	0.09	0.09	0.09	0.07	0.08
berkeley	69,483	0.13	0.12	0.13	0.12	0.13	0.11	0.11
lite	6,363	0.09	0.07	0.09	0.07	0.07	0.07	0.07
forth	743	0.14	0.13	0.16	0.13	0.13	0.12	0.13

In Table 5, the MSE ranges narrowly, from 0.07 for a variety of prediction methods for the LITE QM to 0.16 for the *running average* method for the FORTH QM. This reinforces our conclusion from Table 4: all availability predictive methods have similar results. In Table 5, the MSE for any method for a given QM is at most 0.04 different from the MSE for any other method for the same QM. We do note that *tlpf methods A* and *B* have a slightly lower MSE than any of the other methods, but the *single last observation* method performs nearly as well and requires no optimization of formula variables (such as m). Moreover, the *single last observation* method is trivial to compute and has minimal start up costs.

Given that the vast majority of the predictions were for the CS-TR and NCSTRL QMs, which had maximum MSE values of 0.10 and 0.09 respectively, we can say that approximately 90% of our indexer availability predictions were accurate regardless of the predictive method used.

As another means of comparing the different predictive methods, we now examine Table 6, showing the highest MSE for each predictive method for an individual indexer at each QM in our study. Note that multiple indexers contacted by each QM may be represented in Table 6: the indexer with the highest MSE for one predictive method may be different from the indexer with the least accurate predictions for a different predictive method as viewed by the same QM.

Table 6 - maximum MSE for QM availability predictions for individual indexers

QM	mean of all data	single last obs	running average	low pass filter	timed low pass filter: tlpf value	timed low pass filter: method A	timed low pass filter: method B
cs-tr	0.47	0.39	0.47	0.39	0.39	0.40	0.40
ncstrl	0.44	0.37	0.45	0.37	0.37	0.37	0.45
berkeley	0.50	0.38	0.50	0.38	0.38	0.38	0.41
lite	0.49	0.28	0.50	0.28	0.29	0.29	0.31
forth	0.36	0.39	0.75	0.39	0.39	0.38	0.75

Table 6 shows that the *single last observation* method has the lowest maximum MSE values for individual indexer predictions at all QMs except FORTH. Again, we see that the *single last observation* method performs identically with the *low pass filter* method, and very similarly to methods *tlpf value* and *tlpf method A*. Table 6 reinforces the notion that the *single last observation* method performs as well as, and sometimes better than, more complex predictive methods such as the *low pass filter* or *timed low pass filter* methods.

We now wish to compare availability prediction accuracy with the data consistency: if an indexer consistently responds (or consistently doesn't respond) to the QM before the search timeout, then we would expect our predictions to be very accurate. Our measurement of data consistency is the availability ratio: the number of times an indexer responded to the QM before the search timeout divided by the number of times the QM attempted to contact the indexer.

Table 7 - single last observation availability prediction MSE compared with availability ratios for indexers contacted by CS-TR QM (sorted by availability ratio)

indexer	no. obs	availability	single last obs MSE
ncstrl.cc.vt.edu:8081	1,115	0.39	0.39
lite.ncstrl.org:3803	10,370	0.53	0.29
cs.nyu.edu:80	1,869	0.61	0.37
www.cs.umd.edu:80	2,322	0.62	0.35
www.cc.gatech.edu:81	3,076	0.68	0.03
ncstrl.cs.cornell.edu:8090	13,057	0.87	0.17
www.cs.dartmouth.edu:80	2,510	0.87	0.15
ncstrl.cc.vt.edu:8080	16,957	0.88	0.12
www.cs.umass.edu:80	13,993	0.89	0.15
www.ics.forth.gr:7000	5,162	0.89	0.06
www.icase.edu:80	13,761	0.91	0.14
www.cs.utah.edu:80	910	0.94	0.06
ncstrl.cc.vt.edu:8090	15,824	0.95	0.07
www.cs.uiuc.edu:80	2,785	0.98	0.02
cs-tr.cs.cornell.edu:80	14,868	0.99	0.01
ei.cs.vt.edu:8090	13,653	0.99	0.02
www.tc.cornell.edu:80	13,988	0.99	0.00
dri.cornell.edu:80	14,172	1.00	0.01
all indexers	160,392	0.89	0.10

Table 7 compares the availability ratio of indexers contacted by the CS-TR QM and the MSE of the *single last observation* prediction method. We see that in general, as the availability ratio increases, the MSE for the *single last observation* decreases. This meets our expectations of highly accurate predictions when indexer behavior is highly consistent – when indexers have availability ratios of 0.98, 0.99, or 1.00, the MSE is 0.02 or less.

Note that due to the nature of the availability measurements (1 if the indexer responded to the QM before the search timeout, 0 if the indexer didn't respond to the QM in time), the mean of the availability data is the same as the availability ratio. If we were comparing the MSE for the *mean of all data* predictive method with the availability ratio, we would expect a perfect inverse relationship between availability and MSE (MSE = 1 – availability ratio) for all indexers responding more than half the time. Likewise, we would expect a direct relationship (MSE = availability ratio) for all indexers with an availability ratio of 0.50 or less. If we compare the *mean of all data* MSE values in Table 4 with the availability ratios in Table 7, we can see that this is true.

In Table 7 there are four indexers for which the MSE is significantly different than we would expect: *lite.ncstrl.org:3803*, *www.cc.gatech.edu:81*, *www.ics.forth.gr:7000*, and *www.icase.edu:80*. The first three indexers are instances in which the *single last observation* predictive method out-performs the *mean of all data* predictive method, while the last instance is one in which the *single last observation* method does worse than the *mean of all data* method.

The comparisons of *single last observation* MSE to availability ratios for the other four QMs in our study are similar to those presented in Table 7. In Table 8 we compare the availability ratios for the combined data of all polled indexers for each QM with the MSE for the *single last observation* method.

Table 8 - single last observation availability prediction MSE and availability ratio of all indexers (sorted by availability ratio)

QM	no obs	availability	single last obs MSE
forth	743	0.64	0.13
berkeley	69,483	0.79	0.12
ncstrl	113,511	0.88	0.09
cs-tr	160,392	0.89	0.10
lite	6,363	0.91	0.07
all QMs	350,492	0.87	0.10

As in Table 7, in Table 8 we see that as the availability ratio increases, the MSE decreases. In other words, as the availability becomes more consistent, the accuracy of our predictions increases.

Table 8 also indicates that the availability ratio for all observations is 0.87: QMs received a response from a polled indexer before the search timeout 87% of the time. We could also say that the consistency of the data is high. (The consistency would be equally high if the availability ratio for all observations was 0.13). This high consistency of the data is the largest factor for the overall accuracy of 90% for our *single last observation* predictions for all indexers polled at all QMs.

5 Predictions of Indexer Response Time

Our goal is to predict QM-view indexer behavior as accurately as possibly so the QM can choose fast, reliable indexers for query distribution when presented with overlapped or replicated indexers. Recall that QM-view indexer predictions not only need to address the availability question (see section 4), but also the following question:

> *Response time*: how quickly will the indexer respond to the QM, given that it responds before the search timeout?

We made QM-view indexer response time predictions made with the same seven predictive methods used to make availability predictions. As with the availability predictions, for *timed low pass filter methods A* and *B* we used the *running average* for Q in Equation 3 and we initialized data structures at elapsed time zero to the mean of the observations. We approximated the overhead time for the QM to compute predictions (x in Equation 4) as 2 seconds, and we assumed the data structure updates occurred in the same moment (within one second) that the indexer performance data was recorded in the logs.

In this section, we explain how we chose m for *tlpf* methods (*method A*, *method B*, and *filter value*). We then analyze the accuracy of the response time predictions for all seven methods, determining which methods are most accurate. Last, we compare the prediction accuracy to the consistency of the response time data itself.

5.1 Tuning the Timed Low Pass Filter

As demonstrated in section 4.1 for the availability predictions, the m values for the timed low pass filter algorithms for response time predictions did not adhere to one pattern. Therefore, as with the availability predictions, we chose the “best” m value out of the 16 we tried for each indexer contacted by each QM for each *tlpf* method for response time predictions. We used that “best” m value when comparing *tlpf* predictive methods against the other predictive methods for each indexer at each QM.

When we looked at the MSE for a *tlpf* method for “all indexers” contacted by a QM, we chose the best m value for each indexer. In other words, the *tlpf* method for “all indexers” does not use one single m value for the QM, but a separate m value for each indexer contacted by the QM in this combined data.

5.2 Response Time Prediction Results

We evaluated our predictions by comparing them to observations using mean square error (MSE).

Table 9 shows the MSE for each of the response time predictive methods for QM CS-TR, and in the rightmost column the difference between the highest MSE and the lowest MSE for any method for each indexer.

Table 9 - response time prediction MSE for QM CS-TR

indexer	no. obs	mean of all data	single last obs	running average	low pass filter	timed low pass filter tlpf value	timed low pass filter method A	timed low pass filter method B	range of MSEs
cs-tr.cs.cornell.edu:80	14,748	1.7	1.3	1.7	1.5	1.3	1.4	1.6	0.4
cs.nyu.edu:80	1,135	12.6	21.9	12.5	20.5	12.6	12.5	12.5	9.5
dri.cornell.edu:80	14,119	1.9	3.4	2.2	3.3	1.9	2.2	1.9	1.5
ei.cs.vt.edu:8090	13,462	2.3	1.9	2.5	1.9	1.9	1.9	2.3	0.6
lite.ncstrl.org:3803	5,457	10.8	14.6	11.4	13.7	10.8	11.4	10.8	3.8
ncstrl.cc.vt.edu:8080	14,925	15.6	24.3	16.6	23.2	15.6	16.6	15.6	8.8
ncstrl.cc.vt.edu:8081	440	51.3	65.7	51.9	63.1	51.3	51.9	45.2	20.5
ncstrl.cc.vt.edu:8090	15,004	6.4	9.9	6.9	9.3	6.4	6.9	6.4	3.6
ncstrl.cs.cornell.edu:8090	11,397	9.7	7.9	10.1	7.4	7.1	7.7	8.3	3.0
www.cc.gatech.edu:81	2,097	5.8	8.6	5.8	8.1	5.8	5.8	5.7	2.9
www.cs.dartmouth.edu:80	2,185	6.4	9.4	6.6	8.8	6.4	6.6	6.4	3.0
www.cs.uiuc.edu:80	2,735	1.0	1.7	1.9	1.8	1.0	1.7	1.0	0.9
www.cs.umass.edu:80	12,474	10.0	13.8	9.7	12.9	10.0	9.7	9.5	4.3
www.cs.umd.edu:80	1,433	8.8	8.1	8.8	7.6	7.4	7.7	7.9	1.4
www.cs.utah.edu:80	852	1.3	2.6	1.3	2.2	1.3	1.3	1.3	1.3
www.icase.edu:80	12,473	4.5	8.2	5.4	7.8	4.5	5.4	4.5	3.7
www.ics.forth.gr:7000	4,598	7.3	10.0	7.1	9.4	7.3	7.1	7.0	3.0
www.tc.cornell.edu:80	13,893	1.4	1.3	1.7	1.4	1.3	1.3	1.4	0.4
all indexers	143,427	6.2	8.5	6.6	8.1	5.9	6.3	6.0	2.6

Table 9 has a minimum MSE value of 1.0 for the *mean of all data*, *tlpf value* and *tlpf method B* methods for indexer *www.cs.uiuc.edu:80* and a maximum MSE value of 65.7 for the *single last observation* method for indexer *ncstrl.cc.vt.edu:8081*. Comparing different methods for a single indexer, the range of the MSE values can be as narrow as 0.4 (*cs-tr.cs.cornell.edu:80* and *www.tc.cornell.edu:80*) and as wide as 20.5 (*ncstrl.cc.vt.edu:8081*). So unlike the availability predictions, the response time predictions are not similarly accurate for all predictive methods.

Recall that for the availability predictions, the *single last observation* method gave the same MSE as the *low pass filter* method (with m = 0.95). For the response time predictions in Table 9, the *single last observation* and the *low pass filter* methods have similar MSE values, but they aren’t exactly the same. The *running average* and *mean of all data* MSEs resemble each other -- but recall that the *mean of all data* was computed with the same data we are predicting against (i.e. there was no cross training), while the *running average* method doesn’t require foreknowledge. When comparing the *single last observation* and the *running average* methods, 12 of 18 responding indexers in Table 9 had a lower MSE with the *running average* method.

For 13 out of the 18 indexers responding to QM CS-TR, *tlpf method B* had the lowest MSE value, either singly or tied with another predictive method's MSE. The remaining 5 indexers maximized prediction accuracy with the *tlpf value* method, and sometimes with the *single last observation* method as well. In fact, when *tlpf method B* was not the most accurate method, then the *single last observation* method outperformed the *running average* method; in cases where the *running average* did better than *single last observation*, *tlpf method B* was the most accurate.

Since m values of *tlpf* methods are not cross-trained, it is unsurprising that one of the *tlpf* methods can match or beat the methods not benefiting from prescience. What is interesting is that for QM CS-TR, the *running average* method is never more than 1.0 away from the most accurate method, except in the cases of indexer *ncstrl.cc.vt.edu:8081*, where *tlpf method B* is 6.1 lower than the next most accurate method, and indexer *ncstrl.cs.cornell.edu:8090*, where the *single last observation* method is 7.9, the *tlpf value* method is 7.1 and the *running average* is 10.1. So for nearly all indexers responding to QM CS-TR, the *running average* is within 1.0 of the best method even when the range of the MSE values is as high as 8.8 (indexer *ncstrl.cc.vt.edu:8080*) or 9.5 (*cs.nyu.edu:80*), but requires no prescience or complicated calculations of prediction algorithm variables.

The MSE of response time predictions for the four other QMs in our study are similar to those presented in Table 9; we present the MSE for the combined data for all responding indexers for each of the QMs in our study in Table 10.

Table 10 – MSE for all response time predictions

QM	no. obs	mean of all data	single last obs	running average	low pass filter	timed low pass filter tlpf value	method A	method B	range of MSEs
cs-tr	143,427	6.2	8.5	6.6	8.1	5.9	6.3	6.0	2.6
ncstrl	99,406	7.8	10.2	7.7	9.7	7.6	7.5	7.5	2.8
berkeley	54,852	15.2	24.7	15.2	23.4	15.1	15.2	15.0	9.6
lite	5,811	9.9	14.8	9.9	14.0	9.9	9.8	9.2	5.6
forth	479	67.1	118.8	69.4	113.2	67.1	69.2	64.0	54.8

Table 10 shows that the CS-TR QM not only has the most accurate predictions in our study on average, but that the different prediction methods are most consistent for the CS-TR QM – the MSE values only range from 5.9 to 8.5 for this QM. The FORTH QM has the least accurate response time predictions in our study, on average, and their accuracy varies widely depending on the prediction method, from a MSE of 64.0 for *tlpf method B* to 118.8 for the *single last observation* method.

The most accurate method for predicting response times for the CS-TR QM, when examining the MSE for all responding indexers combined, is the *tlpf value* method. *Tlpf method B* performs best for the other four QMs (though *tlpf method A* performs equally well as method B for the NCSTRL QM), and for the CS-TR QM, the MSE for *tlpf method B* is only 0.1 higher than that for the *tlpf value* prediction method. However, the *running average* method, which is simpler to compute and uses no foreknowledge, performs nearly as well. For four of the QMs (all but FORTH), the minimum MSE value is within 0.7 of the MSE value for the *running average*. For two of these QMs (NCSTRL and BERKELEY), the *running average* MSE is within 0.2 of the most accurate method. Even for the FORTH QM, the MSE for the *running average* method, 69.4, is much closer to the lowest MSE value of 64.0 than it is to the highest MSE value of 118.8. Moreover, since the FORTH QM has far fewer observations than the other QMs, we can still assert that for all response time predictions in our study, the *running average* method performs nearly as well as any of the *tlpf* methods or the *mean of all data* method.

If we take the root mean square (RMS), or the square root of the MSE, then we get a rough approximation of the average error for a prediction. For the CS-TR QM, the *running average* predictions on average are within SQRT (6.6) = 2.6 seconds of the response time observations. For the BERKELEY QM, the *running average* predictions are generally within SQRT (15.2) = 3.9 seconds of the response time observations. For the FORTH QM, the *running average* predictions on average are within SQRT (69.4) = 8.4 seconds of the response time observations.

In order to further compare predictive methods and examine prediction accuracy, we now present Table 11, which shows the highest MSE for each predictive method for an individual indexer at each QM. Note that multiple indexers contacted by each QM may be represented in Table 11: the indexer with the highest MSE for one predictive method may be different from the indexer with the least accurate predictions for a different predictive method as viewed by the same QM.

Table 11 - maximum MSE for QM response time predictions for individual indexers

	mean of	single	running	low pass	timed low pass filter		
QM	all data	last obs	average	filter	tlpf value	method A	method B
cs-tr	51.3	65.7	51.9	63.1	51.3	51.9	45.2
ncstrl	21.0	30.3	21.1	28.6	21.0	21.1	21.0
berkeley	38.5	57.4	38.0	54.4	38.5	37.9	37.8
lite	23.0	50.2	23.7	47.7	23.0	23.7	23.0
forth	95.2	169.1	97.1	161.1	95.2	96.9	88.5

In Table 11, *tlpf method B* has the lowest maximum MSE values for individual indexer predictions at all QMs, though the *tlpf value* and *mean of all data* methods equal the *tlpf method B* MSE value at the NCSTRL and LITE QMs. For three QMs, the worst MSE value for an individual indexer using the *running average* prediction method is very close to the worst MSE value using *tlpf method B*: for NCSTRL, the *running average* MSE is only 0.1 greater than the *tlpf method B* MSE, for BERKELEY it's within 0.2 and for LITE it's within 0.7. For the CS-TR QM, the maximum individual indexer MSE for *tlpf method B* is 45.2, while the least accurate predictions using the *running average* method give a MSE of 51.9. This is a discrepancy of 6.7, while the discrepancy for the FORTH QM is 8.6. While these discrepancies seem large, we need to remember that this is a measurement looking at the average of the squared prediction error.

If we examine the RMS for these worst case *running average* predictions for individual indexers represented in Table 11, then the *running average* method is within SQRT (97.1) = 9.9 seconds of response times, on average, for the least accurate individual indexer predictions. If we ignore the FORTH data, then that calculation drops to SQRT (51.9) = 7.2 seconds: *running average* predictions are within roughly 7 seconds of response times, on average, for the least accurate individual indexer predictions at the CS-TR QM. (Recall from Table 10 that overall, CS-TR QM predictions are within 2.6 seconds of the observed response times.)

We will now compare response time prediction accuracy with data consistency; our measurement of data consistency is the variance of the observed response times.

Table 12 - running average response time prediction MSE and RMS compared with mean response times and variances for indexers responding to the CS-TR QM (sorted by variance)

indexer	no. obs	mean resptime	variance	running avg MSE	running avg RMS	RMS/mean
cs-tr.cs.cornell.edu:80	14,748	1.9	0.8	1.7	1.3	68%
www.tc.cornell.edu:80	13,893	1.7	1.0	1.7	1.3	76%
www.cs.uiuc.edu:80	2,735	3.0	1.0	1.9	1.4	46%
www.cs.utah.edu:80	852	0.3	1.3	1.3	1.2	384%
ei.cs.vt.edu:8090	13,462	1.8	1.6	2.5	1.6	87%
dri.cornell.edu:80	14,119	3.2	1.9	2.2	1.5	47%
www.icase.edu:80	12,473	2.0	4.5	5.4	2.3	116%
www.cc.gatech.edu:81	2,097	2.7	5.3	5.8	2.4	89%
ncstrl.cc.vt.edu:8090	15,004	4.2	6.3	6.9	2.6	63%
www.cs.dartmouth.edu:80	2,185	7.0	6.4	6.6	2.6	37%
www.ics.forth.gr:7000	4,598	5.5	7.1	7.1	2.7	48%
www.cs.umd.edu:80	1,433	10.9	8.0	8.8	3.0	27%
www.cs.umass.edu:80	12,474	4.7	9.4	9.7	3.1	66%
ncstrl.cs.cornell.edu:8090	11,397	4.4	9.5	10.1	3.2	72%
lite.ncstrl.org:3803	5,457	7.2	10.7	11.4	3.4	47%
cs.nyu.edu:80	1,135	2.8	11.9	12.5	3.5	126%
ncstrl.cc.vt.edu:8080	14,925	5.1	15.6	16.6	4.1	80%
ncstrl.cc.vt.edu:8081	440	13.3	51.3	51.9	7.2	54%
all indexers	143,427	3.6	6.3	6.6	2.6	71%

Table 12 compares the mean response times and variances for indexers responding to the CS-TR QM and with the MSE and RMS of the *running average* prediction method. We see that in general, as the variance for an indexer's response times increases, the MSE for the *running average* prediction method increases as well. There are exceptions to this: *www.cs.utah.edu:80* has a variance of 1.3 and an MSE of 1.3, while *www.cs.uiuc.edu:80* has a variance of 1.0 and an MSE of 1.9; two other exceptions occur for indexers *dri.cornell.edu:80* and *www.cs.dartmouth.edu:80*.

The root mean square (RMS) column in Table 12 indicates how much, on average, a *running average* prediction differs from the corresponding actual response time. In the best case for the CS-TR QM, *running average* predictions average a 1.2 second discrepancy from observations (indexer *www.cs.utah.edu:80)*, in the worst case the discrepancy is 7.2 seconds (*ncstrl.cc.vt.edu:8081)*, and overall, the discrepancy averages 2.6 seconds.

When we say that *running average* response time predictions, on average, are within 2.6 seconds of observed response times for the CS-TR QM, that sounds pretty good … until we examine the mean response time for all indexers responding to the QM: 3.6 seconds. So our *running average* predictions, on average, are 71% off from the mean of all response times. The best results are for indexer *www.cs.umd.edu:80*, where the *running average* RMS is 3.0 seconds and the mean response time is 10.9 seconds: for this indexer, the RMS is only 27% of the mean. However, this is the indexer with the second highest mean response time for the CS-TR QM. The indexer with the lowest mean response time of 0.3 seconds, *www.cs.utah.edu:80*, has a *running average* RMS of 1.2 seconds, which is 384% of the mean – the highest percentage of any indexer.

Perhaps the most positive information from Table 12 is that the *running average* RMS is 4.1 seconds or lower for all indexers except *ncstrl.cc.vt.edu:8081*, which only had 440 of the 143,427 observations recorded by QM CS-TR. These RMS figures for predictive methods, combined with the variance or standard deviation of indexer response times could inform the search timeout values.

The comparisons of *running average* MSE to mean and variance of response time data for the other four QMs in our study are similar to those presented in Table 12. In Table 13 we compare the mean response times and variances for the combined data of all responding indexers for each QM with the MSE for the *running average* method.

Table 13 - running average response time prediction MSE and RMS compared with mean response time and variance of all responding indexers (sorted by variance)

QM	no. obs	mean resptime	variance	running avg MSE	running avg RMS	RMS/mean
cs-tr	143,427	3.6	6.2	6.6	2.6	71%
ncstrl	99,406	6.6	7.8	7.7	2.8	42%
lite	5,811	3.4	9.9	9.9	3.1	92%
berkeley	54,852	6.8	15.2	15.2	3.9	57%
forth	479	10.0	67.2	69.4	8.3	83%
all QMs	303,975	5.2	8.5	8.7	2.9	57%

In Table 13, as in Table 12, we see that as the variance increases, the MSE increases: as the consistency of the data decreases, the accuracy of our predictions decreases. Almost half of the response time predictions occur at QM CS-TR; nearly a third occur at NCSTRL: these QMs have the lowest *running average* RMS, 2.6 seconds and 2.8 seconds, respectively. So nearly two thirds of the response time predictions have a discrepancy of no more than 2.8 seconds, on average, from observed response times. All response time predictions at all QMs are 2.9 seconds away from the observed response times, on average.

The mean response time for all indexers responding to all QMs in our study is 5.2 seconds; the RMS for the *running average* prediction method is 2.9 seconds. So our *running average* predictions for all response time data are 57% off from the mean for all response time data. The QM with the lowest ratio between *running average* RMS and mean response time is NCSTRL; the worst ratio occurs at the LITE QM, with an RMS of 3.1 seconds and a mean response time of 3.4 seconds.

The variance for all the response time data in our study is 8.5 seconds; the standard deviation is SQRT(8.5) = 2.9 seconds. This is nearly half of the mean response time of 5.2 seconds for all the data in our study; therefore, on average, the response time data is not highly consistent, or tightly clustered around the mean. This is the largest factor in the accuracy or lack thereof in our response time predictions.

6 Conclusions

In this paper, we examined the accuracy of indexer performance predictions made from the query mediator perspective. Such predictions could be used to inform a QM's selection of indexers for query distribution when optimizing for performance (choosing the fastest, most reliable indexer among a set of overlapped or replicated indexers). By improving the QM choice of indexers, we reduce the time a QM spends waiting for indexers to respond, and hence reduce the time a user waits for search results from a distributed digital library. Response time predictions (and a measurement of their accuracy) can also be used to inform search timeout choices at the QM: judiciously chosen search timeouts could reduce QM wait time and thus, user wait time.

We investigated the accuracy of various algorithms when predicting indexer availability and response times from the QM-view, and learned, unsurprisingly, that prediction accuracy is related to data consistency. We also learned that the simple prediction algorithms we used performed as well as our complex algorithms. The primary utility of our work is that it quantifies the accuracy of different prediction methods as well as the consistency of observed data from an operational, world-distributed digital library.

With respect to indexer availability from the QM-view, our combined data had an 87% response rate. Approximately 90% of our indexer availability predictions were accurate regardless of the predictive method used – this high accuracy is due to the high consistency of the data. One of the simplest prediction methods, the *single last observation* method (in which the previous observation is used as a prediction value), performed as well as or slightly better than other predictive methods.

QM-view predictions of indexer response times were not similarly accurate for all predictive methods. *Timed low pass filter method B* was the most accurate, but the simpler *running average* method performed nearly as well. *Timed low pass filter* methods were not cross-trained, and hence their results might be unduly accurate. The accuracy of response time predictions is closely related to the standard deviation of the observed response times – again, we see a correlation between data consistency and prediction accuracy.

On average, for all QMs in our study, the *running average* predictions differed from the observed response times by 2.9 seconds, and 2.9 seconds is also the standard deviation of all observed response times in our study. The poorest response time predictions for a particular indexer as viewed by a particular QM are a more appropriate measurement for adjusting QM search timeouts. *Running average* predictions differed by 9.9 seconds or less, on average, from the least accurate predictions for individual indexers. Ignoring the small amount of data from the FORTH QM, *running average* predictions differ by roughly 7 seconds from observed response times for the least accurate predictions for individual indexers.

None of the prediction algorithms we used required on-going heavy computation from the QM or more than minimal indexer performance monitoring by the QM. Our future work will involve further explorations of how to improve the resource discovery process in distributed digital libraries, including the application of QM-view indexer performance monitoring and predictions to improve QM performance.

Acknowledgments

This work was supported by DARPA grant MDA 972-96-1-006 with the Corporation for National Research Initiatives, DARPA contract N66001-97-C-8542, and NASA GSRP NGT5-50062. This paper does not necessarily represent the views of CNRI, DARPA, or NASA. The authors are grateful to Robbert van Renesse, Elizabeth Slate and especially David Fielding for their contributions, and to UC Berkeley and ICS-FORTH for the log data.

References

[1] "Information Retrieval (Z39.50): Application Service Definition and Protocol Specification," ANSI/NISO, 1995.

[2] Callan, J. P., Z. Lu, et al., "Searching Distributed Collections with Inference Networks," presented at 18th International Conference on Research and Development in Information Retrieval, Seattle, 1995.

[3] Chang, C.-C. K. and H. Garcia-Molina, "Evaluating the Cost of Boolean Query Mapping," presented at ACM Digital Libraries '97, Philadelphia, 1997.

[4] Chu, W., "Optimal File Allocation in Multiple Computer Systems," *IEEE Transactions on Computers,* October, 1969.

[5] Davis, J. and C. Lagoze, "Dienst Protocol Version 5.0," 1997; http://www.cs.cornell.edu/lagoze/dienst/protocol5.htm.

[6] Dushay, N., J. C. French, et al., "A Characterization Study of NCSTRL Distributed Searching," Cornell University Computer Science, Technical Report TR99-1725, January 1999.

[7] Dushay, N., J. C. French, et al., "Using Query Mediators for Distributed Searching in Federated Digital Libraries," to be presented at ACM Digital Libraries '99, Berkeley, CA, 1999.

[8] French, J. C., A. L. Powell, et al., "Comparing the Performance of Database Selection Algorithms," to be presented at ACM SIGIR Conference on Research and Development in Information Retrieval, Berkeley, CA, 1999.

[9] French, J. C., A. L. Powell, et al., "Efficient Searching in Distributed Digital Libraries," presented at ACM Digital Libraries '98, Pittsburgh, 1998.

[10] French, J. C., A. L. Powell, et al., "Evaluating Database Selection Techniques: A Testbed and Experiment," presented at ACM SIGIR Conference on Research and Development in Information Retrieval, Melbourne, Australia, 1998.

[11] French, J. C. and C. L. Viles, "Ensuring Retrieval Effectiveness in Distributed Digital Libraries," *Journal of Visual Communication and Image Representation,* 7 (1), pp. 61-73, 1996.

[12] Gravano, L., C.-C. Chang, et al., "STARTS: Stanford Proposal for Internet Meta-Searching," presented at ACM SIGMOD International Conference on the Management of Data, 1997.

[13] Gravano, L., H. Garcia-Molina, et al., "The Effectiveness of GlOSS for the Text-Database Discovery Problem," presented at ACM SIGMOD International Conference on the Management of Data, 1994.

[14] Lagoze, C., "From Static to Dynamic Surrogates: Resource Discovery in the Digital Age," *D-Lib Magazine*, June 1997.

[15] Lagoze, C., E. Shaw, et al., "Dienst Implementation Reference Manual," Cornell University Computer Science, Technical Report TR95-1514, May 1995.

[16] Lasher, R. and D. Cohen, "A Format for Bibliographic Records," Internet Engineering Task Force, RFC 1807, June 1995.

[17] Leiner, B. M., "The NCSTRL Approach to Open Architecture for the Confederated Digital Library," *D-Lib Magazine*, December 1998.

[18] Roszkowski, M. and C. Lukas, "A Distributed Architecture for Resource Discovery Using Metadata," *D-Lib Magazine*, June 1998.

[19] Vingralek, R., Y. Breitbart, et al., "Web++: A System for Fast and Reliable Web Service," to be presented at the 15th International Conference on Data Engineering, Sydney, Australia, 1999.