Using Experience to Guide Web Server Selection

Maria L. Gullickson[a], Catherine E. Eichholz[b], Ann L. Chervenak and Ellen W. Zegura[c]

[a]Computer Science Dept., University of Washington, Seattle, WA, USA

[b]Computer Science Dept., Christian Brothers University, Memphis, TN, USA

[c]College of Computing, Georgia Tech, Atlanta, GA, USA

ABSTRACT

We examine the use of the *anycasting* communication paradigm to improve client performance when accessing replicated multimedia objects. Anycasting supports dynamic selection of a server amongst a group of servers that provide equivalent content. If the selection is done well, the client will experience improved performance. A key issue in anycasting is the method used to maintain performance information used in server selection. We explore using past performance or *experience* to predict future performance. We conduct our work in the context of a customized web prefetching application called WebSnatcher. We examine a variety of algorithms for selecting a server using past performance and find that the overall average and weighted average algorithms are closest to optimal performance.

In addition to the WebSnatcher application, this work has implications for responsible network behavior by other applications that generate network traffic automatically. By using the techniques we present here, such applications can reduce network and server load, potentially improving performance for interactive applications. The results can also be used to reach conclusions about the performance that would be obtained if anycasting were used in an interactive application.

Keywords: anycasting, web prefetching, server selection, past experience

1. INTRODUCTION

A client requesting multimedia objects (pages that include images, frames, and video) from servers on the World Wide Web will experience large variations in important performance metrics, such as transfer bandwidth and access delay. These variations are caused by time-of-day and server-dependent differences in network paths, network load, server capabilities and server utilization. For information that is found only on a single server, the client has no choice but to tolerate the performance that the server can currently deliver. For information that is replicated at more than one server, however, the client can potentially improve performance by making a careful selection among the servers.

This paper presents one approach to improving client performance when accessing replicated multimedia objects: the anycasting communication paradigm. We use the term "anycasting" to denote a paradigm in which a *name* is used to refer to a service that can be provided by any of a group of servers. For example, the name "Seattle weather" might refer to a group of four servers that all provide forecasts of weather in Seattle. The client issues a request using the anycast group name; our anycast architecture uses a resolver to pick a particular server in the group, and returns the identity of the server to the client. The resolver keeps information on the performance of the servers in the group and uses that information to predict which server will provide the best performance on the current request.

In previous work we defined an architecture for application-layer anycasting[1] and examined a hybrid server-push/client-pull method for maintaining performance information at the resolver.[2] In this paper, we explore the use of an alternative method for maintaining server metrics, namely *past experience*. This technique uses the information obtained over time in accessing servers to predict future performance. We had previously included past experience in an enumeration of possible metric maintenance techniques,[1] but had not evaluated this method. For suitable applications, past experience is attractive because the performance information is obtained in the course of normal information retrieval, without additional overhead.

This work was supported in part by NSF Careers Award CCR-9702609 and by an REU Supplement to NSF Careers Award MIP-9502669.

We examine past experience in the context of a particular application, namely a customized web prefetching application called WebSnatcher. WebSnatcher prefetches objects over the web according to a user's specified interests. The application allows the user to specify groups of equivalent anycasting servers; in addition, it allows users to specify explicit web locations (URLs) and keywords that are used to search for pages relevant to user interests. WebSnatcher prefetches web pages and associated objects, including images and frames, and stores them on the user's local file system. The user avoids network delays when accessing prefetched objects in local disk storage.

In WebSnatcher, we use anycasting specifically for collecting data that change fairly quickly over time. Examples include stock values, weather forecasts and traffic reports. To keep the prefetched information current, WebSnatcher will periodically return to the web servers to obtain updated information. This periodic access to the same server (or server within an equivalent anycast group) provides an ideal opportunity to use experience to guide selection.

This paper presents a performance evaluation of several anycast resolver algorithms used in the WebSnatcher application to choose a server from an anycast group. The algorithms for picking a server include: the server with the shortest network routing path; a round-robin scheme; the server with the best overall average performance; the server that initially has the shortest access time; and the server with the smallest moving weighted average, where recent history is weighed more heavily than the more distant past. We find that the overall average and moving weighted average algorithms have performance that is close to optimal. Choosing the server with the shortest initial access time also performs well in our study; however, this may not be true in general. Significant changes in server or network load may invalidate initial measurements as predictors of server performance; periodically revisiting all servers to update the access time measurements used for prediction is desirable.

One might ask why server performance is important for a non-interactive application such as WebSnatcher. First, we believe that it is important for applications that automatically generate network traffic to behave in responsible ways. By selecting servers that can quickly provide responses, WebSnatcher reduces network and server load. Further, our earlier work indicates that the use of anycasting by *some* clients can improve performance for *all* clients (including those who select servers at random) because anycasting tends to balance load across network paths and servers.[2] Thus, responsible WebSnatcher behavior can reduce the burden on interactive clients. Second, one can use the results of this study to reach conclusions about the performance that would be obtained if anycasting were used in an interactive application, as opposed to a pre-fetching application.

The remainder of the paper is organized as follows. Section 2 explains the anycasting paradigm, describes the WebSnatcher application, and outlines the use of anycasting in WebSnatcher. Section 3 describes the server selection algorithms we studied and presents their performance for several groups of anycasting servers. Related work is discussed in Section 4. We mention directions for future work in Section 5 and conclude in Section 6.

2. ANYCASTING IN THE WEBSNATCHER APPLICATION

In this section, we provide background on the anycasting paradigm and our application-layer anycasting architecture. We then describe WebSnatcher, a customized pre-fetching application based on user-specified interests. Finally, we explain the use of anycasting within the WebSnatcher application.

2.1. Architecture Overview

Anycasting was originally defined[3] as an IP network layer service that provides:

> "a stateless best effort delivery of an anycast datagram to at least one host, and preferably only one host, which serves the anycast address."

In this definition, an *IP anycast address* is used to identify a group of hosts. The hosts in this group are equivalent in the sense that a sender can be satisfied by interacting with any one of the hosts. The sender requests interaction by sending a datagram with the IP anycast address for the group in the destination address field. The datagram is then routed using anycast-aware routers to at least one of the hosts identified by the anycast address.

In our work[1] we adopt a more general view of anycasting as a *communication paradigm* that is analogous to the broadcast and multicast communication paradigms. In the anycasting paradigm, a *name* is used to refer to a service that can be provided by any of a group of servers. The original anycasting proposal[3] can be viewed as providing the anycasting service definition *and* examining the provision of this service within the IP layer, where the naming is done with an IP anycast address. This network-layer-supported anycasting has several limitations, the most important being that application-specific metrics cannot be maintained at the network layer.[1] These limitations motivate our application-layer approach to anycasting.

Whereas network-layer support hinges around the use of anycast IP addresses, our application-layer support makes use of *anycast domain names* (ADNs). An ADN uniquely identifies a (potentially dynamic) collection of IP addresses, which constitutes an *anycast group* *. The function of an application-layer anycasting service is thus to map an ADN into one or more IP addresses. An important feature of application-layer anycasting is that it does not require modifications to network layer operations.

Our design centers around the use of *anycast resolvers* to perform the ADN to IP address mapping. Clients interact with the anycast resolvers according to a basic query/response cycle illustrated in Figure 1: a client generates an *anycast query*, the resolver processes the query and replies with an *anycast response*. A key feature of the system is the presence of *metric databases*, associated with each anycast resolver, containing performance data about servers. The performance data can be used in the selection of a server from a group, based on user-specified performance criteria. Filters at the resolver and at the client provide the mechanism to apply user-specified criteria to select a server.

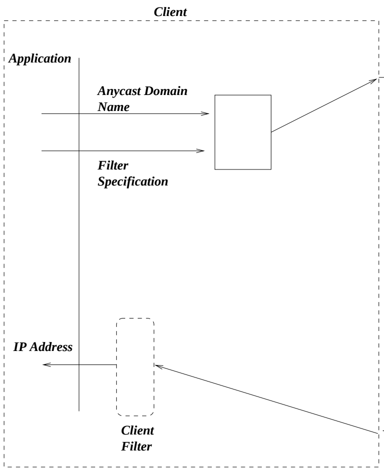

Figure 1. Anycast name resolution query/response cycle

2.2. WebSnatcher: Customized Web Prefetching

WebSnatcher is an application that prefetches multimedia objects over the World Wide Web according to a user's specified interests. WebSnatcher stores prefetched web pages, images and frames on the user's local file system. The user avoids network delays when accessing prefetched objects in local disk storage. Prefetching has been used in other applications and research efforts to reduce network delays.[4–6]

In addition to prefetching pages from specific (non-replicated) servers, WebSnatcher allows a user to specify explicitly which servers they consider to provide equivalent data. Examples of potentially equivalent servers include those providing stock quotes, weather forecasts, and international news. Note that we do not assume that servers in the anycast group are exact replicates, instead —from the user's perspective— they are functionally equivalent. They will return similar data in different formats, any of which would satisfy the user †. The amount of data returned, the network routing and the network load vary among the servers in the anycast group and vary over time for a particular server.

*It is also possible to define an anycast group as a collection of server domain names (or aliases). In this case the anycasting service provides a mapping from an ADN into a host domain name or alias. Obtaining the IP address in this case requires an additional DNS server lookup.

†An interesting variation on the anycasting scheme would allow the user to express preferences amongst the different servers, for example based on the appearance of the web pages that are returned. This policy information could be used with metric information to select the best server.

After WebSnatcher identifies a web page to be prefetched and the corresponding server, the application retrieves that web page and stores it on the user's local file system. Next, Websnatcher parses the fetched web page to identify multimedia objects such as images or frames; it also identifies hyperlinks to other web pages. The application then recursively prefetches multimedia objects and follows hyperlinks to fetch, store and parse new pages. WebSnatcher continues to retrieve and store objects until it reaches the depth of recursion specified by the user.

2.3. Anycasting in WebSnatcher

The characteristics of the WebSnatcher application lead us to use the anycasting framework in three specific ways. First, WebSnatcher collects performance information on servers that handle the application's prefetching requests. WebSnatcher then provides this performance data to the anycasting resolver. Second, we co-locate the anycasting resolver with the WebSnatcher client. That is, we maintain performance information on a per-client basis. While some sharing of performance information among proximate clients may be possible, we anticipate that differences in user preferences will place limits on sharing. Third, as described above, we provide an interface for the user to define and modify the set of equivalent servers that make up an anycast group.

The use of past experience is motivated by the observation that users often make server selection decisions based, in part, on past performance. That is, if one finds a particular server to be unreachable, one is likely to avoid that server for some time. The past experience method is particularly attractive because collecting performance data is free and no server or network modifications are required. Collecting past experience data is free because we can piggyback requests for this data onto requests that are already issued by the user. This method is well-suited for applications like WebSnatcher that frequently return to the same anycast group to fetch time-dependent information and, simultaneously, gather experience data.

WebSnatcher's use of past experience for server selection is in contrast to our previous work, which focused on an architecture for application-layer anycasting[1] and the use of a hybrid server-push/client-pull method for maintaining server metrics.[2]

3. ANYCASTING PERFORMANCE

In this section, we present performance results for our WebSnatcher anycasting experiments. We performed these anycasting experiments on four sets of "equivalent" servers that provided the following services: news, today in history, a Leo horoscope, and weather forecasts for the city of Seattle. Table 1 gives information about the servers in each anycast group. The table includes the geographic location of each server as well as the number of network hops along the routing path from a client at Georgia Tech to each server, determined using *traceroute*.[7] Since routing paths may change over time, this data represents a snapshot of path lengths at one point in time during the course of experimentation. (Note that we did not increase the default traceroute time-to-live beyond 30, so entries with a count of 30 may actually be longer.)

Figure 2 shows the domains involved in the network routing paths for the servers in the Seattle weather anycast group; most paths include multiple hops in a domain. The figure indicates that there is little overlap of network routes, since the different servers are supported by different service providers. Graphs of the network topology for other anycast groups also show little overlap in network paths.

To evaluate the server selection algorithms, we configured the WebSnatcher application to visit each server regularly and collect performance information. Then we post-processed the information to determine the performance that the application *would have seen*, had it used a particular selection method. Table 1 shows performance statistics for the total data collected, for each server. The *Time to First Byte* measurement begins when the client sends a request to the socket and ends when the client receives the first byte of requested data from the server. The *Time to Last Byte* represents a total time to transfer a page and all documents associated with it; the interval begins when the client initiates a server connection and ends when the last byte of the last file is received. Thus, Time to Last Byte may include extra processing and the time to fetch images, frames and other pages over hyperlinks. We measure *Throughput* only during periods of data transfer, not including connection setup or processing overhead. There may be several intervals of data transfer that correspond to fetching a page and the images, frames and hyperlinks associated with it. Throughput is calculated as the total data transferred divided by the sum of the transfer times, where one transfer interval is the time from sending the request over the socket to receiving the last byte of data.

The statistics in Table 1 were collected over a period of 7 days in February 1998. The application ran on an SGI Origin 200 with a fast Ethernet network connection. Each server was visited approximately every several

Server	Location	Hop Count	Mean Time to First Byte (seconds)	Mean Time to Last Byte (seconds)	Mean Throughput (bits/sec)
Leo Horoscope					
www2.romance.net	Toronto, Ontario, Canada	30	1.42	2.62	24327.88
www.rpa.net	Rochester, NY, USA	14	0.17	0.70	1666.25
my.excite.com	Redwood City, CA, USA	14	0.74	2.01	9668.68
stars.metawire.com	Hollywood, CA, USA	13	1.98	3.22	12760.83
www.techweb.com	Manhasset, NY, USA	15	0.56	1.68	15408.42
www.tvguide.com	Radnor, PA, USA	14	0.06	0.46	46719.56
News					
www.yahoo.com	Santa Clara, CA, USA	16	0.17	0.98	20223.83
www.msnbc.com	Redmond, WA, USA	30	0.68	2.00	23693.15
nt.excite.com	Redwood City, CA, USA	14	1.45	2.19	13553.16
www.newscurrent.com	Marina, CA, USA	15	0.61	1.00	15944.88
cgi.pathfinder.com	New York, NY, USA	30	0.88	2.58	18249.56
Today in History					
www.yarranet.net.au	Yarra, Victoria, Australia	20	1.05	3.36	1269.28
www.historychannel.com	New York, NY, USA	13	0.22	0.66	12535.27
www.geocities.com	Santa Monica, CA, USA	17	1.82	6.08	13321.00
lcweb2.loc.gov	Washington, DC, USA	30	0.27	1.03	16561.22
www.outlawpress.com	Albuquerque, NM, USA	15	0.12	0.50	10634.78
www.thehistorynet.com	Leesburg, VA, USA	12	0.12	0.47	3909.06
www.uta.fi	Tampere, Finland	27	3.16	5.88	5401.58
Seattle Weather					
weather.yahoo.com	Santa Clara, CA, USA	16	0.70	2.14	13728.41
www.komotv.com	Seattle, WA, USA	17	0.30	0.90	2994.83
www.atmos.washington.edu	Seattle, WA, USA	16	0.25	0.67	2127.14
www.usatoday.com	Arlington, VA, USA	12	0.22	1.38	60978.61

Table 1. Anycast server groups

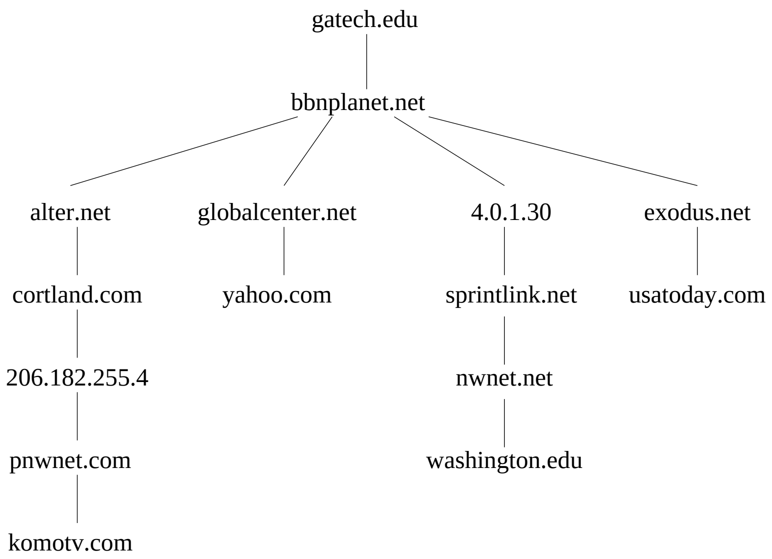

Figure 2. Network topology for Seattle weather group

minutes. Occasionally, the application collecting the server measurements failed and had to be restarted; therefore, the measurements contain several gaps of up to several hours each. Each mean value in the table is the the mean of at least 167 server measurements.

3.1. Selection Algorithms

Next, we present details of the selection algorithms used in our anycasting experiments. We evaluated all the algorithms using the three metrics discussed in the last section: Time to First Byte, Time to Last Byte and Throughput. We represent the i^{th} performance measurement (either throughput or response time) on server j as $X_{i,j}$. The first measurement on server j is labeled $X_{0,j}$. M denotes the number of servers in the anycast group. For the j^{th} server, there are N_j measurements of past behavior. We use these historical values in the following algorithms for selecting one server from the anycast group:

- **Average:** Choose the server with the best mean performance within the anycast group.

$$\min_M \left(\frac{\sum_{N_j} X_{i,j}}{N_j} \right)$$

- **Moving Weighted Average:** Choose the server with the smallest moving weighted average, where the i-th average at server j, $D_{i,j}$, is computed by

$$D_{0,j} = X_{0,j}$$

$$D_{i,j} = \alpha X_{i,j} + (1 - \alpha) D_{i-1,j}$$

  for all $i > 0$, where α is a constant between 0 and 1. This algorithm weighs recent measurements more heavily when calculating average performance. Increasing the α parameter increases the weight of recent values.

- **Minimum:** Visit each server once, and choose the server that has the minimum response time (or maximum throughput) within the anycast group. For the response time values, this is:

$$\min_M (X_{0,j})$$

- **Hop Count:** Choose the server with the smallest hop count from the client machine to the server, using measurements collected once using traceroute. Note that this snapshot information will not be entirely accurate if routing paths change in length.

- **Round Robin:** Select servers in a round robin fashion. The server chosen on the k^{th} data retrieval is the $(k \bmod M)^{th}$ of the M servers.

- **"Optimal":** For comparison, we include an "optimal" choice of servers, which is the best choice we could have made. This choice is determined based on all the collected data from the servers.

The *minimum* and *average* algorithms make static or seldom-changing server selections. The minimum algorithm visits each server once at the start of execution and makes a permanent choice among the servers. The average algorithm always chooses the server with the best average performance; its initial choice is based on one measurement per server. In practice, server choice rarely changes for the average algorithm. Performance of the minimum and average algorithms should improve if we periodically revisit all the servers to update the performance measurements on which server selection is based.

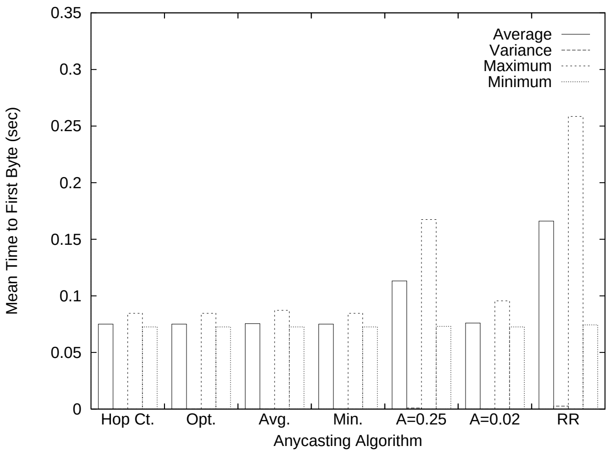

Figure 3. Time to First Byte for News anycast group. Algorithm abbreviations are for *hop count*, *optimal*, *average*, *minimum*, *weighted average* with parameters $\alpha = 0.25$ and $\alpha = 0.02$, and *round robin*. There are four bars for each algorithm representing mean, variance, maximum and minimum for the response time measurements made for each algorithm.

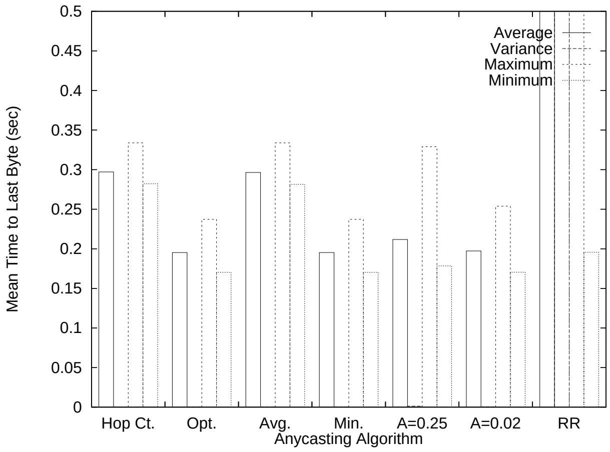

Figure 4. Time to Last Byte for News anycast group. Measurements for the *round robin* algorithm are off the scale of the graph. They include $average = 2.28$ sec, $variance = 12.70$, and $maximum = 11.27$.

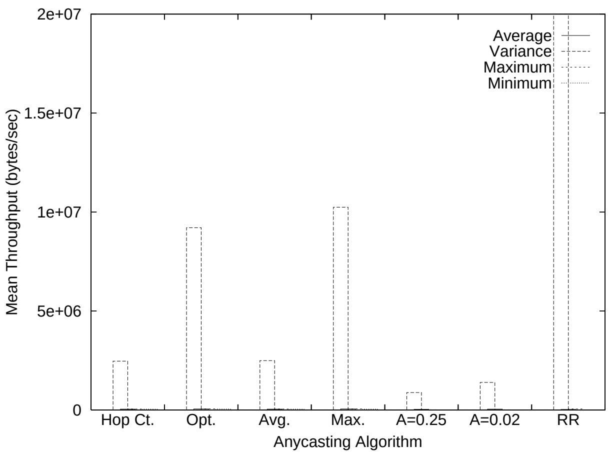

Figure 5. Throughput for News anycast group. These numbers are dominated by variance. Other values are not visible on the graph.

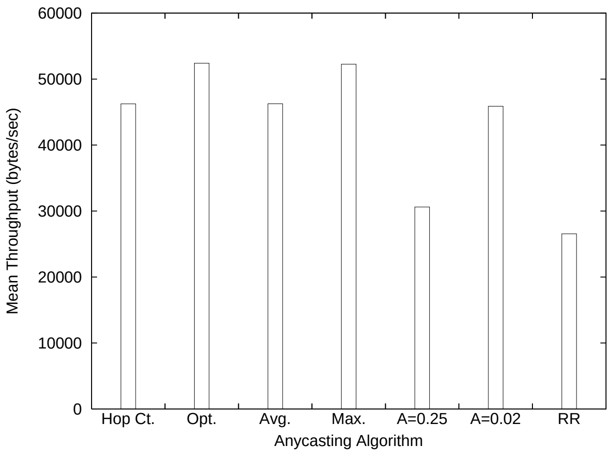

Figure 6. Shows only the average throughput for News anycast group.

3.2. Performance Results

In this section, we present the measured performance of these anycast resolver algorithms. The measurements in this section were gathered over a period of four days in July, 1998. Each server was visited approximately 280 times at intervals of approximately 10 minutes. The WebSnatcher application ran on a Sun workstation under the Solaris operating system. Some gaps in our measurements correspond to the application failing and later being restarted. The anycast groups and the hop counts are the same as those presented in Table 1, with a few exceptions where web pages changed names or had slight path changes.

After collecting data from all the servers at regular intervals, we performed post-processing on the complete set of measurements. For a particular algorithm, the performance results below include only those measurements that actually would have been made during algorithm execution. We also discarded some measured values that appeared to be atypical to avoid skewing our results. For example, unusually long access times most likely reflected servers being temporarily unavailable. Therefore, the mean results presented below are "trimmed mean" values; we discard the top *and* bottom ten percent of the performance measurements in calculating the mean values.

3.2.1. Evaluation of Metrics

Figures 3, 4 and 5 show the performance of the resolver algorithms for the News anycast group using the Time to First Byte, Time to Last Byte, and Throughput metrics. Each graphs shows mean, variance, maximum and minimum server performance. (For the throughput metric, the variance dominates the other values; Figure 6 shows only the mean throughput values for the News server for easier comparison.) The Time to Last Byte and Throughput algorithms include measurements of the complete data transfer time, while the Time to First Byte measures only the time until the client receives the first byte of data.

Using the Time to First Byte metric results in good and predictable performance. When selecting servers based on this metric, the average algorithm and the moving weighted average algorithm with an appropriate α parameter produce mean performance that is close to optimal, while avoiding large maximum values and maintaining low variance. The other metrics are not as consistent. The Time to Last Byte metric exhibited relatively large maximum values and variance for some algorithms, for example, *round robin*. The Throughput metric exhibited very large variance. These latter two metrics are more sensitive to variations in network and server load *during* transfers, making it difficult to predict which server will perform best for a future transfer. For the remainder of this paper, we use the Time to First Byte metric to make predictions in the anycast resolver algorithms.

3.2.2. Evaluation of Algorithms

We compare the performance of our algorithms to *optimal* performance. We calculate optimal performance by visiting every server at every time interval and using this complete knowledge to pick the best server for the current transfer. The optimal algorithm is not practical, since it would generate too much server and network traffic. However, it provides a basis for evaluating the performance of practical algorithms.

Simple, practical alternatives to the optimal scheme that do not rely on past experience include *round robin* and *hop count*. In the round robin algorithm, we simply switch servers on every access, visiting each server an equal number of times. Hop count is a static algorithm that selects a server based solely on the network routing distance to the server. Figures 3, 7, 8 and 9 show the Mean Time to First Byte performance for the News, Today in History, Leo Horoscope, and Seattle Weather anycast server groups. These figures show generally poor performance for the *round robin* and *hop count* algorithms. The round robin algorithm alternates among servers, experiencing a mix of the best and worst performance; if there is a high variance in server or network performance, the round robin scheme performs poorly with respect to optimal. The hop count algorithm always chooses the closest server. This is not necessarily a good choice, since the nearby server or the network path to that server may be highly loaded, resulting in poor performance. These results are consistent with observations by others.[8,9]

Among the algorithms that rely on past experience, the *average* algorithm performs fairly well for all the anycast groups. The average algorithm maintains a running average of each server's performance, with recent values given equal weight with older values; the algorithm chooses the server from the anycast group with the lowest mean response time to first byte. The average algorithm has mean performance close to optimal and generally small variance. However, maximum values are higher than for the optimal algorithm in some cases. This suggests that the average algorithm occasionally makes a poor server choice. This happens rarely enough that the mean and variance show little effect.

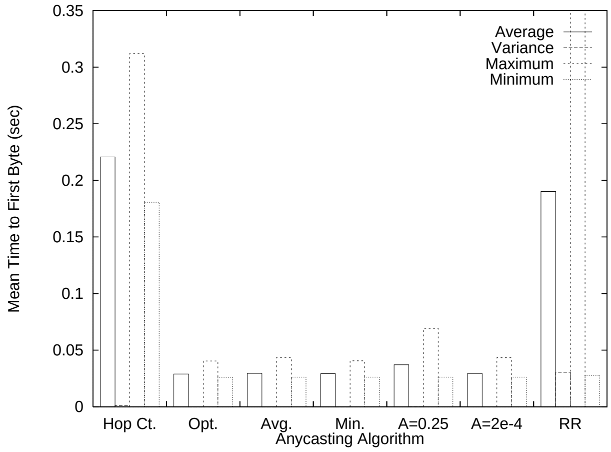

Figure 7. Time to First Byte for Today in History anycast group

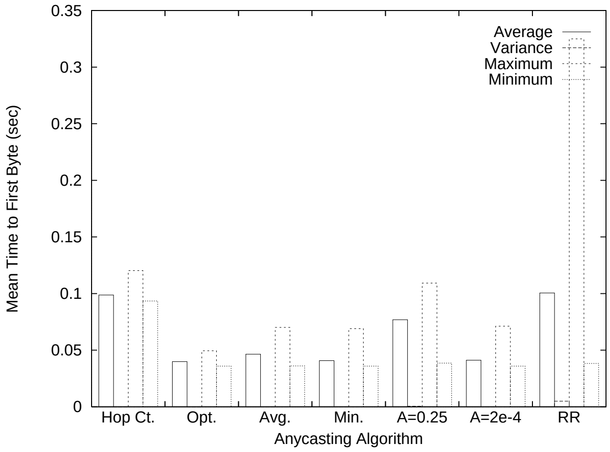

Figure 8. Time to First Byte for Leo Horoscope anycast group

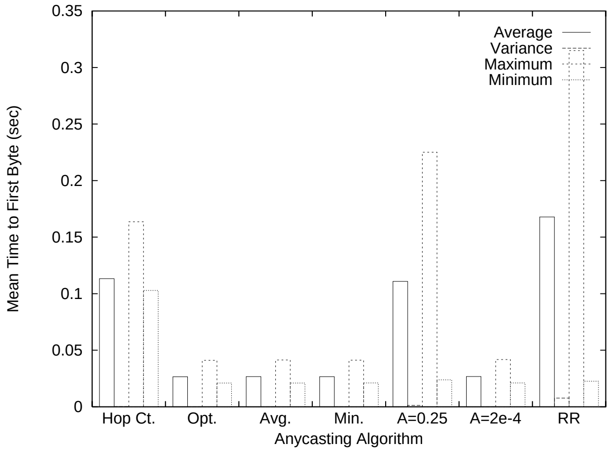

Figure 9. Time to First Byte for Seattle Weather anycast group

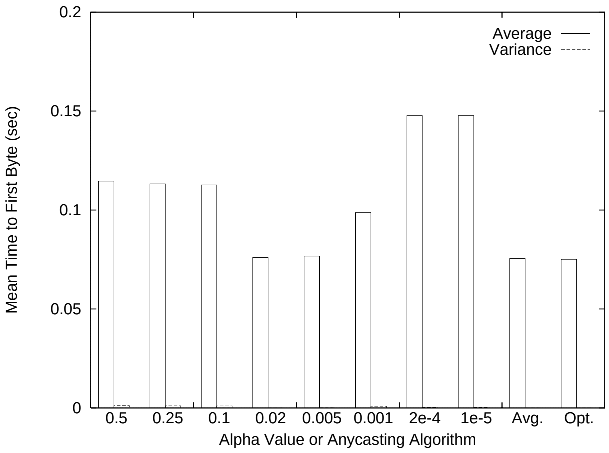

Figure 10. Evaluation of α parameter for moving weighted average for News servers. Minimum, average and optimal algorithms are included for comparison. Shows that performance is best for a range of small α values. Choice of α values that are too large or too small hurt performance.

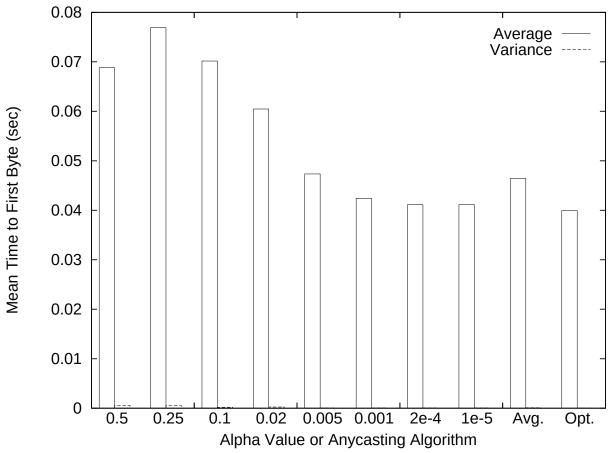

Figure 11. Evaluation of α parameter for moving weighted average for the Leo horoscope servers. Minimum, average and optimal algorithms are included for comparison. For appropriately small values of α, the moving weighted average algorithm performs better than the average algorithm.

The performance of the *moving weighted average* algorithm is sensitive to the choice of the α parameter. This algorithm assigns greater weight to recent server measurements in calculating a weighted average or decay value. Because recent measurements reflect changes in server or network loads, we would expect this algorithm to perform better than the average algorithm, which assigns equal weights to all server measurements. We illustrate the effect of changing the α parameter in Figures 10 and 11, which show Time to First Byte performance results for the News and Leo anycast groups. Besides the performance of the moving weighted average algorithm with a variety of α parameter values, these graphs include performance for the average and optimal algorithms. Both graphs show a range of small α parameters for which performance is optimal. Choosing an α parameter that is too large or too small hurts performance, apparently by placing too much or too little emphasis on recent performance values compared to longer experience. An important question for future work is the development of schemes to choose an appropriate α parameter value for a particular anycast group.

Figures 3, 7, 8 and 9 show Mean Time to First Byte performance for $\alpha = 0.25$ and for a value of α in the range of optimal values. (Three of these graphs use an optimal value of $\alpha = 0.0002$.) In each case, the weighted average algorithm with an appropriate α parameter performs as well as the average algorithm. In one case, for the Leo anycast group, the performance of the weighted average algorithm is better than for the average algorithm.

The *minimum* algorithm also performs well for each of the four anycast groups we studied. We visited each server once and used that single performance measurement to make a static selection of a server that received all future requests to the anycast group. This selection algorithm produced performance that was close to optimal in Figures 3, 7, 8 and 9. However, if significant changes in server or network loads had occurred during the period of our study, these initial measurements might no longer be good predictors of server performance. A better approach would be to revisit the servers periodically and update the minimum performance values. We experimented with revisiting servers for the various anycasting groups in our study, but found no performance improvement, since performance was already near optimal. In general, however, such periodic revisiting of servers will likely be necessary.

4. RELATED WORK

The server or resource finding problem has been the subject of much investigation for over a decade. Most relevant to our work are several recent papers that address server selection using measurements from clients. Carter and Corvella[10,11] have considered selection based primarily on the characteristics of the path leading to the server. While they acknowledge the desirability of using server load information as a guide to server selection, their work does not address this issue (except for a limited experiment[11]).

Two other recent papers incorporate server load into client-side selection.[12,9] The WebSeAl work[12] monitors the response time for each request from a client and uses these times to select a server to satisfy a subsequent request. Thus, data collection is entirely driven by client requests. One algorithm is used for selection, namely a probabilistic weighting based on the average response time from the last N measurements. This algorithm compares favorably to selection of the closest server in a limited set of experiments.

The work by Sayal et al.[9] also monitors response times for requests and uses the measurements to select a server. The response time measured is the time to first byte, and the goal is to minimize this time. This work proposes two selection algorithms: a Probabilistic algorithm similar to the WebSeAl approach and a Refresh algorithm. The Refresh algorithm combines data from client requests with probes to access a server if the most recent measurement has reached an expiration time. The two new algorithms are compared to four algorithms previously studied.[8] The Probabilistic and Refresh algorithms both compare favorably with the alternatives in a limited set of experiments.

Both papers suffer from the lack of justification for why a client would make repeated requests, closely spaced in time, to the same server. The experiments appear to use back-to-back requests, thus the collected data is optimally fresh and the results are likely to be better than one would observe in practice. The WebSnatcher application that accesses time-varying data provides a justification for frequent requests to the same server. Further, we experiment with requests that are spaced in time (by several minutes), rather than occurring back-to-back. We also consider a wider variety of performance metrics that may be of interest to a user.

Finally, the Service Location Working Group of the IETF has been developing protocols to facilitate the discovery and selection of network services.[13] Their emphasis has been on an enterprise network, rather than the wide-area. Further, they have focused on architectures and protocols, rather than performance studies.

5. FUTURE WORK

In the short term, we will extend this work to evaluate the performance of the anycasting scheme when multiple clients are accessing servers, using both anycasting and direct requests. Our past work with multiple clients indicates that the use of anycasting by *some* clients can improve performance for *all* clients (including those who select servers at random) because anycasting tends to balance load across network paths and servers.[2]

In the longer term, we plan to investigate the sharing of information by users in close geographic proximity. We will also allow users to express preferences about which servers' data formats they prefer, to be used as another input to the resolver algorithm.

6. CONCLUSIONS

We have presented performance results for several algorithms that use past experience to select a server from an anycast group. The algorithms for picking a server include: the server with the shortest network routing path; a round-robin scheme; the server with the best overall average performance; the server that initially has the shortest access time; and the server with the smallest weighted average, where recent history is weighed more heavily than the more distant past. We evaluate these algorithms within the WebSnatcher prefetching application. WebSnatcher is well-suited for use with anycasting when the prefetched information changes periodically and can be found on multiple servers.

We examined three performance metrics; all are potentially important for a user accessing multimedia information. We found that the Time to First Byte metric appears to be easiest to predict. It avoids large maximum values and maintains low variance, meeting the goals of good and predictable performance. This cannot be said for the other two metrics. The Time to Last Byte metric exhibited large maximum values and variance in some cases, while the Throughput metric exhibited large variance. These latter metrics are more sensitive than Time to First Byte to variations in server and network load *during* transfers, making it difficult to predict which server will perform best for a future transfer.

Using the Time To First Byte Metric, we found that the *average* and *moving weighted average* algorithms are consistently able to produce average performance that is close to optimal. For the *moving weighted average* algorithm, we studied sensitivity to the α parameter, and found that performance is best for a range of small α parameters. Increasing or further decreasing the value of α harms performance either by weighing recent performance too heavily or not heavily enough. The *minimum* algorithm also gave close to optimal performance in our study; however, in general, changes in server or network load may invalidate initial measurements as a basis for server performance predictions.

Periodically revisiting all servers to obtain updated performance measurements is desirable. For comparison with these algorithms, we include the *"optimal"* algorithm; however, this algorithm is not practical, due to the heavy network and server burden required to collect the necessary data. Practical alternatives to collecting past experience, such as the *hop count* and *round robin* algorithms, perform significantly worse than the algorithms that consider past performance when selecting servers within an anycast group.

REFERENCES

1. S. Bhattacharjee, M. H. Ammar, E. W. Zegura, V. Shah, and Z. Fei, "Application layer anycasting," in *Proceedings of INFOCOM 97*, 1997.
2. Z. Fei, S. Bhattacharjee, E. Zegura, and M. Ammar, "A novel server selection technique for improving the response time of a replicated service," in *To appear in INFOCOM 98*, 1998.
3. C. Partridge, T. Mendez, and W. Milliken, "Host anycasting service," *RFC 1546* , November 1993.
4. T. S. Loon and V. Bharghavan, "Alleviating the Latency and Bandwidth Problems in WWW Browsing," 1997. http://timely.crhc.uiuc.edu/Papers/usits_v1.ps.
5. *Wget Manual*, 27 May 1997. http://www.lns.cornell.edu/public/COMP/info/wget/wget_toc.html.
6. "The Informant - Common questions." FAQ, 1997. http://informant.dartmouth.edu/about.html.
7. V. Jacobson, "Traceroute." available from ftp://ftp.ee.lbl.gov/traceroute.tar.Z.
8. M. Crovella and R. Carter, "Dynamic server selection in the internet," in *Proceedings of Third IEEE Workshop on the Architecture and Implementation of High Performance Communication Subsystems (HPCS'95)*, August 1995.
9. M. Sayal, Y. Breitbart, P. Scheuermann, and R. Vingralek, "Selection algorithms for replicated web servers," in *Workshop on Internet Server Performance (WISP)'98*, 1998.
10. R. L. Carter and M. E. Crovella, "Server selection using dynamic path characterization in wide-area networks," in *Proceedings of INFOCOM 97*, 1997.
11. R. L. Carter and M. E. Crovella, "Dynamic server selection using bandwidth probing in wide-area networks," Tech. Rep. BU-CS-96-007, Computer Science Department, Boston University, Boston, MA, 1996.
12. M. Karaul, Y. Korilis, and A. Orda, "WebSeAl: Web Server Allocation," Tech. Rep. TR1997-752, Department of Computer Science, New York University, 1997.
13. J. Veizades, E. Guttman, C. Perkins, and S. Kaplan, "Service location protocol," *RFC 2165* , June 1997.