Chapter 9

Multi-Source Prefetching

9.1 Introduction

In this thesis we have examined Web predictions from two general sources — prediction from content, and prediction from history. These can be broken down into many independent sources of prediction information:

- Links from the current Web page.
- Links from other browser sources, such as bookmarks.
- Links from content in user's context, such as Usenet news and e-mail.
- URLs based on the user's historical access patterns.
- URLs based on the average user's access patterns as seen by the server.
- URLs based on a workgroup's access patterns as seen by a proxy.

Some of these we have explored in depth in previous chapters. However, each of these sources of predictions have limitations. Each provides a different viewpoint and each has relevant knowledge at different times.

This chapter will explore the results of combining predictions from multiple sources. In particular, we will examine the performance improvements possible when:

- combining content-based prediction and history-based prediction.
- combining multiple history-based predictions from various sources.

Each combination is capable of providing performance improvements in some situations.

While not common, other researchers have proposed the idea of combining information from various sources for prefetching. Hine *et al.* [HWMS98] builds models of resources likely to be accessed in the same session and combines that knowledge with a categorization of the client's interest in resources at the site. In contrast, our approach will build models of user behavior from the perspective of the client as well as the server, but does not attempt to classify the user interest.

Jiang and Kleinrock [JK97, JK98], on the other hand, explicitly suggest the use of both client and server-based predictions. In their approach, if the client has visited this page less than five times and the server has a prediction, then use the server's prediction. Otherwise, use the client's prediction. Unfortunately, the experimental tests are too limited to determine realistic performance measures.

9.2 Combining Multiple Predictions

Prediction is often quite similar to classification, and combining multiple classifiers has a history of success in machine learning tasks. For example, boosting is a method for finding accurate classifiers by combining many weaker classifiers [FS96, SFBL97]. The weighted majority [LW94b] and Winnow [Lit88] algorithms can be seen as methods for combining the classifications of many simplistic "experts". However, combining ranked predictions in machine learning is not so common.

In a sense, the PPM-style predictors introduced in Chapter 4 already combine multiple predictions. PPM takes the predictions of multiple contexts (i.e., the predictions of Markov models of different orders) and combines them to generate an overall probability distribution of the possible next actions.

Our approach will be similar. After every action in sequence, each prediction source provides a (possibly empty) list of predictions, each associated with a weight. We will then merge those predictions using a fixed weight, as we did in Section 4.5.3 with performance similar to PPM. More complex merging functions are certainly possible, and may offer improved performance (as they have in other areas such as search engines [CSS99, DKNS01]).

One issue relevant (but not specific) to our application is the concern for trust of predictions. If we were to implement a prefetching system that incorporated the use of server hints as a source of predictions, we would need a trust model for those predictions. Since the predictions are likely to influence the prefetching activity of our system, unscrupulous server operators could send hints with high weights to cause, for example, increased traffic that might be reported to advertisers, or if given sufficient traffic, to overload some unsuspecting target server. While we do not explore this issue further, we note that some mechanism will be needed to model the trustworthiness of the servers sending such hints.

9.3 Combining Content and History

In this section we consider the task of combining content-based and history-based predictions. We will use the same log as described in Chapter 6, apply the history-based mechanisms from Chapter 4, and compare the predictive performance of each separately and in combination. Unfortunately, because we will be using content-based predictions, we will be unable to use NCS to simulate response times, as the content-based predictions include objects that were never requested on the original trace, and thus we do not have sizes or retrieval times for those objects. We will instead focus on predictive performance by incorporating the content-based predictions into a modified version of the history-based prediction codes used in Chapter 4.

Recall the performance of the content-based predictor from Chapter 6. For comparison, we'll use the top-1 and top-5 similarity-based orderings using up to 20 terms around the anchor, and we'll measure the predictive performance over all points (not just the possible ones). In this environment, the content-based predictor gets 5.3% and 15.8% correct for top-1 and top-5, respectively. A PPM-based predictor (using bigrams and trigrams) that makes a prediction whenever the probability of success is estimated to be .1 or above gets 6.8% and 8.4%, respectively. However, if we combine the two predictors — by using the content-based predictor whenever the history-based predictor did not use all of its allowed predictions, we get much better performance, as shown in Figure 9.1. For this dataset, at least, the two predictors make predictions in mostly

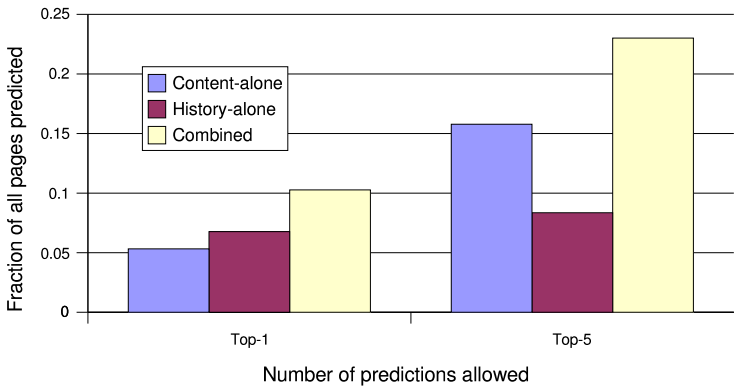

Figure 9.1: Predictive accuracy on the full-content Rutgers trace using prediction from history, from content, and their combination.

different scenarios, resulting in performance that is close (between 85% and 95%) to the sum of the performance of the two predictors alone.

9.4 Combining History-Based Predictions from Multiple Sources

In this section we examine the performance potential for combining multiple history-based predictions. We consider three logs — the SSDC Web server log, a Web server log from NASA, and the UCB Web proxy log.

For each trace we will present the results of an arbitrary combination of simulation parameters. For the Web server logs we will also example a wider selection of possible prediction parameter settings. In particular, we will examine the effect of the following on response times:

- the number of predictions permitted (from 1 to 5),
- the maximum n-gram size used (from 2 to 4),
- the minimum support for prediction (from 1 to 20), and
- the minimum confidence for prediction (from .01 to .2).

9.4.1 Web server log: SSDC

9.4.1.1 Initial experiments

As described in Section 4.4.3, the SSDC Web server log records the requests made by users of a modem-connected small software development company's Web server. We will use NCS to simulate a sample of potential prefetching configurations and evaluate the client-perceived response times that result from those configurations. In our simulations we will assume the use of a modem with a slightly less than 32kbps bandwidth and 100ms one-way latency. A mix of clients connections are likely (i.e., both modem and broadband users) so the simulated client is given approximately 120kbps bandwidth and 40ms one-way latency. Thus clearly the origin server bandwidth is the bottleneck, as it was for the system in reality.

When caching or prefetching at the client is used, the client will have an additional 2MB cache. Prediction models at either the client or server will be built with the same parameters: n-grams must be between 1 and 4 requests long; a scenario must be seen at least 10 times before a prediction is made; and the probability in a prediction must be at least .1 to be used. The latter two provide thresholds to require some confidence in the prediction before spending the resources to prefetch.

Table 9.1 summarizes the performance of various configurations. It provides the mean and median response time for each approach, as well as the bandwidth required, and the fraction of requests served by prefetched content, plus the fraction of requests that prefetched content that was unusable.

From the same experiments, Figure 9.2 depicts the fraction of requests served with

	Response time		**Fraction of**	**% reqs pref'd**	**% reqs pref'd**
Approach	**mean**	**median**	**original bw**	**successfully**	**unsuccessfully**
Prefetch perfectly	4.23s	0.95s	97.5%	30.88%	1.26%
Prefetch both	4.64s	1.17s	133.7%	34.06%	59.94%
Prefetch server	4.62s	1.18s	132.3%	33.89%	58.98%
Prefetch client	5.16s	1.68s	94.7%	0.08%	0.11%
No prefetching	5.16s	1.68s	94.7%	0%	0%
No cache	5.91s	1.91s	100%	0%	0%

Table 9.1: Summary statistics for prefetching performance when one prediction is made with the SSDC log.

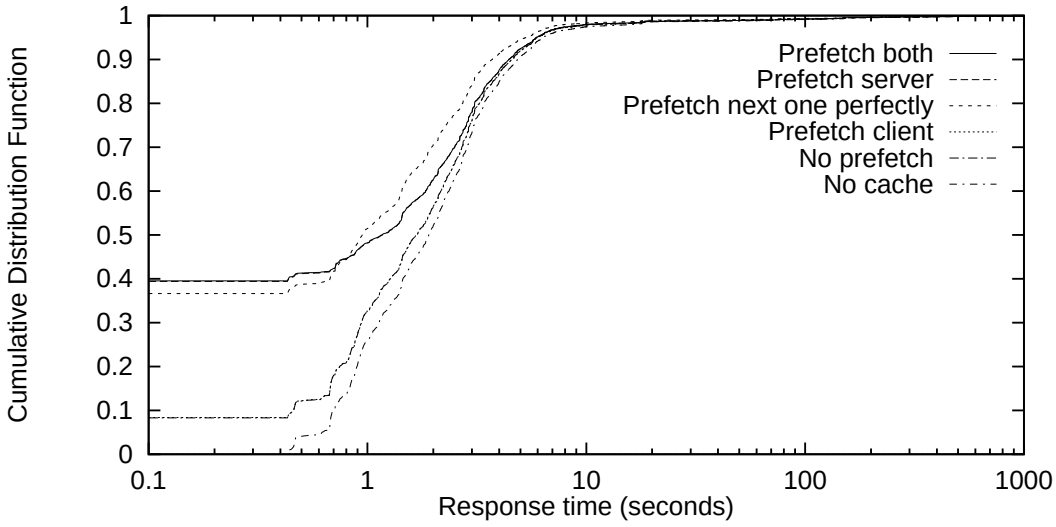

Figure 9.2: Cumulative distribution of client-side response times when only one prediction is used with the SSDC log.

at most a given response time, by showing the cumulative distribution of client-side response times. The bottom-most line corresponds to the performance seen without any additional caching or prefetching. As a result, it has the worst of all performances with a mean and median response times of 5.91s and 1.91s (recorded in Table 9.1 while using the originally recorded bandwidth (defined for comparison as 100%). The next two lines overlap almost entirely — the configuration in which an additional cache at the client is used, and when those clients can prefetch their single best prediction. This is because the clients are rarely able to make predictions (because of the minimum thresholds established). With an object hit rate of over 8%, the mean and median response times improve, to 5.16s and 1.68s, and bandwidth needs are reduced by 5%.

The server, however, has the benefit of seeing all requests and thus having lots of data to provide for a better model. When the server provides the predictions for the client to prefetch, it achieves client object hit rates of 39.4% and mean/median response times of 4.62s/1.18s. Even with that performance, the per-client model can still provide some benefit, as when the two are combined, hit rates increase slightly to 39.6%, reducing the median slightly to 1.17s, but increasing the mean response time to 4.64s. Bandwidth usage in both cases increases by close to a third.

At the top we show performance provided by the perfect predictor — that is, if after

Approach	Response time mean	Response time median	Fraction of original bw	% reqs pref'd successfully	% reqs pref'd unsuccessfully
Prefetch perfectly	4.23s	0.95s	97.5%	30.88%	1.26%
Prefetch both	4.41s	0.75s	179.1%	41.94%	122.8%
Prefetch server	4.55s	0.94s	170.4%	39.42%	106.8%
Prefetch client	5.16s	1.68s	94.8%	0.10%	0.21%
No prefetching	5.16s	1.68s	94.7%	0%	0%
No cache	5.91s	1.91s	100%	0%	0%

Table 9.2: Summary statistics for prefetching performance when five predictions are made with the SSDC log.

each request the client knew the next request needed by the user, it could attempt to prefetch at that time. This scenario provides a median response time of just .95s and reduces mean response time to 4.23s while using only 97.5% of original bandwidth. The perfect predictor has bandwidth usage less than 100% because of the use of caching, but has greater usage than the caching-only configuration because it has some requests that are unsuccessful (not retrieved in time to be useful) and may additionally push some otherwise useful objects out of cache.

If we allow up to five predictions, we can improve performance further. Table 9.2 provides statistics over the same set of configurations (and depicted again as a CDF in Figure 9.3), but with the use of five predictions for prefetching. Five choices

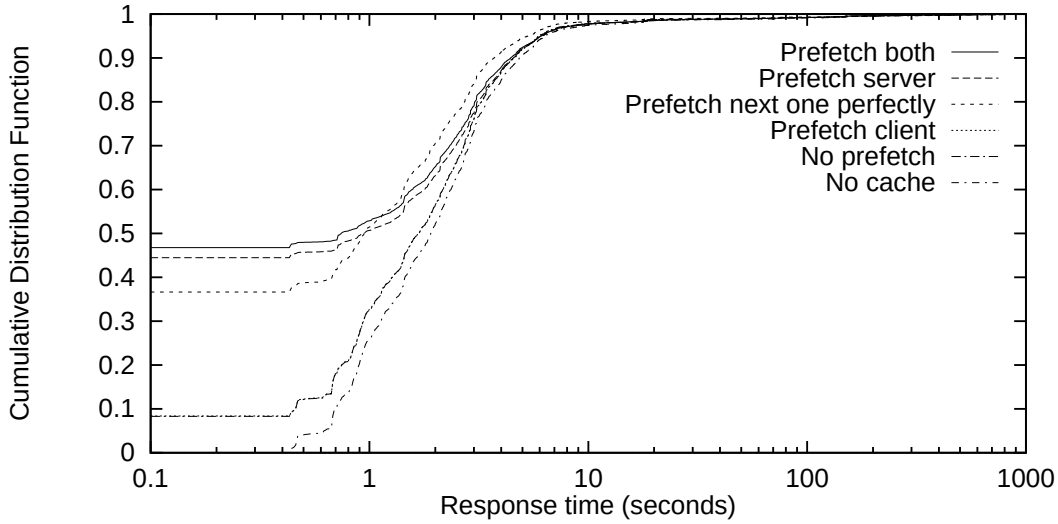

Figure 9.3: Cumulative distribution of client-side response time when five predictions are used with the SSDC log.

does not help client prediction in this dataset, but the server's model can increase hit rates further, to 44.5% and decrease mean and median response times to 4.55 and .94s respectively. The cost for this increased performance is increased bandwidth usage — 70% more than the original. When combined with the client predictor, hit rates reach 46.8% and client response times drop to a mean of 4.41s and median of .75s, at a cost of 75% more than the original bandwidth.

At first glance, such performance improvements seem impossible, since hit rates and median response times are considerably better than the perfect predictor. However, there is a rational explanation. Although the perfect predictor knows exactly what the next request will be, that knowledge is only available when the current request is made. In many cases, it is not possible to retrieve the object before it is requested. The five guess configurations, on the other hand, prefetch many more objects, but do so further in advance, allowing the object to arrive perhaps multiple requests before it is needed and thus providing even higher hit rates and shorter average response times.

9.4.1.2 Parameter studies

We now systematically examine how performance changes as the prediction parameters are modified. We first investigate the relationship between median response times to that of mean response times. We show the asymmetry of the two metrics here to provide a reference point for comparison as we will focus on median measurements in future graphs.

Figure 9.4 plots the median versus mean response times for the various configurations of our prefetching system that uses server hints for the SSDC trace. One point represents the caching-only configuration; the other points signify the other tested configurations. Mean response times range from slightly over 4 seconds to just over 18 seconds. Median response times vary from under .5 seconds to close to 2.1 seconds. This figure shows that while many configurations can significantly improve median response times (since there are many points well under the caching-only configuration), none significantly improve mean response times (since better points are not much below the caching-only configuration), although many significantly increase mean response

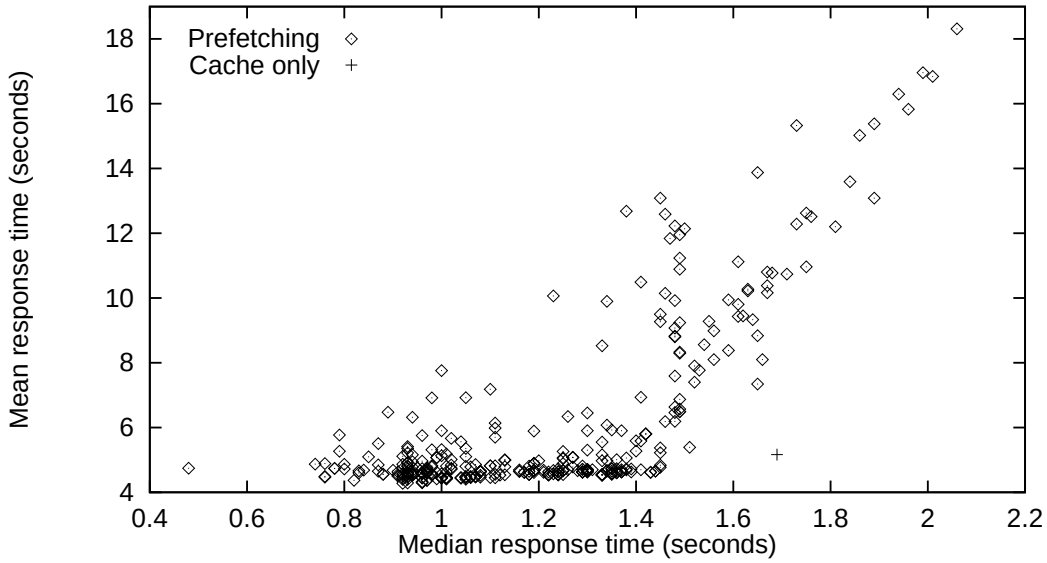

Figure 9.4: Scatterplot comparing median versus mean response times for various configurations using the SSDC trace.

times. It also suggests that the very large or slow to retrieve objects are less likely to be prefetchable (since they are a significant contributor to the mean, which does not improve much).

Our next concern is for the cost of prefetching as compared to its benefits, and how parameters might be used to optimize the benefits gained. Therefore, in Figure 9.5, we plot the median response time versus bandwidth usage. We also distinguish among the number of predictions permitted (labelled Prefetching 1, Prefetching 2, etc. in the figure). All of the prefetching approaches use more bandwidth than a caching-only configuration, but it is apparent that many prefetching approaches are capable of reducing the median response time while using no more than twice the bandwidth of the original trace. The figure also demonstrates that many configurations are able to cut the caching-only median response time in half (or better).

We can also make some observations based on the number of predictions permitted. Using just a single best prediction is typically insufficient for large improvements in response times. However, configurations using two predictions are sufficient to provide a range of points along the lower left edge, demonstrating various tradeoffs between bandwidth and response time. The use of three or four predictions result in curves

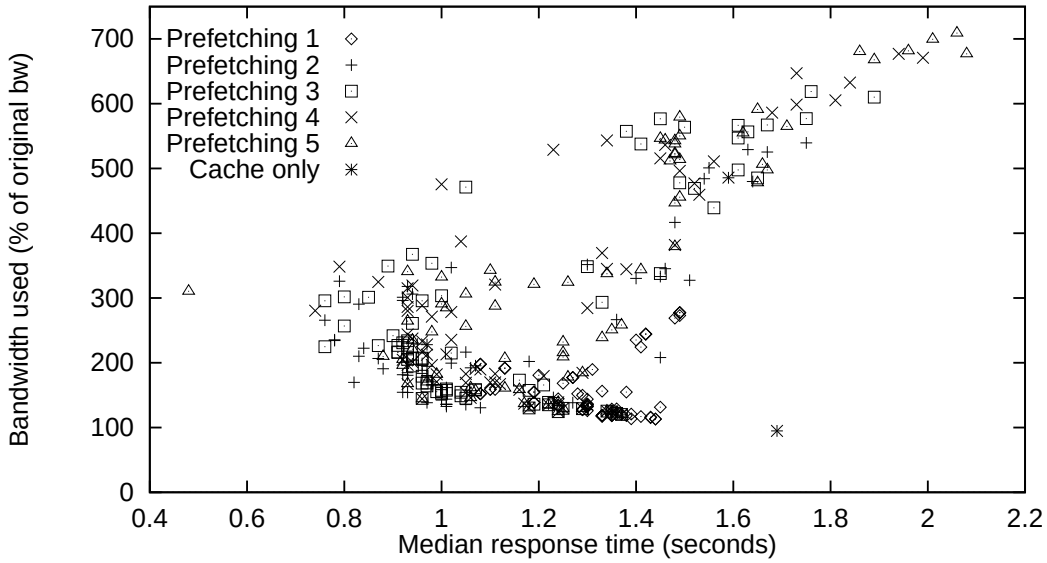

Figure 9.5: Scatterplot comparing median response time versus bandwidth consumed using the SSDC trace, highlighting the number of predictions permitted.

slightly further back. Five predictions are likewise further back, but are also more spread out through the field, achieving both the best and worst median response times.

Finally, four points deserve highlighting. First is the caching only configuration, using 94.7% bandwidth to provide a 1.69 seconds median response time. Second, a five prediction point provides an outlier with much better median response time than any others, at 0.48 seconds. That configuration allowed n-grams up to length four (the maximum tested) and required a minimum of two repetitions and minimum probability of 0.1 (both in the middle of ranges tested). Third, while not one of the smallest medians, a two prediction point along the edge provides an excellent tradeoff of 170% bandwidth usage to achieve a median response time of 0.82 seconds. Later, in Figure 9.8 we will use this configuration with the label "Second best server". It used a fixed n-gram length of two, required ten repetitions and a minimum confidence of 0.05 to make a prediction. Fourth, in the top-right corner lies the point with the largest median response time (our "Worst server"). It also used a bigram length of two and minimum confidence of 0.05, but required only 1 repetition. Thus, many predictions satisfied the requirements and were used, resulting in less successful prefetching, leading to congestion and increasing delays.

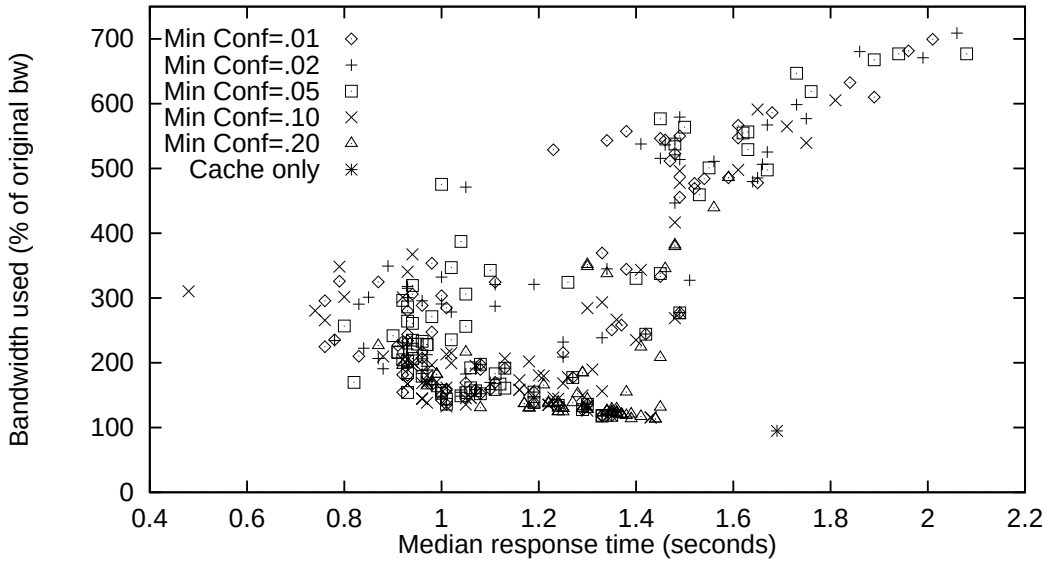

Figure 9.6: Scatterplot comparing median response time versus bandwidth consumed using the SSDC trace, highlighting the minimum confidence required.

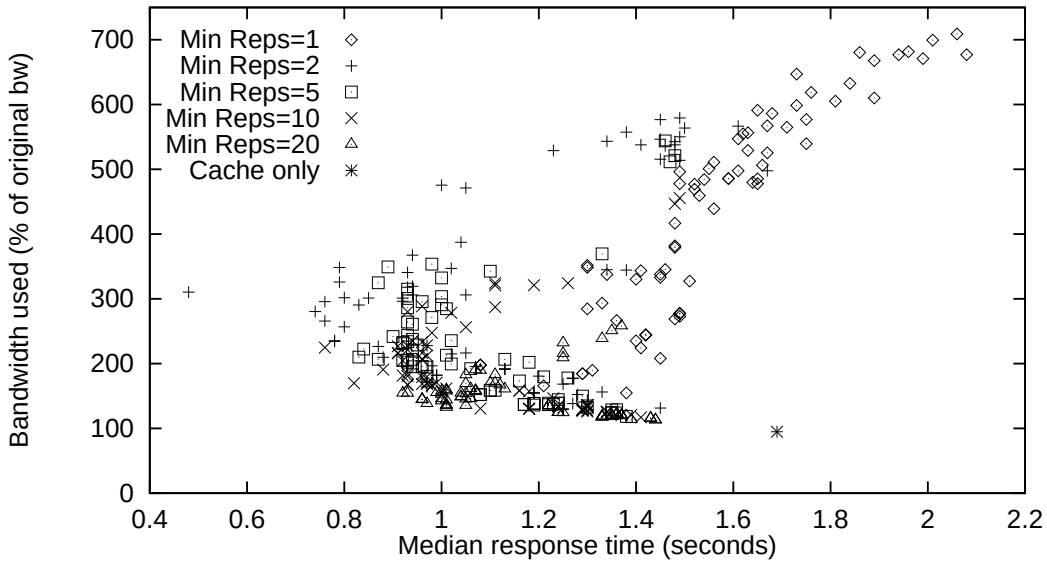

Figure 9.7: Scatterplot comparing median response time versus bandwidth consumed using the SSDC trace, highlighting the minimum repetitions required.

In Figures 9.6 and 9.7 we re-examine the median response time and bandwidth used, but in the context of the parameter settings for the minimum confidence and repetitions required, respectively. While the various confidence values are scattered throughout Figure 9.6, there are definite clusterings in Figure 9.7. The latter shows that the minimum repetition setting should certainly be larger, rather than smaller. The

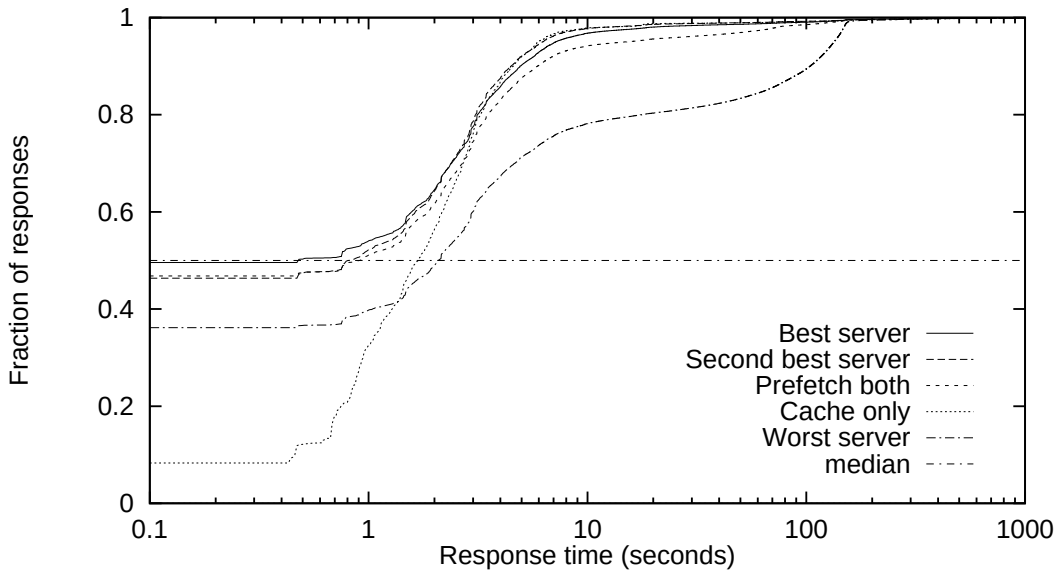

Figure 9.8: CDF of simulated response times for various configurations using the SSDC trace.

best points (along the lower left edge) are primarily those with minimum repetitions of 10 or 20. Thus, it is obvious that better predictions are made when the model has seen the pattern many times.

In Figure 9.8, we plot the cumulative distribution function of the response times for various configurations on the SSDC trace. At the bottom we show the caching-only configuration. The top curve is occupied primarily by the configuration getting the very lowest median response time. However, while it does have a better hit rate and median, its curve ends up being slightly worse than our "Second best server" for response times greater than two seconds. For comparison, we note that the really poor performing configurations, such as our "Worst server" shown here, can perform much worse than the caching-only configuration when they overwhelm limited resources, causing congestion and delays.

While not plotted graphically, we can compare the summation of all response times simulated for each configuration (e.g., the total waiting time experienced by the simulated users). We find that in comparison to the caching-only configuration, the worst configuration cost users an additional 344% in delays, while the best saved 8.15% in waiting time. Surprisingly, the second best saved users more than 15% in delays, while

transmitting only 80% additional bytes compared to the caching-only configuration. This is explained by the "best" server's poorer response times for documents that were large, causing suboptimal performance after about 80% of responses in Figure 9.8.

9.4.2 Web server log: NASA

9.4.2.1 Initial experiments

In addition to the SSDC trace above, we provide prefetching analysis from a sample (12 days, 858123 requests) of a second Web server trace, from NASA [Dum95]. For this simulation using NCS, we assume a server connected at T1 speeds (1.5Mbps, 10ms one-way latency) and clients identical to that of the SSDC simulations (approximately 120kbps and 40ms one-way latency).

Again, the client will have an additional 2MB cache. All prediction models will use n-grams between 1 and 3 requests long, and the scenario must be seen at least 10 times before a prediction is made with a probability of at least .1. Each predictor will be permitted to make just one prediction.

Table 9.3 summarizes the performance of the various configurations. It provides the mean and median response time for each approach, as well as the bandwidth required, and the fraction of requests served by prefetched content, plus the fraction of requests that prefetched content that was unusable. Using the same experiments, Figure 9.9 shows the distribution of client-side response times. The bottom-most line corresponds to the performance seen without any prefetching (just caching). As a result, it has the worst of all performances with a mean and median response times of 1.62s and .47s while using 90.8% of the original bandwidth used without a cache. Client-based prefetching

Approach	Response time		Fraction of original bw	% reqs pref'd successfully	% reqs pref'd unsuccessfully
	mean	median			
Prefetch both	1.37s	0.19s	131.7%	38.4%	46.6%
Prefetch server	1.38s	0.20s	129.0%	38.0%	48.7%
Prefetch client	1.61s	0.46s	91.0%	1.0%	0.2%
Cache only	1.62s	0.47s	90.8%	0%	0%

Table 9.3: Summary statistics for prefetching performance when one prediction is made with the NASA log.

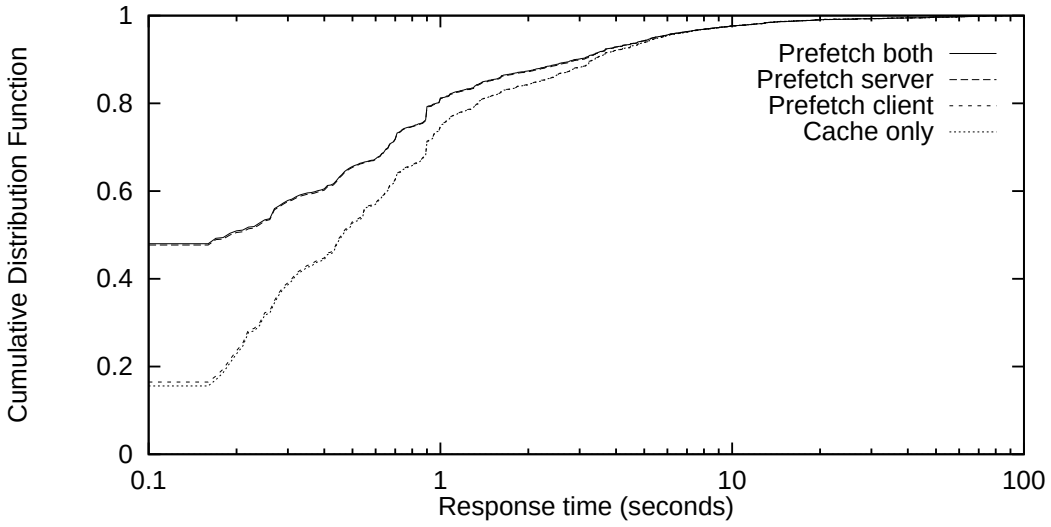

Figure 9.9: Cumulative distribution of client-side response time when only one prediction is used in the NASA trace.

helps slightly, reducing mean and median response times to 1.61s and .46s, at a cost of an additional 1% of bandwidth. Server-based prefetching helps tremendously; median response time drops by more than half to .2s, and mean by 15% to 1.38s, but at a cost of 29% more than the original bandwidth. The combination of client and server predictions results in only slightly better performance — mean response time stays the same, and median drops to .19s. Bandwidth usage increased slightly to 131.7%.

9.4.2.2 Parameter studies

We now examine whether performance changes for simulations based on the NASA trace in the same way that performance changed for SSDC-based simulations.

The initial NASA configurations tested above are limited. In fact, with ideal parameter settings, it is possible to serve more than half of the requests from client cache. This makes the median response time metric useless for comparative purposes (but potentially great for marketing — "reduced median response time by 100%!"). Instead, in Figure 9.10, we compare the 75^{th} percentile response time to the mean, again for the various configurations of our prefetching system that uses server hints for the NASA trace. Unlike the SSDC data, it shows that many configurations are able to improve both 75th percentile response times and mean response times. However, we note again

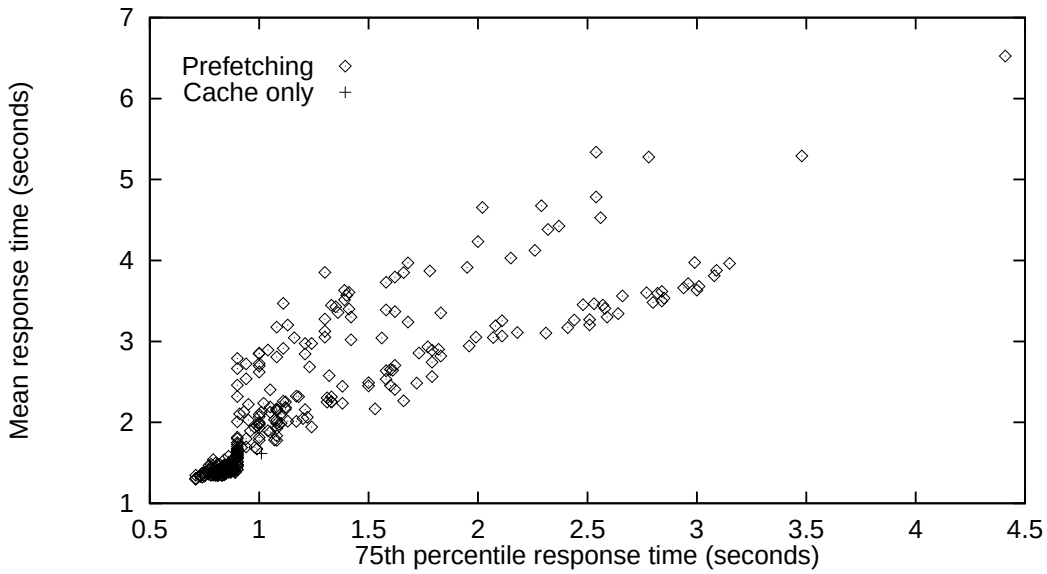

Figure 9.10: Scatterplot comparing 75th percentile versus mean response times for various configurations using the NASA trace.

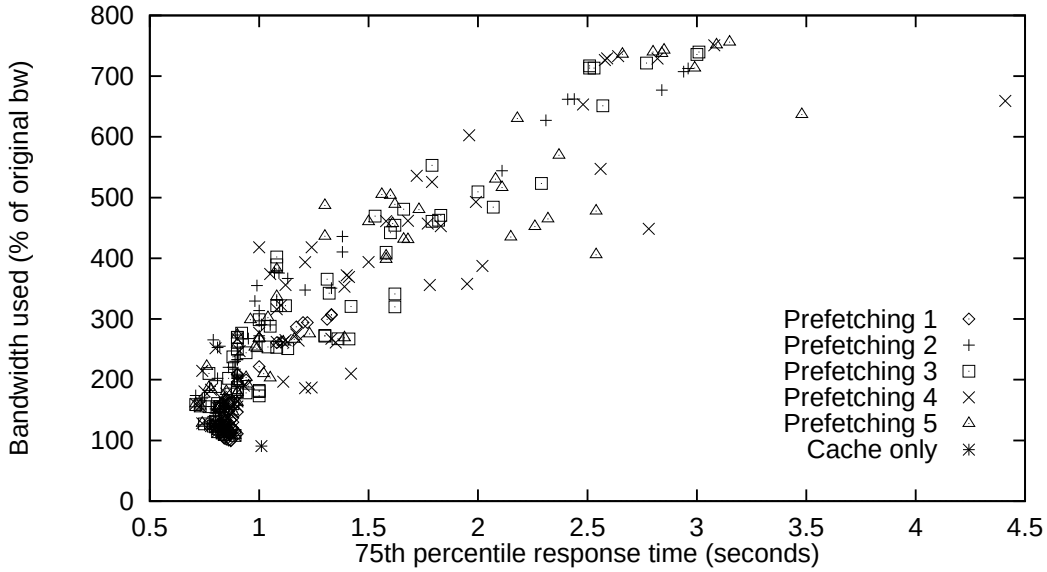

Figure 9.11: Scatterplot comparing median response time versus bandwidth consumed using the NASA trace, highlighting the number of predictions permitted.

that poor configurations can hurt in both dimensions.

We compare the cost of prefetching to its benefits on the 75^{th} percentile response times in Figure 9.11. We also distinguish among the number of predictions permitted. Like the SSDC experiments, it is apparent that many prefetching approaches are capable of reducing the median response time while using no more than twice the bandwidth of

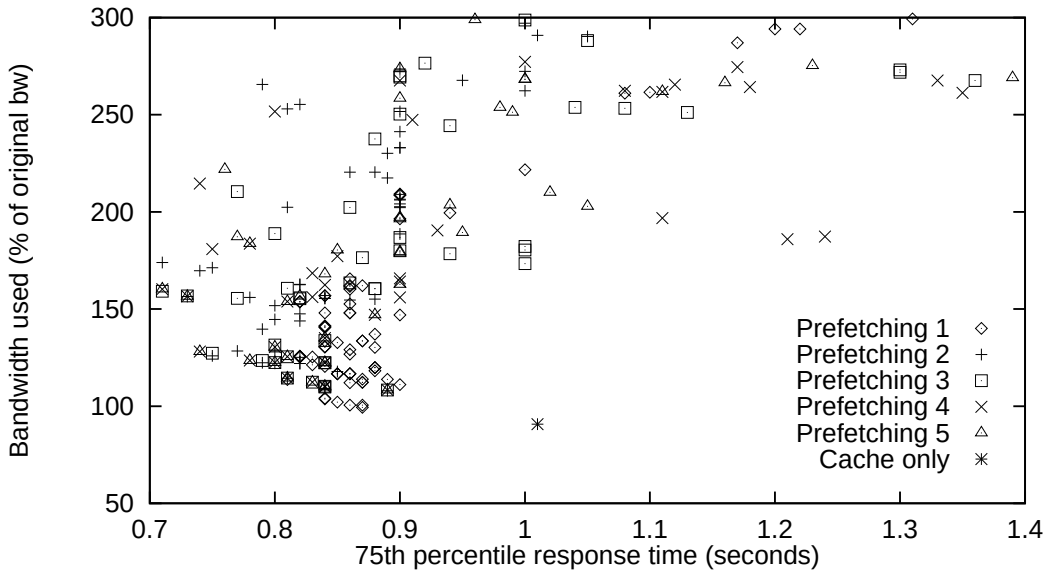

Figure 9.12: A magnified view of the best points in a scatterplot comparing median response time versus bandwidth consumed using the NASA trace, highlighting the number of predictions permitted.

the original trace. The figure also demonstrates that many configurations are able to cut more than 25% off the caching only response times. While not shown, the caching-only configuration had a median response time of 1.615 seconds, and more than half of the configurations tested achieved hit rates of greather than 50%, making their median response-times zero.

Observations can also be made based on the number of predictions permitted. While a single best prediction does improve response time, they do not perform as well as successful configurations using a larger number of predictions. In constrast, configurations using two, three, four, or five predictions all succeed in getting significant improvements in latency without bandwidth requirements getting too large (seen better in Figure 9.12), but are also well-represented throughout the scatterplot.

The caching only configuration uses 90.8% bandwidth to provide a 1.01 seconds 75^{th} percentile response time. The best configuration achieved a response time of .71 seconds at a cost of 168.4% of the original bandwidth. It made up to five guesses using n-grams of length two, with a minimum of 5 repetitions and minimum probability level of .1.

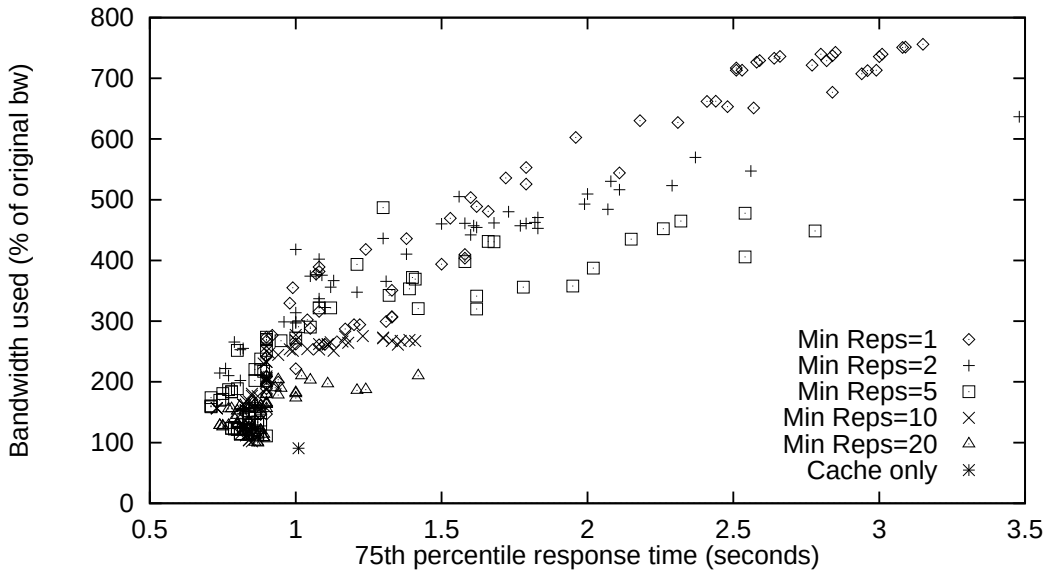

Figure 9.13: Scatterplot comparing median response time versus bandwidth consumed using the NASA trace, highlighting the minimum repetitions required.

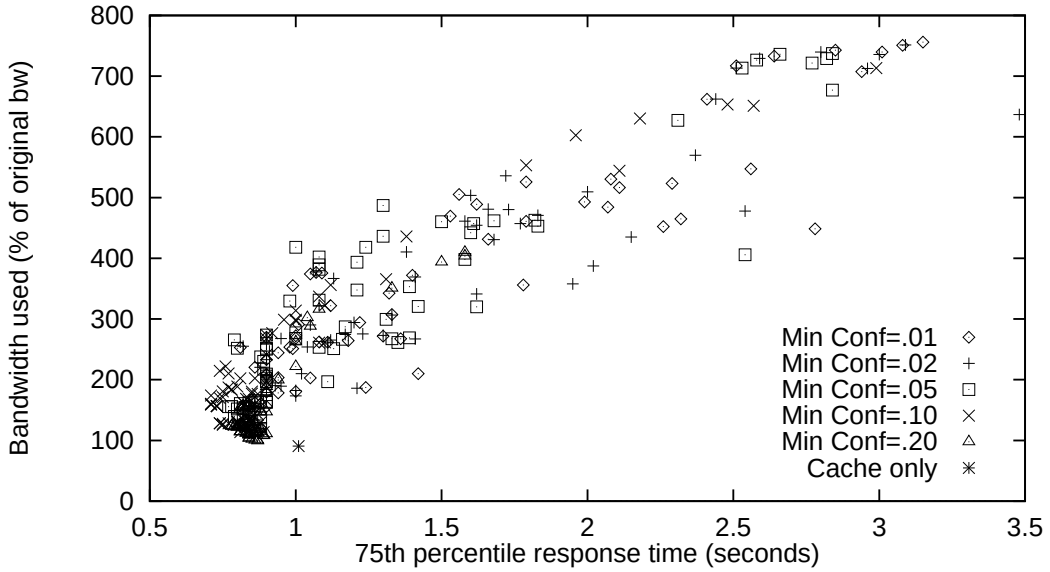

Figure 9.14: Scatterplot comparing median response time versus bandwidth consumed using the NASA trace, highlighting the minimum confidence required.

While the SSDC traces showed significance only for the number of repetitions, Figures 9.13 and 9.14 both show visible clustering. Typically the best performing points have minimums of at least five repetitions, and .1 probability.

In Figure 9.15 we plot the cumulative distribution function of the response times for extreme configurations on the NASA trace. Unlike the SSDC trace, we can see

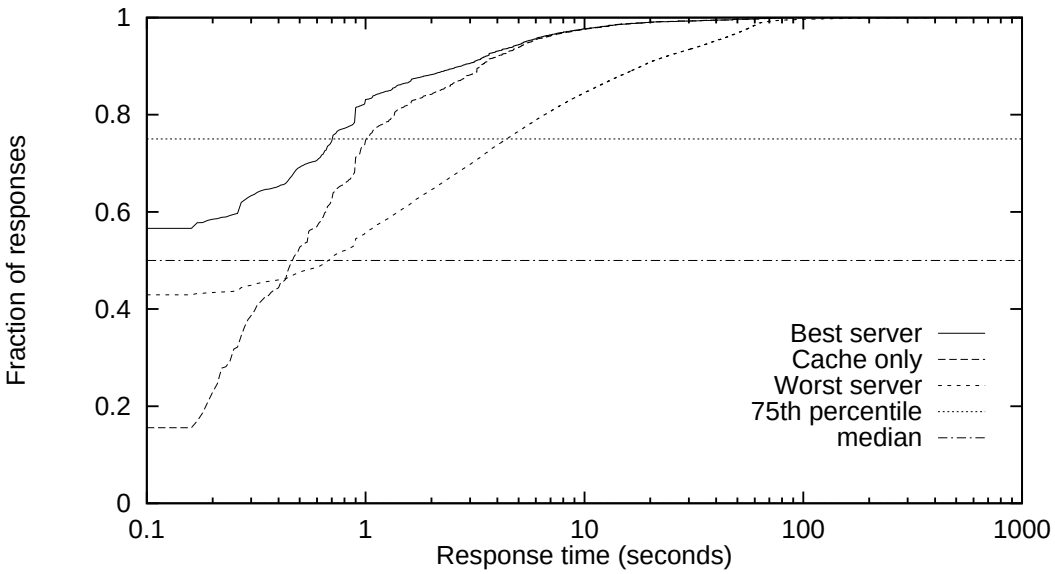

Figure 9.15: CDF of simulated response times for various configurations using the NASA trace.

improvements in response times for close to 95% of all responses. As in the SSDC case, the worst-performing configuration performs significantly worse for essentially all non-cached responses (which account for more than 40%).

Calculation of the sums of all response times simulated for each configuration demonstrates that in the worst case, users will wait an additional 304%, while in the best case, users will wait 19.7% fewer seconds while transmitting 85% additional bytes compared to the caching-only configuration.

9.4.3 Proxy server log: UCB

As described in Section 4.4.1, the UCB Home-IP Trace records the requests made by relatively low-bandwidth dialup and wireless users of the UC Berkeley Home IP service.

Unfortunately, straightforward simulation of history-based prefetching using the UCB trace finds essentially no improvement in client-side response times (in which the median is 1.91s). Plotting the CDF of the prefetching and non-prefetching implementations shows just a single overlapping curve. This result is initially perplexing, and so we consider it further below.

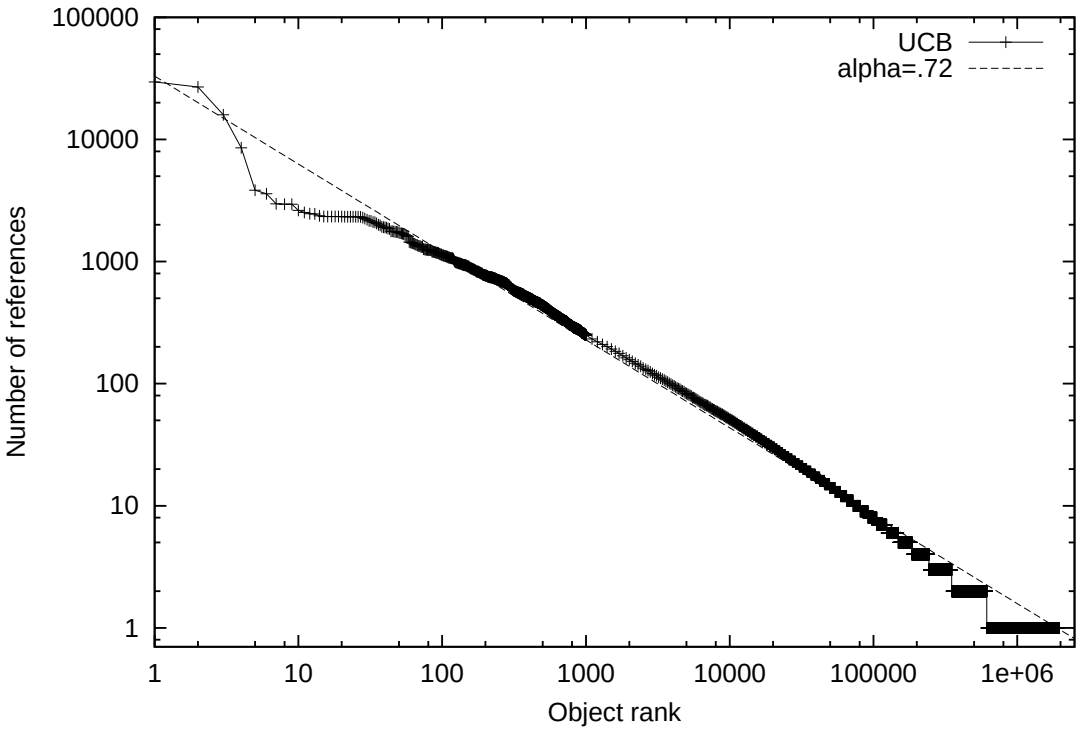

Figure 9.16: Frequency of request versus request rank for the UCB Home-IP trace.

9.5 Proxy Log Analysis

This chapter has explored two combinations of predictions — combining content and history, and combining multiple views of history. In both cases we demonstrated by simulation situations in which combinations can provide better performance than either prediction source alone.

However, the combination of multiple views of history do not appear to be effective when tested on the UCB Home-IP usage trace. While the Home-IP trace is large, containing millions of requests, it also has a large number of clients and a large number of servers (more than 40,000) represented. As a result, the number of requests per server is relatively low — much fewer than the server-specific traces we have discussed so far. Highly referenced servers in this trace contain between 20,000-100,000 references, but there are few such servers. Thus the primary difficulty is a lack of data. Without enough references to individual pages, they will not be predicted in the future.

A comparison of some of the characteristics of the UCB trace with others may be

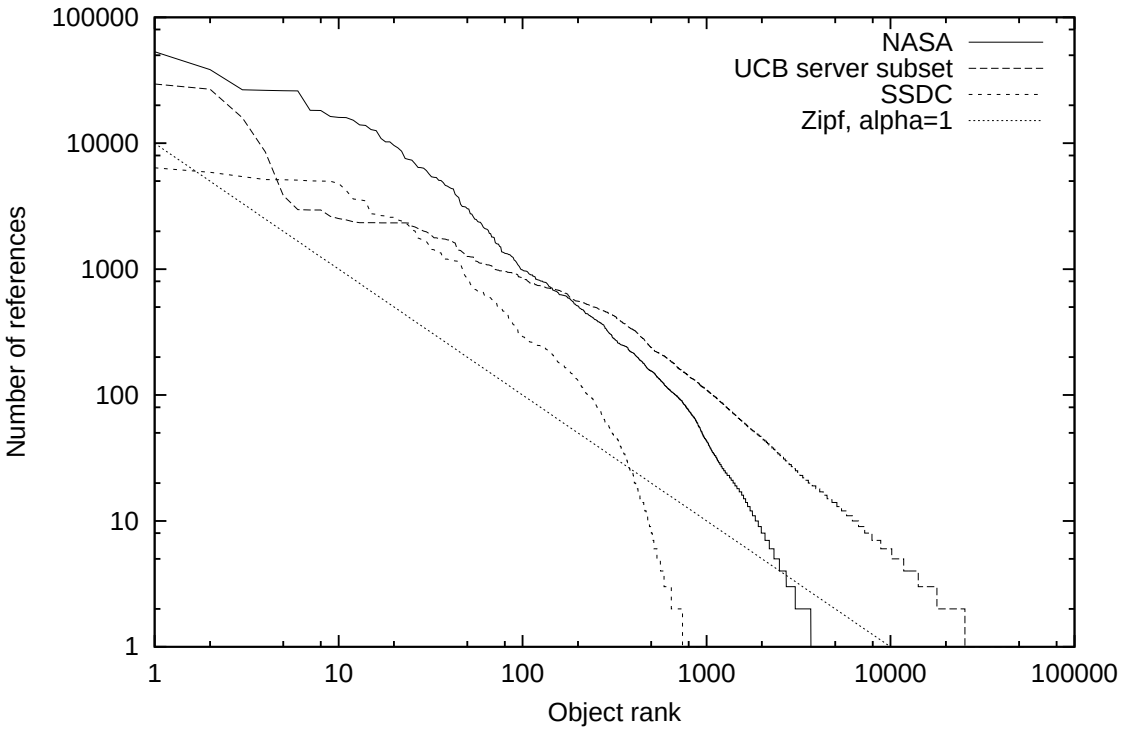

Figure 9.17: Frequency of request versus request rank for various server traces.

helpful. Figure 9.16 shows the number of times a particular request was made versus the rank of that request (where a rank of 1 signifies the most popular) for the UCB proxy trace. Figure 9.17 shows the same for a number of server traces as well as subsets of the UCB trace. A CDF view of all traces can be found in Figure 9.18. The UCB trace is a good fit to the Zipf-like model, in which the relative probability of a request for the ith most popular page is inversely proportional to $1/i^{\alpha}$. However, the server traces are less good of a match. If a Zipf curve were matched to them, α would be higher than typical,[1] and in general they do not as well match the log-log line typical of Zipfian distributions. In fact, other work has found that not all Web traffic has a Zipf-like request popularity with α less than 1 [PQ00]. While not identified as such, the effect is also visible in plots in earlier work [AW97].

One aspect that is apparent is that the Web server traces appear to have fewer infrequent requests. The least frequent request (a reference that appears once, often

[1]For example, [MWE00] shows proxy cache logs with values for α between .74 and .84 with good fits to rank vs. frequency of reference plots, and [BCF+99] examines a different set of proxy traces with α values between .64 and .83.

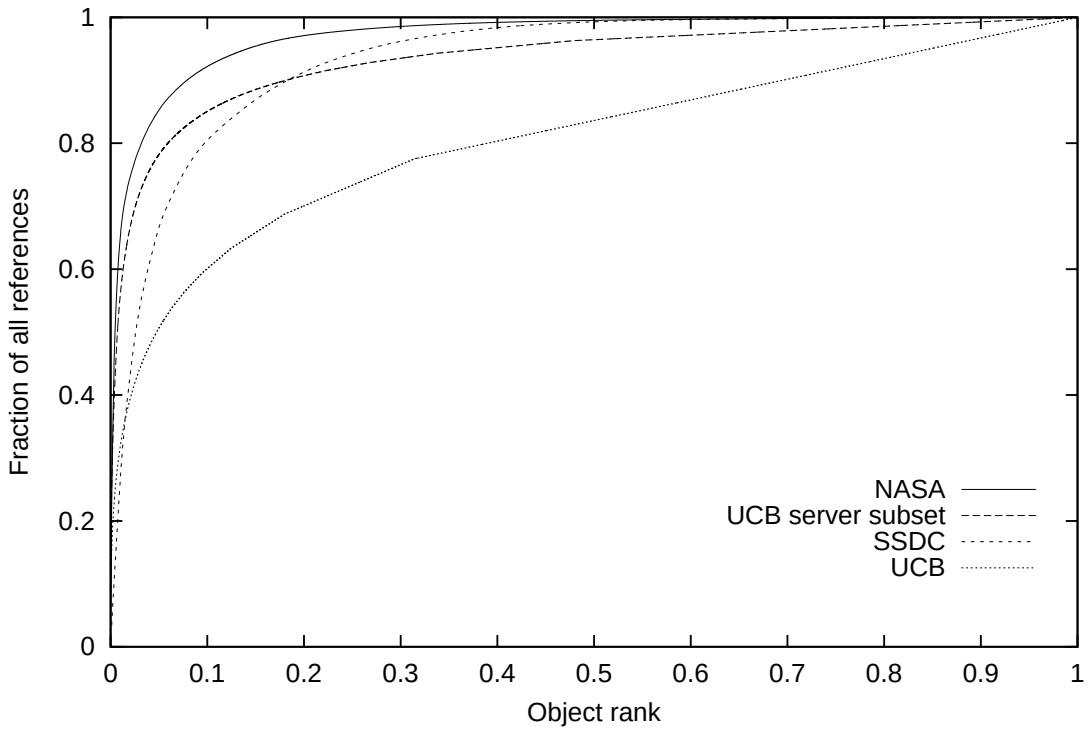

Figure 9.18: Cumulative distribution function of frequency of request versus request rank for various Web traces.

called a one-timer) is much more prevalent in the UCB trace than the others. One timers account for 22.5% of all references. In fact, pages referenced less than 10 times account for more than half of all references (51.8%). In contrast, pages referenced less than 10 times in the NASA trace account for just 1% of all references, and in the SSDC trace account for even less — .6% of all references. In the UCB servers subset, this accounts for 12.9% of all references. For a set of proxy traces, Mahanti *et al.* [MWE00] reports one-timers consisting of between 17.8% and 47.5% of all requests.

Figure 9.19 depicts the cumulative distribution functions of the frequency of page references (that is, the ordered list of the number of distinct times some object was referenced versus the total number of requests for pages in that category). It shows, for example, that all three UCB-based traces (the original UCB trace, a 100,000 request UCB subset, and the UCB server subset) have a larger fraction of requests for rare objects than the standard server-based traces (NASA and SSDC).

The more typical analysis is concerned primarily with the number of objects that

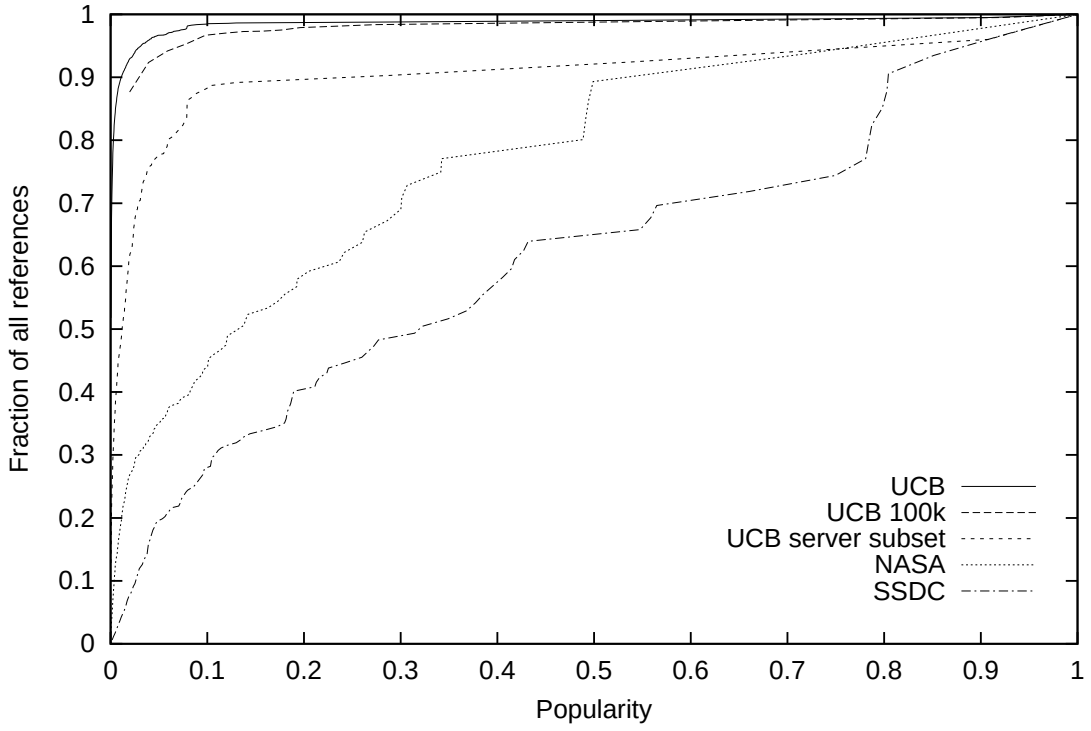

Figure 9.19: The CDF of frequency of page references, ordered by increasing frequency.

account for a large percentage of requests (thus ideal for caching). In the NASA trace, the top-ranked 1% of references account for 65% of the trace. For the SSDC trace, it is 80.5%. Similarly, Doyle *et al.* [DCGV01] describes very large traces from IBM's web site in which a small number of popular objects account for an unusually large percentages of requests. In a 1998 trace, the top 1% of objects generate 77% of all requests. In a 2001 IBM trace, the top 1% accounted for 93% of all requests. In contrast, the top-ranked 1% of UCB references account for only 31.8% of all references.

Therefore, it appears that the popularity of documents in proxy traces are quite different from that of Web server traces. The large fraction of one-timers and other rare pages make an informed prediction difficult, if not impossible.

While the UCB trace as a whole makes effective prefetching difficult, there are still some servers represented within it that have a reasonable number of requests made to them. Thus we selected the top-twenty servers (that is, those that served the most requests). These twenty servers (out of more than 40,000) handled a total of 12.5% of

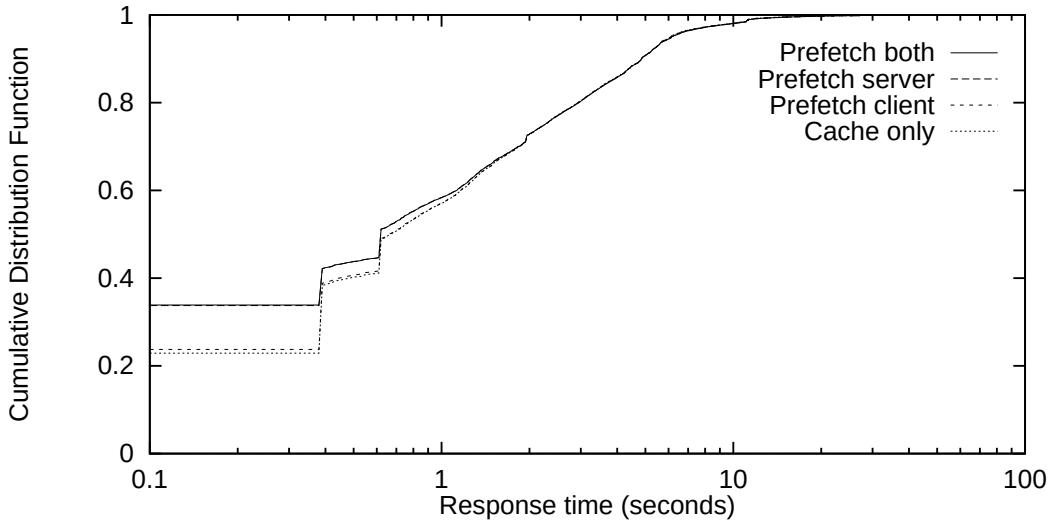

Figure 9.20: CDF of response times for various prefetching configurations using the UCB server subtrace.

all requests in the trace. However, the trace examined still only represents 12 days of traffic, which is relatively short time period for clients to learn and benefit from past experience.

NCS was used to simulate client response times under a variety of prefetching scenarios, much as we did with the server logs earlier in this chapter. The results are shown in Figure 9.20. Using the server's prediction model (either for server alone, or in combination with the client's model), we get a improvement in median response time of 50ms. While not reflected in the response times, the client model does provide increased request hits — without prefetching, 170941 requests were satisfied by the 2MB client cache. Client prefetching increases this to 177441 hits. When added to server prefetching, the benefit is much smaller — an increase from 251954 to 252807.

As a result of relatively strict limits on when predictions are made, the number of requests did not affect bandwidth utilization significantly. Even when prefetching based on client and server predictions, bandwidth utilization was just 16.4% higher than the original trace, and just 24.2% higher than just passive caching with the 2MB client cache.

9.6 Summary

This chapter has explored two combinations of predictions — combining content and history, and combining multiple views of history. In both cases we demonstrated that the combination of predictions may provide better performance than either alone.

We also explored performance as prediction parameters were varied. Simulation experiments with NCS in this chapter demonstrated that prefetching on server traces could cut median response times in half or better, and reduce overall client-perceived response times by 15-20%, by transmitting an additional 80-85% bytes over what would be transmitted by the equivalent caching-only configuration.

From this we found that the minimum repetitions threshold was an important value for getting high quality predictions, while the minimum confidence threshold was much less useful. The number of predictions and the size of the n-grams used for matching was less important — arguably useful points were found with various values of those parameters, although a single prediction did appear to be less helpful than two or more.

However, we've also used this chapter to point out some characteristics of when prefetching will not be particularly effective. In particular, when rare objects comprise a large fraction of all requests (as they do in the UCB proxy trace), history-based predictions are at a severe disadvantage. While intuitions and some data suggest that longer traces (i.e., building a model over a longer period of time) will help, the real answer will be to use multiple prediction sources for prefetching. Content-based prediction is one answer, as is the use of server hints when they are available.