Using machine learning techniques for intelligent web content prefetching

Badrish Chandramouli
Sita Iyer
Department of Computer Science, Duke University
{badrish, sita}@cs.duke.edu
December 05, 2003

1. Introduction

A typical web browsing session consists of the user opening the index page of a particular website, and then going through a sequence of pages using hyperlinks. Generally, there is a delay of some time (known as "think time") before the user clicks on the next hyperlink in the sequence. A browser extension could constructively exploit the think time by intelligently prefetching the most likely pages that the user is expected to click into. If the prefetching is successful, the user would see excellent response time while surfing, even when on a slow connection.

2. Project description

This project evaluates (by simulation using traces) different machine learning strategies that could be used for effective web prefetching. The basic idea is for each web server to maintain state that can assist in deciding what pages to prefetch.

Our project deals with the server side techniques that can aid in prediction of good prefetch candidates. The following are the techniques we considered:

2.1. Maximum likelihood estimates of transition probabilities

In this technique, the server maintains a Markov chain, with each state being a unique HTML page on the web server. The link between states *i* and *j* in the chain would correspond to the probability of traversing to link *j* from page *i*. Using this model, we can apply MLE to predict the transition probabilities of the Markov chain.

Assume that a browser requests page *i* from the server. In response, the server sends not just page *i*, but also a set of most likely pages (and their corresponding transition probabilities from *i*) that the user could traverse to, from page *i*. The browser utilizes this information and performs selective prefetching based on the available bandwidth and think time.

2.2. Single-layer perceptron

In this alternative, the web server maintains one perceptron per HTML page. The perceptron for a page *i*, would take as input a bit vector of size *N*, which indicates whether the last *N* visits to pages that contained *i* as a link actually resulted in a click-through to page *i*. We could then apply gradient descent to learn the nature of click-throughs to page *i* by users.

When the browser requests a particular page, the server would compute the outputs of the perceptrons for each link on that page and thus determine the candidates for prefetching. This information would be sent to the client, along with the requested page. As before, the browser could use this information as required.

Current state-of-the-art browsers and web servers do not have provision for prefetching web content, and thereby do not exploit think time and idle client bandwidth. Our techniques outlined result in a good accuracy of prefetch prediction, and translate to better response time for end users, especially those on slow connections.

3. Workload

We performed trace-driven simulation of the techniques proposed in the previous section. We used the traces available at http://ita.ee.lbl.gov/html/contrib/ClarkNet-HTTP.html. These traces contain two week's worth of all HTTP requests to the ClarkNet WWW server. ClarkNet is a full Internet access provider for the Metro Baltimore-Washington DC area. We used one week's traces for training and the other week's traces for testing. Both traces were parsed by a parsing tool that we wrote in order to extract the relevant information such as the host making the request, time the request was made, URL (page) requested, and the size of the page requested. We only considered HTML page requests and only those requests which did not generate an error response from the web server.

4. Methodology

4.1. Modeling a Markov chain using maximum likelihood transition probabilities

We modeled the web server state by using a Markov chain, where each state represents a URL and a transition from one state to another represents a click through from the first URL to the next. The state transition probability then directly maps to the probability of clicking on a particular hyperlink (to traverse to the next state) when browsing the web page (current state). We first wrote a program that operates on the training data set,

calculates the transition probabilities from one page to another and generates the MLE matrix using the training data. This matrix represents the Markov chain that we modeled. Transition probabilities were calculated not just considering the transitions of different hosts, but also considering multiple browsing sessions by the same host. The fact that there were indeed multiple browsing sessions by the same user was decided based on a threshold for the difference in times between two consecutive accesses by the same host.

In order to test the effectiveness of this technique, we used the matrix of transition probabilities to assist in prefetching the most popular clicks from a given page in the test data. Apart from investigating the usefulness of pure prefetching, we also studied the effect of using a cache to store the prefetched data which could be used for future accesses too and not just immediate ones as in the case of a pure prefetch. Of course, each cached page was allowed to remain valid only for a certain timeframe after which it would be evicted lest the users get stale data. Our observation was that prefetching coupled with the use of a cache led to the best results though prefetching by itself gave tremendous gains in terms of user response time.

Our experiments and results prove the same. We experimented with various sizes of cache, prefetch and user internet connection speed. The think time of the user was obtained from the difference in time between two consecutive accesses by the same user.

4.2. Modeling the system using neural networks

We designed a system where there would be one neural network per URL. The n inputs to the perceptron corresponded to the last n accesses to pages containing that URL as a hyperlink. Each input was 0 or 1 based on whether the URL was clicked when the user was on that page. An output of 1 indicated that the page needs to be prefetched and 0 indicated that the page did not need to be prefetched. We used the training data to train the perceptrons, and then applied the neural network to predict prefetches in the test data. The results of this endeavor were not as pronounced as those obtained by the Markov chain probability estimations, and were comparable to a random prefetch policy. The reason for this is that the last *n* accesses to pages containing a URL as a link and the chances of a clickthrough to that page did not correlate to the probability of that page being accessed from the current page.

5. Experiments

The following are the internet connection speeds (in kbps) that we simulated in order to test the effectiveness of prefetching: 33.6, 40.0, 44.0, 50.0, 56.0, 64.0, 80.0 and 100.0. Beyond 100 kbps, prefetching did not have much impact, as even without prefetch the surf time was minimal. Hence we have not included the results beyond 100 kbps.

For each connection speed, we calculated the average surf time for the following cases:

1. No prefetch and no cache i.e. using the state-of-the-art technology.
2. Prefetch only (no cache) and allowing a maximum of 3 pages to the prefetched
3. Prefetch (maximum up to 3 pages) and a cache that can hold a maximum of 4 pages
4. Prefetch only (no cache) and allowing a maximum of 4 pages to the prefetched

5. Prefetch (maximum up to 4 pages) and a cache that can hold a maximum of 4 pages

6. Experiments 4 and 5 were repeated with cache sizes of 5 and 6 pages.

7. We compared a 33.6 kbps connection that used no prefetching, pure prefetching and prefetching coupled with caching with a 56 kbps and 64 kbps connection that used none of these enhancements. The purpose of this experiment was to test weather or not a 33.6 kbps connection with our enhancements can match the performance of higher speeds thereby benefiting the users on slower connections.

8. We used the ANN with input length 4 and performed experiments 1-4.

6. Results

Figure 1 demonstrates the performance of the different schemes under various connection speeds for a cache size of 4 pages.

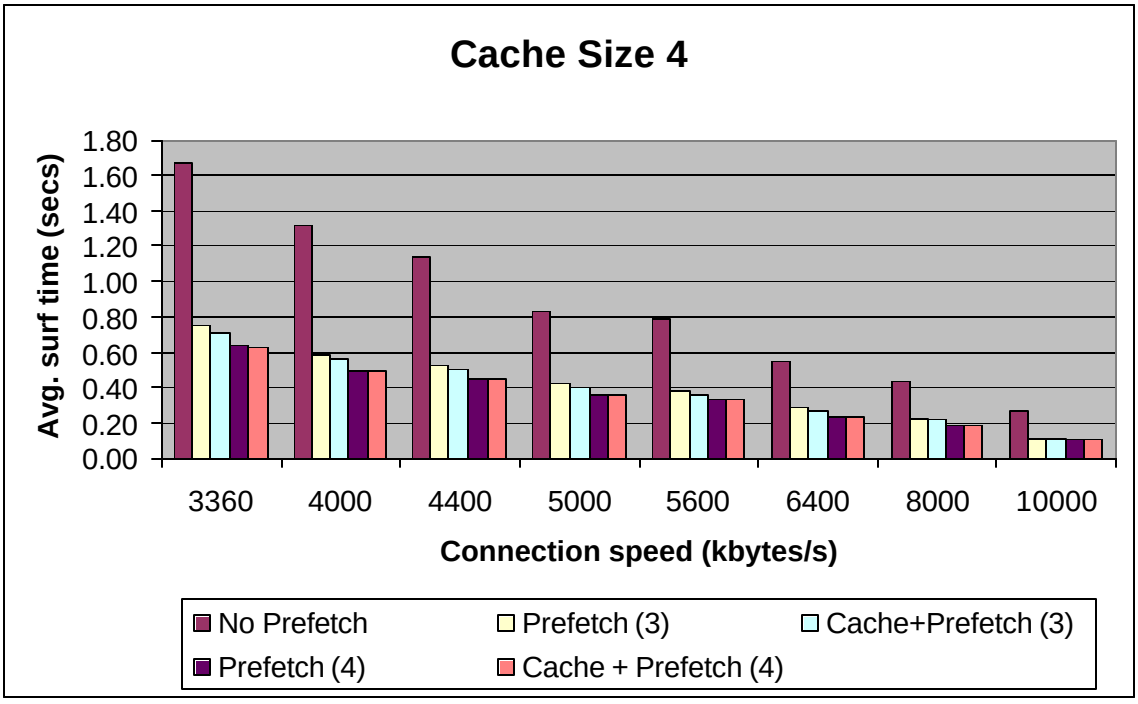

Figure 1: Average surf time for various connection speeds using different techniques with a cache size of 4 pages

We see that for a given connection speed, the average surf time decreases as we move from no prefetch to pure prefetch to prefetch with cache. We also see that purely prefetching 4 pages does better than prefetching 3 and using caching. This is because the rather small cache size is unable to improve the performance beyond that achieved by prefetching 4 pages.

Figure 2 demonstrates the performance of the different schemes under various connection speeds for a cache size of 5 pages. The results are somewhat identical to that obtained in the previous case except that the use of a large cache results in lower average surf time as can be seen from the third and fifth bar for each connection speed (where caching is used). When compared to the previous graph, we observe that the increase in the cache size by just one page leads to better performance.

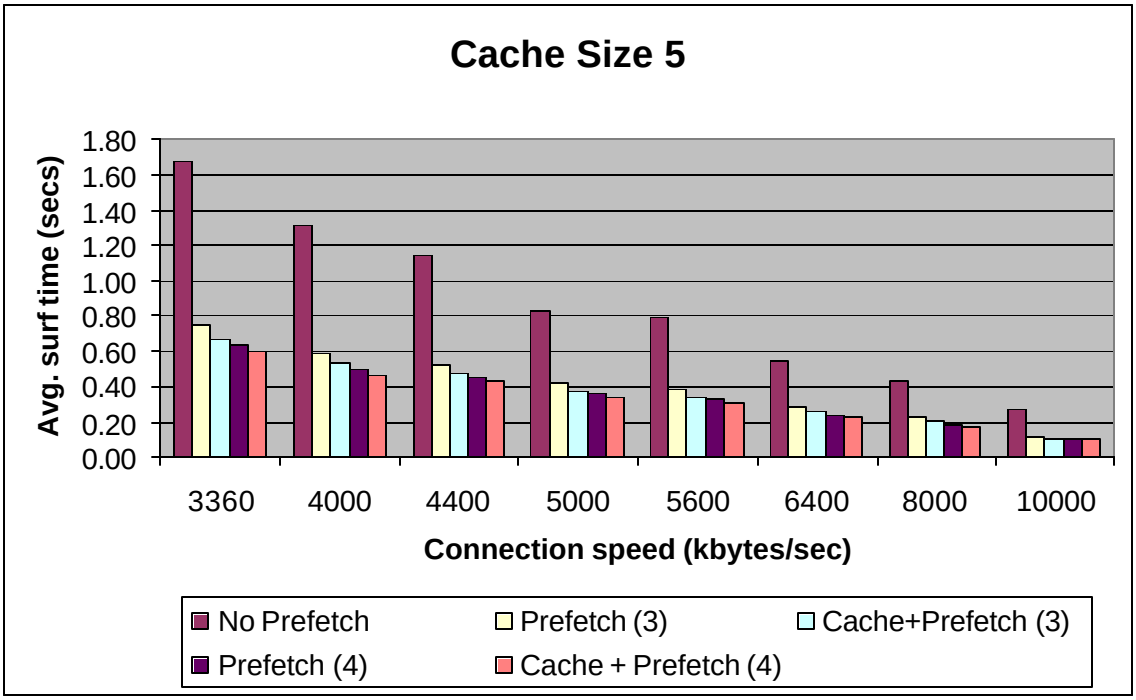

Figure 2: Average surf time for various connection speeds using different techniques with a cache size of 5 pages

Figure 3 demonstrates the performance of the different schemes under various connection speeds for a cache size of 6 pages. Here we see that prefetch of size 3 along with caching outperforms prefetch of size 4 as opposed to the prior two figures. This is due to the bigger cache being used. Once again increasing the cache size exhibits better performance as can be seen by comparing the third and fifth bar with the previous graphs

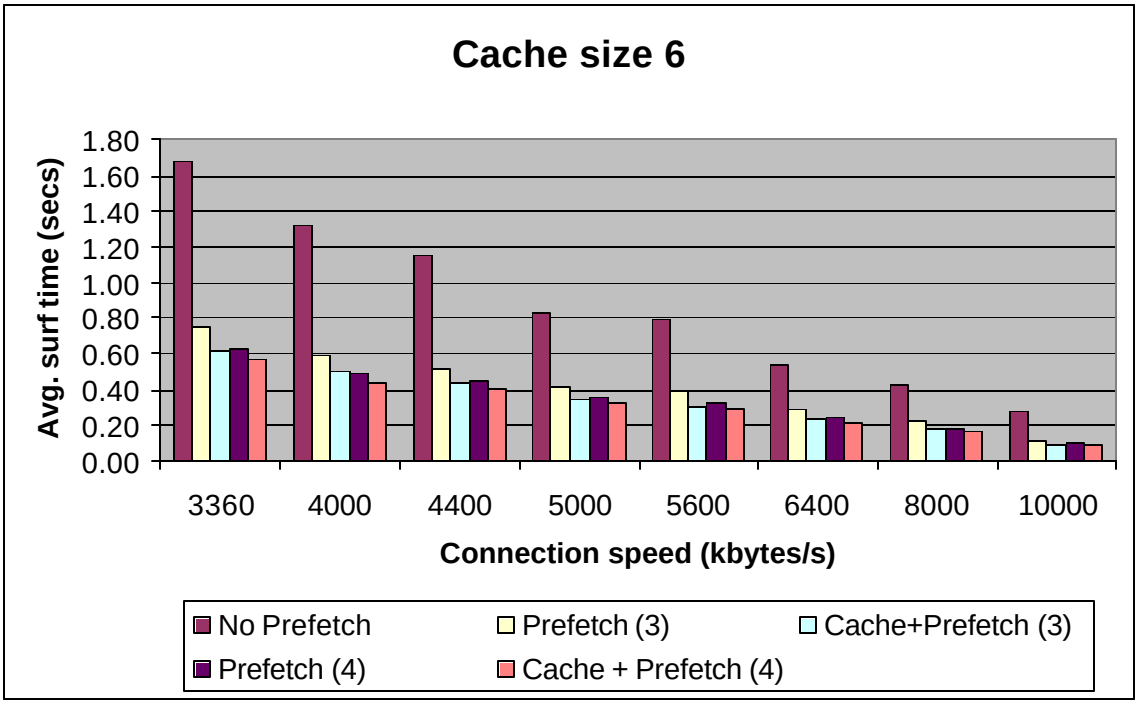

Figure 3: Average surf time for various connection speeds using different techniques with a cache size of 6 pages

Figure 4 shows how a 33.6 kbps connection using just pure prefetching can start to outperform a plain 56 kbps connection and almost equal the performance of a 64 kbps connection. The two horizontal straight lines in the graph correspond the average surf times of the 56 and 64 kbps connection without use of prefetching or caching.

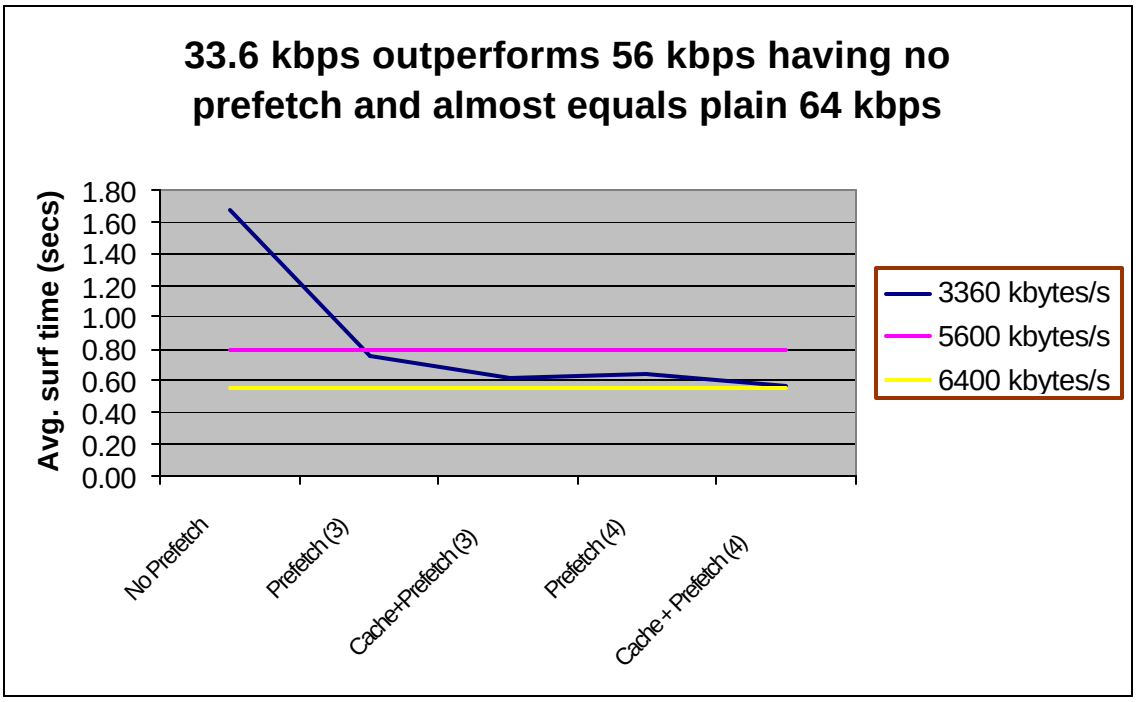

Figure 4: Avg. surf time of a 33.6 kbps connection under various schemes in comparison with that of 56 and 64 kbps connection under no prefetching and no caching.

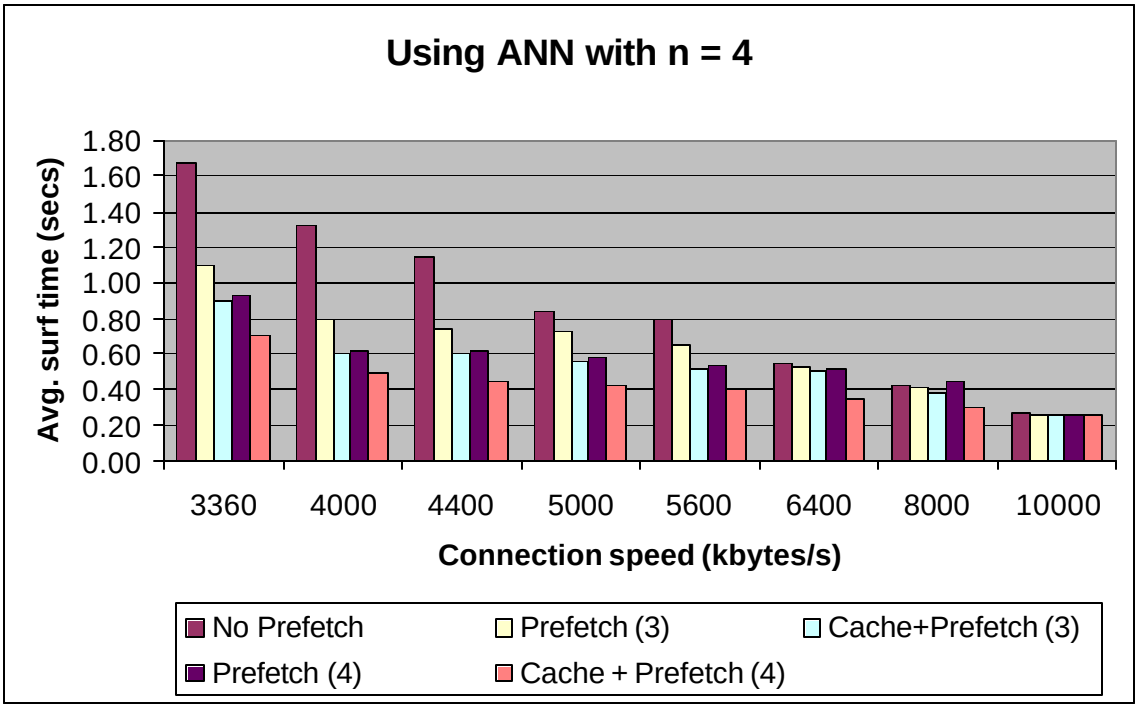

Figure 5: Average surf time for various connection speeds using different techniques with a cache size of 5 pages and using the ANN methodology

Figure 5 shows the outputs for the perceptron method. We note that the improvements are not as much as those obtained using Markov chain models.

7. Conclusions

We find that prefetching based on maximum likelihood estimates of clicks is an effective way to reduce the latency perceived by users on slow (dial-up) connections. This reduction in latency can be further improved by using a cache to store the prefetched pages and use them in future requests to those pages. Pure prefetching alone results in about 50% reduction in the average surf time. This directly translates to a better response time to the end user and gives the user the benefits of a faster connection.

While the neural networks did show an improvement over the state-of-the-art, they did not live up to expectations. This is probably because of the specific choice of inputs that we studied. As part of the future work on this project, we would like to explore different alternatives to come up with features that would give a better improvement in performance.

References

[1] Bishop, Christopher M. Neural networks for pattern recognition. Oxford Press.

[2] R. Durbin et al. Biological sequence analysis. Cambridge university press.

[3] http://ita.ee.lbl.gov/html/traces.html - Internet traffic archive