EPiC Series in Computing
Volume 58, 2019, Pages 203–209
Proceedings of 34th International Conference on Computers and Their Applications

Web Request Predictions

Randy Appleton, Ben Slater, Connor Laitinen, Luke Ammel, Cody Malnor
rappleto@nmu.edu
Northern Michigan University

Abstract

If one could predict future web requests, it would be possible to make the web much faster. One could fetch web resources before they are needed. When the human user clicks on a link, the needed data would already have been downloaded.

We have created several algorithms that attempt to predict future web requests based on past histories. Our research evaluates and compares these prediction algorithms against real histories of web usage. Prediction algorithm results are compared based on correct predictions, erroneous predictions, and prediction rate.

Some algorithms make predictions rarely but accurately, while others may predict more often but with less accuracy. To take full advantage of this, we combine multiple algorithms and use different voting strategies to determine the best prediction.

1 Introduction

Users want the web to be fast.

There are two factors when determining browsing performance: bandwidth and latency. In many environments, it is not possible to create low latency, high bandwidth networks. For example, in areas receiving satellite Internet, latency is high but bandwidth is reasonable. When providing Internet connectivity by satellite to isolated consumers, long latencies are simply unavoidable because of the long distances involved, and the finite speed of light. Ground based, fast Internet connectivity is also not possible when traveling via airliner or ship. Even on the ground, many areas, including many poor areas of the world, are still not connected via DSL or cable-internet, let alone fiber optic connections.

If one could predict future requests, it would be possible to reduce latency and make the web significantly faster. The purpose of this research was to develop a predictor that can accurately predict what a user will browse next, thus reducing the appearance of latency and providing a faster web experience. Such techniques only recently became possible. By utilizing the capabilities of the recently published HTTP version 2.0, is it now possible for a web server to push soon to be needed resources to a connected web browser [1, 12,13]. By correctly identifying needed files and deliver them to the web browser before they are needed, perceived web speed will improve.

2 Previous Work

Previous studies have shown that prefetching is effective at reducing user perceived latency. Research shows that a combination of caching and prefetching doubles web performance compared to simple caching [2,3,4,5,6,7,8]. According to [3,4], a combination of web caching and prefetching can potentially improve latency up to 60%, whereas web caching alone can improve latency up to 26%[7].

Different studies show different levels of predictive success. It is difficult to directly compare predictors as they are usually evaluated against different workloads and measured using different metrics. This work compares several different prediction algorithms against one another using the same workloads and scored using the same metrics. Some of the algorithms presented combine various prediction algorithms using voting schemes to choose a result. Such ensemble predictors have been shown to perform better than their component parts [9].

3 Methodology

To test our prediction algorithms, log files from different web servers were collected. This data contained IP addresses and web sites for each request made to that server. Each IP address was made anonymous. The data used in our test came from two sectors, business and academic. The business data consisted of the log files of a company website while the academic data consisted of the log files of a server on our university's network. By gathering log files from various sources, we were able to represent the variations present in real-world data. All algorithms were tested against the same log files (about 200,000 requests) to make a fair apples-to-apples comparison. Below, we present data using the academic log files; the results from the business log files were very similar showing the general applicability of the results.

Code was written to test our predictors against the collected data. Our test program was designed to use our predictors to predict future URL requests using only the client IP address, and previous history of requests. The predicted URL was then compared with the actual future requests found in the log file. The test program scored the solutions on correct predictions, erroneous predictions, and prediction rate.

This methodology is similar to our previous work in [10].

Developing Prediction Algorithms

We tried many different approaches to our problem. Predictors based on many different ideas were developed. Results were compared to the results in our previous work [10]. Hoping to improve performance, we chose to develop ensemble predictors. A search of previous work showed that voting predictors, which combine the results of many different prediction algorithms to make a single prediction, often outperform their component parts [11]. This information became the foundation for developing voting-based prediction algorithms.

4 Prediction Algorithms

Each predictor was scored using several metrics:

- **PredictionRate** – How often did the algorithm make any prediction.
- **SuccessRate10** – How often was the algorithm's prediction within the next ten web requests. Sometimes a prediction algorithm will predict a file, and although that file is not the very next

request, it is requested in the near future. Prefetching based on such predictions is still productive and useful, and should count as a success.

- **WrongRate10** – How often was the algorithm's prediction not within the next ten web requests.
- **Accuracy10** – Percent of predictions which were within the next ten requests. Accuracy10 = SuccessRate10 / (SuccessRate10 + WrongRate10).

Clearly, a perfect predictor would have a successful prediction rate of 100%, and a wrong prediction rate of 0%, but no known predictor can achieve this.

Note that the successful prediction rate added to the wrong prediction rate is rarely 100%. Good prediction algorithms will often fail to make any prediction; for certain situations there is not enough information to make a prediction.

As described above, we consider Accuracy10 to be a better metric than simply predicting the next request. Our goal therefore is to make a prediction algorithm with both a high PredictionRate and a high Accuracy10.

In practice, there is often a trade-off between number of predictions made and the error rate. This trade-off ranges from making a few very predictions of excellent quality to making many predictions, of which each has only a moderate chance of correctness. Every correct prediction improves performance. Wrong predictions may or may not reduce performance, depending on how many resources were utilized to generate and send the unneeded data, and whether those resources were idle or in use. Sending unneeded data can impose costs for networks that bill based on data sent, such as satellite links or phone networks. Also, sending unneeded data over a busy network channel may slow the entire system including both the server, the network, and other users of these, whereas sending unneeded data over a free network channel may not significantly slow the system.

5 Predictors

Below is a list of prediction algorithms. Each algorithm takes as input the most recent URL being requested, and the history of previous requests. The predictor is allowed to make zero or one predictions. Predictors that predict multiple files per request are an important area of future work.

Although we don't describe every predictor we tested, we do describe ideas that we thought might work, and yet did not. Our hope is that others can learn from this, or at least not repeat these mistakes. However, analysis focuses on successful ideas.

5.1 The Last Instance Predictor

This is a simple predictor. When a resource is requested, we look back in history for the most recent time where the same user requested the same resource, and predict what followed. If this is a first request, no prediction is made. This algorithm produces many predictions, and is often wrong.

PredictionRate	91%
SuccessRate10	34%
WrongRate10	57%
Accuracy10	37%

Table 1: Last Instance Predictor

5.2 The Pattern Predictor

The Pattern predictor looks for an exact duplicate of long sequences of the most recent requests, and then projects the future based on these duplicates. For example, if the last four files requested were (fileA, fileB, fileC, fileD), the pattern predictor will first look for another instance in the history of accesses for the pattern (fileA, fileB, fileC, fileD). If it finds such an instance it will predict whatever followed. Otherwise it will look for the shorter pattern (fileB, fileC, fileD) and if found predict whatever followed. If there are no copies of the length four and three patterns, it will look for the length two pattern (fileC, fileD), and eventually just (fileD). In practice we do not look for infinite length patterns, but start at length twenty because experimentation shows there are very few matching patterns of greater length. This algorithm often makes a prediction (69% of the time), and when it does it is often right (78% of all predictions are correct).

PredictionRate	69%
SuccessRate10	55%
WrongRate10	14%
Accuracy10	78%

Table 2: Pattern Predictor

5.3 The Pattern By User Predictor

The Pattern By User predictor works just like the Pattern predictor, except that we look at only the currently requesting user's history. The intuition behind this idea is that users might have different access patterns, and basing predictions of one user on the histories of other users might lead to errors. Since limiting the histories to be examined uses less data, we would expect fewer predictions to be made. But experience shows that most users **do** follow the same patterns. Therefore, this results in both less prediction volume and less accuracy.

PredictionRate	34%
SuccessRate10	24%
WrongRate10	10%
Accuracy10	70%

Table 3: Pattern By User Predictor

5.4 The Before Then After Predictor

The Before-Then-After predictor is both very simple, and very similar to the Pattern predictor. At any given moment in time, we look at the two previous requests. We then search the history for as many occurrences of this size two pattern as possible, and predict the most common thing to follow. It is the same as the pattern predictor with a fixed pattern size of two. If there are no such occurrences in the history, we make no prediction.

PredictionRate	90%
SuccessRate10	45%
WrongRate10	45%
Accuracy10	50%

Table 4: Before then After Predictor

5.5 Ben’s Predictor

This predictor is a more complex attempt to predict the next request. It compares two possibilities: the most common file that followed the current request for all users taken as a whole, and the most common file that followed the current request for for currently requesting user. It executes the following steps: (1) Find the every request of the current file made by the current user (2) Find the most common next request by the same user (3) Find the percentage of the most common follow-up compared to the total number of requests for the current file. (4) Do the same computation globally, ignoring user IDs. Ben’s predictor predicts whichever prediction has the higher percentage. This algorithm has higher rate of predictions, but a lower accuracy than the Pattern predictor.

PredictionRate	81%
SuccessRate10	48%
WrongRate10	33%
Accuracy10	60%

Table 5: Ben's Predictor

5.6 Luke’s Voting Algorithm

Ensemble, voting prediction algorithms are known to often perform better than their component parts[9]. Even the famous Netflix prize was won by a voting algorithm[11]. Our voting algorithm works by taking the votes cast by the other predictors described above and finding the most common one between them. This simple majority voting algorithm is the best performing voting algorithm of the many ones we tried.

Note that this algorithm will make a prediction if any of the constituent algorithms described above makes a prediction, thus the very high prediction rate. We also tested a variant of this ensemble algorithm that allowed each of the constituent algorithms to vote “no prediction” and would consider that as the possible final answer if “no prediction” gained the most votes. To our surprise, this modification showed a reduction in both PredictionRate and Accuracy10, and is therefore strictly inferior to the unmodified algorithm.

PredictionRate	91%
SuccessRate10	67%
WrongRate10	24%

Accuracy10	75%

Table 6: Luke's Voting Algorithm

6 Conclusion

After reviewing our findings it is clear that there is a significant performance increase to be found from intelligent prefetching of web resources.

Luke’s Voting Algorithm and the Pattern Predictor stood out among all predictors. The Pattern Predictor had a prediction rate of 69%, and of them 78% were correct. It therefore predicted 55% of all requested files. Luke’s Voting Algorithm has a prediction rate at 91%, and 75% of them were correct. It therefore predicted 67% of all requested files. This represents a tradeoff between prediction accuracy and having a high prediction rate. Overall Luke’s Voting Algorithm best fits our criteria, offering the highest prediction rate with only a very minimal decrease in accuracy compared to the Pattern Predictor.

Using Luke’s Voting Algorithm will allow a web server to predict which files a web client will use, send the predicted files to the web client before the client asks for them, and do so with high probability (~75%) that the file will actually be needed. The web server can do this any time there is unused bandwidth between the client and server, and for typical networks this is very often. Doing so will substantially improve web performance

References

[1] HTTPbis Working group. Hypertext Transfer Protocol version 2.0. http://tools.ietf.org/html/draft-ietf-httpbis-http2-09

[2] T. M. Kroeger, D. D. E. Long, and J. C. Mogul, “Exploring the bounds of web latency reduction from caching and prefetching”, Proceedings of the USENDC Symposium on Internet Technology and Systems, (1997), pp. 13-22.

[3] U. Acharjee, Personalized and Artificial Intelligence Web Caching and Prefetching. Master thesis, University of Ottawa,Canada(2006).

[4] Y.f. Huang and J.M. Hsu, “Mining web logs to improve hit ratios of prefetching and caching”. Knowledge-Based Systems, 21(1), (2008), pp. 62-69.

[5] G. Pallis, A. Vakali, and J.Pokorny, “A clustering-based prefetching scheme on a Web cache environment”, Computers and Electrical Engineering, 34(4), (2008). pp.309-323.

[6] W. Feng, S. Man, and G. Hu, “Markov Tree Prediction on Web Cache Prefetching”, Software Engineering, Artificial Intelligence(SCI), Springer-Verlag Berlin Heidelberg, 209,(2009). pp. 105–120.

[7]Int. J. Advance. Soft Comput. Appl., Vol. 3, No. 1, March 2011 ISSN 2074-8523

[8] Singh, N, Panwar, A. ; Raw, R.S.. Enhancing the performance of web proxy server through cluster based prefetching techniques, Advances in Computing, Communications and Informatics (ICACCI), 2013 International Conference, 22-25 Aug. 2013

[9]Opitz, D.; Maclin, R. (1999). "Popular ensemble methods: An empirical study". Journal of Artificial Intelligence Research. 11: 169–198. doi:10.1613/jair.614

[10] Evaluating Several Different Web Prediction Algorithms (with Randy Appleton, Josh Thompson, and Matthew Menze). 29th International Conference on Computers and Their Applications (CATA-2014). March 24-26, 2014. Los Vegas NV.

[11]Y. Koren, "The BellKor Solution to the Netflix Grand Prize", (2009). http://www.netflixprize.com/assets/GrandPrize2009_BPC_BellKor.pdf

[12] Wang, Jia. "A survey of web caching schemes for the internet." ACM SIGCOMM Computer Communication Review 29.5 (1999): 36-46.

[13] Wang, Jia. "A survey of web caching schemes for the internet." ACM SIGCOMM Computer Communication Review 29.5 (1999): 36-46.