Evaluating Several Different Web Prediction Algorithms

Randy Appleton, Josh Thompson, Matt Menze
Northern Michigan University
rappleto@nmu.edu

Abstract

Web performance is very important. One way to improve performance is through caching. But caching is already widely used, and studies suggest that much of the theoretically achievable performance from caching is already being realized. Prefetching techniques could improve performance beyond that of caching but require future knowledge. This paper evaluates several different prediction algorithms for gaining this future knowledge by exploring the trade offs between prediction rate, erroneous prediction rate, and the resources needed to make the prediction. Previous studies[1,2,3] have developed many prediction algorithms; each one evaluated with a unique pattern of requests in a unique environment. By evaluating several algorithms in the same environment using the same data, we can more clearly compare the algorithms.

Keywords: prefetching, prediction, web cache.

1. Introduction

Users want the web to be fast. But there are several sources of delay when a user requests a web page. The request must be sent from client to server. The server must generate the data, which can be slow if the page is computationally expensive to generate or requires querying databases. The reply must be sent to the client. Finally the client must render the page into a visual form.

If one could predict future requests, it would be possible to make the web much faster. Predictions of the future might be generated by the web server or the web client, but the web server has a significant advantage over the web client. The server has seen the pattern of access of (potentially thousands) of other users, and can use this to make better predictions. Once the web server has made predictions about future web requests, it can send the data to the client. This can be done several ways, using for example SPDY's “server push” or “server hint” technologies[11].

Many studies have shown that the combination of caching and prefetching doubles the performance compared to single caching [4,5,6,7, 8, 9, 10]. According to [6,7] , a combination of web caching and prefetching can potentially improve latency up to 60%, whereas web caching alone improves the latency up to 26%[10].

This paper explores how web servers might predict the future needs of clients. Many different algorithms exist and significant research has been done on this problem. Different studies show different levels of predictive success. However it is difficult to compare one algorithm against another since they were 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.

2. Methodology

We evaluate the prediction algorithms using simulations based on log files from real web servers. We use data from both an academic web site and a commercial web site, since it is important to test prediction algorithms against a variety of web access patterns. We use simulations since we want to test a variety of algorithms against the same log files.

Input to the simulator is a list of file requests, and the associated user (or host) who generated the request. It is important to know which user made which request since otherwise unrelated accesses that occur close in time might be incorrectly considered as related.

We score each algorithm using several metrics

- SuccessRate – How often did the algorithm predict the next web request.
- WrongRate – How often did the algorithm wrongly predict the next web request.
- Accuracy – Percent of predictions that were correct. Accuracy=SuccessRate/(SuccessRate+WrongRate).

- SuccessRate10 – How often was the algorithm's prediction within the next ten requests. Sometimes a prediction algorithm will predict a file, and although that file is not the next request, it is requested in the near future. Prefetching based on such predictions is still productive and useful.
- WrongRate10 – The fraction of predictions that were not within the next ten requests.
- Accuracy10 – Accuracy10=SuccessRate10/(SuccessRate10+WrongRate10).
- Adaptability – Can the algorithm offer either a few great prediction, or many good ones, as needed. In other words, is the algorithm tunable to different situations and different needs,

Obviously, a perfect algorithm would have a successful prediction rate of 100%, and a wrong prediction rate of 0%. No known algorithm 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.

In practice, there is often a trade-off between number of predictions made and the error rate. Often one can either make a few very likely predictions, or make many predictions where each one of which has only a moderate chance of correctness. Each successful prediction improves performance. Each wrong performance may or may not reduce performance, depending on how many resources were needed to generate and send the unneeded data, and whether those resources were idle or in use. In particular, sending unneeded data over a busy network channel might slow the entire system, whereas sending unneeded data over a free network channel might not slow the system much at all.

Often the algorithms will have a threshold parameter that adjusts the amount and riskiness of predictions. The appropriate tolerance for risk depends greatly on particular environment and conditions at that moment. The best prediction algorithms can adjust, making either more or better predictions as needed.

3. The Algorithms

We evaluated each of the algorithms below using the metrics above.

SimpleLookBack

The SimpleLookBack algorithm is the simplest reasonable algorithm we know of. When a request for a file A arrives, look back into the past to see what request came immediately after the last request of A, and predict this file.

Adaptable	No		
	Academic	Business	Avg
SuccessRate	19.7%	25.2%	22.5%
WrongRate	35.6%	59.1%	47.4%
Accuracy	35.7%	29.9%	32.8%
SuccessRate10	30.2%	38.5%	34.4%
WrongRate10	25.0%	46.0%	35.5%
Accuracy10	34.2%	45.5%	40.0%

This algorithm is not adaptable; there is no way to tell it to only make predictions when the algorithm is either reasonably certain or only very certain. Instead, it will always make a prediction except on the first instance of a file, when the file has no history upon which to make predictions.

This algorithm is very computationally simple. One easy way to compute it is to have a hash table of all files that are referenced, who's key is the previous file and who's value is the file that followed. For realistic web sites, these tables fit easily in RAM and can be accessed in O(1) times.

LookbackByUser

This algorithm is just like SimpleLookBack, except instead of examining all previous requests, it only looks back at the history of the current user. Making this change means that the predictions are more accurate (47% vs 33%) since other users with possibly different access patterns will not effect the current user's prediction. Predictions however are less common (33% vs 70%) because those times when other people have requested a file, but it is not in the current user's history will produce no prediction using this algorithm, but would have made a prediction under SimpleLookBack.

Like SimpleLookBack this algorithm is not adaptable; it will make a prediction every time a user requests a file except for the first instance of that particular file by that particular user..

This algorithm is again very computationally simple. One easy way to compute it is to have a hash table of all files that are referenced, who's key is the previous file concatenated with the user's ID (login name, IP number, or any other identifier) and who's value is the file that followed. For realistic web sites, these tables fit easily in RAM and can be accessed in O(1) times.

Adaptable	No		
	Academic	Business	Avg
SuccessRate	12.5%	18.7%	15.6%
WrongRate	13.9%	20.3%	17.1%
Accuracy	47.3%	47.9%	47.6%
SuccessRate10	16.8%	24.8%	20.8%
WrongRate10	9.7%	14.1%	11.9%
Accuracy10	63.4%	63.8%	63.6%

PatternPredictor

The intuition behind this predictor is that if we can find another instance of your access pattern, we can make reliable predictions, and that longer patterns are more reliable that shorter ones.

The PatternPredictor looks for the an exact duplicate of long sequences of the most recent accesses, 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 the instance. Otherwise it will look for the shorter pattern (fileB, fileC, fileD) and if found predict whatever followed that. 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 few patterns of greater length.

The PatternPredictor generates more correct predictions measured by SuccessRate10 than either the SimpleLookBack or LookbackByUser predictor, but at the cost of significantly more CPU and memory usage. It can be implemented at least two ways. One is to use a very large hash table with keys for every pattern of all lengths 2 .. 20, and the other to simply store the most recent history and search it using quick algorithms inspired by text searching[12]. Experimentation shows that one can produce answers on reasonable hardware in less than 0.001 seconds(i.e. much quicker than a network transfer).

Adaptable	No		
	Academic	Business	Avg
SuccessRate	22.9%	30.6%	26.8%
WrongRate	25.7%	49.7%	37.7%
Accuracy	47.1%	38.1%	42.6%
SuccessRate10	31.7%	43.6%	37.7%
WrongRate10	16.9%	36.7%	26.8%
Accuracy10	65.2%	54.3%	59.8%

SubsetPredictor

The SubsetPredictor can be viewed as a modification of the PatternPredictor where, instead of looking for an exact duplicates of long sequences of the most recent accesses, we search for an approximately identical set. The most recent accesses form a set of n files when we disregard the order in which they occurred. We search for the most recent set (ignoring order) of n files which contain at least K percent of the recently accessed files.

The idea here is that two long sequences of file accesses that differ only by a few files regardless of order to precede the same set of files. For example, if the last four files requested were (fileA, fileB, fileC, fileD), the SubsetPredictor will find the most recent set of four files which contain at least three of files A through D.

Note this table below is in a different format from the others. This format shows how the SuccessRate and WrongRate can vary as the subset size is adjusted; choosing the right subset length is an engineering question whose answer depends on the computational and network environment. Also note that the accuracy is nearly constant across varying subset lengths (which surprised us).

Computationally, this algorithm can be implemented by storing the recent history, and searching upon file requests. Again, this can be done much quicker than a network transfer.

Adaptable	Yes					
	Academic		Business		Avg	
Subset Length	2	10	2	10	2	10
SuccessRate	21.5%	5.0%	22.3%	5.4%	21.9%	13.5%
WrongRate	29.4%	5.6%	34.4%	17.5%	31.9%	18.8%
Accuracy	42.2%	47.2%	39.4%	66.7%	40.8%	44.0%
SuccessRate10	32.4%	7.0%	37.8%	8.7%	35.1%	21.1%
WrongRate10	18.6%	3.6%	7.2%	7.2%	12.9%	8.3%
Accuracy10	63.5%	65.9%	42.7%	68.9%	53.1%	59.5%

MostCommonFollowerPredictor

In this predictor, we find all previous occurrences of the current file. Each of these occurrences has a file that follows, and we predict the file that occurs most often in that set. In other words, if fileA is the current file, we predict the file that followed fileA in the history most often with the proviso that it must have occurred at least 25% of the time. Two tuning parameters are to vary the length of the history to consider, and to vary the 25% threshold.

We chose the 25% threshold by experimentation. Larger thresholds would reduce the number of predictions while increasing the accuracy. Smaller thresholds would do the opposite.

Computationally this algorithm can be implemented by a hash table of liked lists. The key is the current file, and the value of the hash table is a linked list of files that have followed, along with a count of each files occurrence. Both memory usage and computational time are reasonable.

Adaptable	Yes		
	Academic	Business	Avg
SuccessRate	16.4%	21.4%	18.9%
WrongRate	25.7%	32.8%	29.2%
Accuracy	32.0%	39.5%	36%
SuccessRate10	24.5	31.5%	28.0%
WrongRate10	17.7	22.8%	20.2%
Accuracy10	58.0	58.0%	58.0%

4. Conclusion

The table below summarizes our results. Two algorithms are of note. The SubsetPredictor while using a small subset size produces a large number of predictions with a high accuracy. The LookBackByUser algorithm produces a smaller set of predictions with a similar accuracy, but at a lower computational cost.

Using these algorithms will allow a web server to predict which file a web client will use, and then send the predicted file to the web client before the client asks for them, and do so with high probability (~60%) 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 improve substantially improve performance.

Algorithm	SuccessRate10	WrongRate10
SimpleLookBack	34.4%	35.5%
LookBackByUser	20.8%	11.9%
PatternPredictor	37.7%	26.8%
SubsetPredictor (size=2) (size=10)	 35.1% 21.1%	 12.9% 8.3%
MostCommonPredictor	28.0%	20.2%

Bibliography

[1] James Griffioen and Randy Appleton. "Reducing File System Latency using a Predictive Approach", Proceedings of the 1994 Summer USENIX Technical Conference, Cambridge MA, June, 1994.
[2] Venkata Padmanabhan , Jeffrey C. Mogul. “Using Predictive Prefetching to Improve World Wide Web Latency”, Computer Communication Review, 1996.
[3] Davison, Brian D. "A survey of proxy cache evaluation techniques."Proceedings of the Fourth International Web Caching Workshop (WCW99). 1999.
[4] Wang, Jia. "A survey of web caching schemes for the internet." ACM SIGCOMM Computer Communication Review 29.5 (1999): 36-46.
[5] 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.
[6] U. Acharjee, Personalized and Artificial Intelligence Web Caching and

Prefetching. Master thesis, University of Ottawa,Canada(2006).
[7] 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.
[8] 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.
[9] 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.
[10]Int. J. Advance. Soft Comput. Appl., Vol. 3, No. 1, March 2011
ISSN 2074-8523
[11] http://www.chromium.org/spdy/link-headers-and-server-hint
[12] Fast String Searching Algorithm, with R.S. Boyer. Communications of the Association for Computing Machinery, 20(10), 1977, pp. 7.