

Predicting Web Cache Behavior using Stochastic State-Space Models ^{*†}

Amitayu Das
Cisco Systems, Inc.
170 W Tasman Dr, San Jose, CA
amitayu@cisco.com

Ritendra Datta, Bhuvan Urgaonkar, Anand Sivasubramaniam
Dept. of CSE, The Pennsylvania State University
University Park, PA
{datta, bhuvan, anand}@cse.psu.edu

Abstract

Accurate analytical models of Web caches are desirable as they can provide inexpensive ways to make resource provisioning decisions at a cache itself as well as at the Web servers it is servicing. Explicitly modeling a Web cache has two major shortcomings: (i) several simplifying assumptions about the operation of the cache for mathematical tractability resulting in loss of accuracy and (ii) measurements of phenomena internal to the cache that may not always be available without adding monitoring hooks within the cache. Therefore, in this paper, we turn towards statistical techniques to develop a model that is non-intrusive (that is, requires no additions to the cache) and treats the Web cache as a black-box (that is, operates solely by observing readily available inputs/outputs and requires no knowledge about the internals of the cache). Relying on the intuition that the internal dynamics of a cache can be captured by a first-order time-dependent process, we develop a model called SMCP, based on the well-studied linear Gaussian state space model, to observe, characterize, and predict the hit rates at a Web cache. A comparison with time-independent models, including one based on Linear Regression (LR), validates our intuition for the need to employ a time-dependent model. A detailed evaluation shows the efficacy of our model with LRU and LFU, two representative cache replacement policies. In our experiments, SMCP predicts hit ratio within 0.1 (absolute value) of their actual value 77.5% and 65% of the times for LRU and LFU, respectively. Secondly, SMCP captures the time-varying behavior more accurately than done by several time-independent models.

Keywords: Web cache, Stochastic state-space model, block-box approach, time-dependent model, prediction, hit

^{*}This research was supported in part by NSF grant CNS-0720456 and a gift from Cisco Systems, Inc.

[†]Published in Proceedings of the Second International Workshop on Scalable Data Management Applications and Systems, Las Vegas, NV; July 2008.

rate.

1 Introduction

Web caches are widely used for reducing the response times of Web clients as well as to reduce the network traffic received by Web servers. A key performance metric of a Web cache is its hit rate. Accurate prediction of the hit rate at a Web cache is desirable since it can provide valuable information about client performance as well as the traffic expected to arrive at the associated Web servers. Such information, in turn, can be used for making resource provisioning decisions at the Web servers or at the cache itself. As an example, a low hit rate during an upcoming time period may indicate an impending increased arrival rate at a Web server, which may then provision more resources to handle this increased workload. Additionally, such prediction must be amenable to efficient, online algorithms in order to be practical.

The hit rate of a Web cache is a complex function of the current contents and their sizes, the incoming requests, the cache size, and the replacement algorithm. Developing models that explicitly capture the working of a cache is often a non-trivial exercise and requires measurements internal to the cache that may not always be readily obtainable. The motivation of this paper stems from the need to build an easily interpretable model which can represent the internal states of the cache without being intrusive and can be built upon minimum amount of information to be effective. This paper makes the following contributions in this context.

- We develop an approach for predicting the hit rate of a Web cache that is based on a linear Gaussian state space model. Our approach is *black box* in nature in that it does not require explicit information about the system (such as replacement policy or cache size); it relies solely on observations of hit rates and request arrivals in the past. Furthermore, our approach does not require any additions or modifications to a cache and is computationally efficient for online use.

- Our approach is *time-dependent* in that it has mechanisms for adapting its representation of the internal state of the cache to dynamically changing workloads.
- We conduct evaluations using a real Web cache trace to show the efficacy of the SMCP approach for two popularly used cache replacement policies, namely LRU and LFU. We also propose a number of time-independent statistical prediction techniques, including one based on Linear Regression, and demonstrate the improvement offered by our approach over these.

2 Background and Motivation

Web caches are deployed at various locations between Web servers and their clients, including points close to the clients and the servers. These caches differ from several other caching systems in that the units of transfer, storage, and eviction are all files. The files requested, stored and transferred typically show high variability in their sizes [6, 4, 8]. In addition, replacement policies need to be taken into consideration as well (e.g., recency, frequency etc.) [3, 11].

Consequently, understanding the cache behavior, given the time-varying nature of workload, becomes very challenging. Efforts to understand this behavior for multi-level caching systems have included the building of analytical models for enhancement of performance [20, 12]. One problem with analytical modeling of behavior is that it requires detailed information about the system in advance and requires a great amount of time for building. Moreover, they often end up being limited views of the system, due to the simplifying assumptions typically made. Consequently, the generation of the model becomes a time-consuming process which often fail to handle the time-varying behavior of incoming stream of requests. A major challenge lies in the ability to capture the internal cache dynamics which, at a given time depends upon the most recent cache state and the current request. A black-box approach does not explicitly require information from the system, and hence, has good potential for non-intrusive online applicability [16].

The behavior of the cache state can be safely considered to be a first-order time-dependent process, where the new cache state depends entirely on the current state and the incoming request, without direct depends on its past states. A number of different statistical learning techniques can be considered for modeling cache by a black-box approach, with the goal being to estimate the hit-ratio over windows of requests. These techniques can be roughly divided into time-dependent models and time-independent methods. Time-independent approaches to learning, such as regression and classification can be employed to predict the output (hit ratio) directly from the input (request window). While various linear and non-linear regression meth-

ods can be used to estimate the output precisely, classification methods such as CART [5] and SVM [18] can be used to estimate classes/ranges of hit ratio values. For example, CART has been employed in the past for prediction of storage device performance [19]. A potential shortcoming of time-independent methods is the absence of cache state in the prediction process. Moreover, for the classification approaches, even a multi-class method can predict ranges rather than precise values of hit ratio.

On the other hand, time-dependent models are more closer to the actual cache behavior in principle. One possibility is the use of hidden Markov models (HMMs) for capturing cache state dynamics over time. A limitation of this approach lies in the fact that standard HMMs are finite-state machines, while the cache may not necessarily have a finite state space. To circumvent this problem, we make use of linear Gaussian state space models, often referred to as stochastic linear dynamical systems, which can be thought to be an extension of the HMM to infinite state-space. The motivation of this paper stems from a need to build an easily interpretable system which can effectively capture the internal state transition of the system, without being intrusive. The choice of linear state-space model is appropriate in that it is fairly simple, well-studied, and agrees well with the expected cache behavior. While the addition of time-dependence to the model is intuitive, we test whether it actually plays a role or not by also experimenting with Linear Regression alongside. Previous research efforts considered employing machine learning techniques to solve problems in computer systems [14, 7, 17]. However, to the best of authors' knowledge, no previous work has addressed modeling cache dynamics to predict its behavior.

3 Black-box Models

In the absence of explicit information about the cache, intuitions about its expected behavior/interactions can help build a black-box model for its performance. The specific problem we are looking to solve is as follows. The input to the system is considered to be a window of requests of length L , represented by a *feature vector* $U \in \mathbb{R}^d$ summarizing the window characteristics, such as mean file size (more details on this in Sec. 4), where d is the number of characteristics used. The output of this model is the unknown quantity we wish to predict, which in this case is the cache hit ratio, represented by $Y \in \mathbb{R}$, $0 \leq Y \leq 1$. In the case that the learning method is time-dependent and models the cache state, a state vector $X \in \mathbb{R}^c$. These variables assume (potentially) different values at different points of time, and hence are subscripted by the time t . The problem is therefore to make accurate predictions about Y_t ahead of time, with no further information provided. How much ahead in time to predict, is the next question.

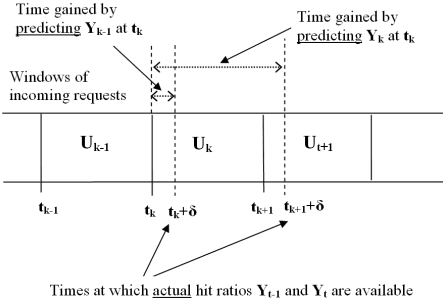


Figure 1. Prediction time-line

We construct a setting for the prediction that can potentially produce benefit in a real-work scenario. In Fig. 1, we show how the time of prediction affects the total advantage gained. Suppose δ is a small time (all times are on a relative scale) required to compute the actual hit ratio after the window of requests has arrived. Assuming that the prediction is considerably faster than the actual cache-based computation, predicting Y_{k-1} at time t_k leads to a gain of δ time. If δ is too small, it may not give the system enough time to make any resource adjustments based on the prediction. It will rather be more advantageous to make the prediction of Y_k at time t_k , which will give us a benefit of $(t_{k+1} - t_k + \delta)$. This could be a significant amount of time, depending on the chosen window size L and the temporal pattern of incoming requests. Hence, we wish to make prediction about a window even before the first request in that window is received. Therefore, in all our cases, Y_t is modeled given X_t as input, but during prediction, the prediction for Y_t is used as an estimate for Y_{t+1} , under the assumptions that (a) cache state does not change significantly over a single request window, and (b) there is local homogeneity in temporal patterns of requests.

3.1 Linear Regression (LR)

We first attempt to solve this problem through the use of a simple time-independent learning model, the linear regression. The model attempts to capture correlation between the predictor variables and the dependent variable, if such correlation exists. The LR model in our case can be represented by

$$Y_t = a_0 + \sum_{i=1}^d a_i U_t(i) + \epsilon \quad (1)$$

with U_t being the predictor variable, Y_t being the dependent variable that is predicted from U_t , and ϵ being a small error. The coefficients (a_0, a_1, \dots, a_d) are typically estimated in the learning phase, which involves training samples drawn from the system consisting of (U_t, Y_t) pairs. There are standard methods for approximation of these coefficients [15].

In particular, the linear least squares method is used in our case. Note that while the coefficients are linear, the values of U_t can include non-linear terms of the actual feature vectors as well. As discussed previously, the predicted value of Y_t is used as an estimator for Y_{t+1} .

Clearly, in the cache setting, LR uses only the request window features to predict the hit ratio, completely ignoring the role of the internal cache state (which the incoming request may be independent of). In other words, given the same request window, the hit ratio will likely vary depending on the current cache conditions. However, a linear regression model will make no distinction and will rather predict the same value in both cases. This shortcoming leads us to look for time-dependent models that can take into consideration both the temporal nature of requests and the state of the cache for the purpose of hit ratio prediction. As we see in sec. 4, this shortcoming does seem to play a role in generating poorer results than with the time-dependent model described below.

3.2 Stochastic State-Space Model

To capture the internal dynamics of the cache, we introduce the notion of an unobserved *state vector* $X_t \in \mathbb{R}^c$ which essentially attempts to model the current cache state at a time t . The dimensionality c of the state vector is expected to reflect on the complexity in state dynamics that the model can capture. The chosen time-dependent model is based upon the known pattern of cache behavior, suitably modified for a window-of-requests setting, as described below:

1. Cache state at an arbitrary time t_{k-1} is represented by a state vector $X_{t_{k-1}}$.
2. Cache receives a window of requests at time t_k , represented by U_{t_k} .
3. The incoming requests U_{t_k} and the previous cache state $X_{t_{k-1}}$ both help determine the new cache state X_{t_k} item. The updated cache state X_{t_k} now determines the outcome of the incoming requests, represented by Y_{t_k} .

To model this behavior, we employ a *linear Gaussian state-space model* with observable input-output. Hence we only care about the state dynamics at these intervals. While the actual requests, cache state change, and hit/miss response occur for each incoming request, we choose to model their behavior over windows of requests rather than over individual requests.

Note further that once we know the cache state at time t_{k-1} , we no longer care about the cache states that preceded it, for the purpose of determining the new cache state at time t_k . This is because typically the current state of a

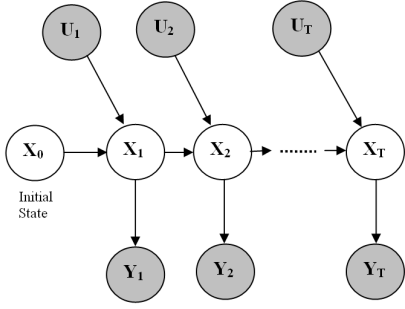


Figure 2. Graphical representation of a state-space model with observable input and output. The shaded nodes represent observable variables.

cache summarizes all the factors that can affect the cache response. If all variables are treated as random processes, then this fact can be written in terms of conditional probabilities as follows:

$$Pr(X_{t_k} | U_{t_k}, X_{t_{k-1}}, X_{t_{k-2}}, \dots) = Pr(X_{t_k} | U_{t_k}, X_{t_{k-1}}) \quad (2)$$

The state change over time is thus a first-order process by nature. Keeping all these factors in mind, an appropriate model for the cache behavior is the linear Gaussian state-space model, as summarized below:

$$\begin{aligned} X_t &= A X_{t-1} + B U_t + w_t \\ Y_t &= C X_t + v_t \end{aligned} \quad (3)$$

$$A \in \mathbb{R}^{c \times c}, B \in \mathbb{R}^{c \times d}, C \in \mathbb{R}^{1 \times c}, w_t \in \mathbb{R}^c, \text{ and } v_t \in \mathbb{R}.$$

The vector $w_t \sim N(0, Q)$ is the state noise drawn from a zero-mean Gaussian distribution with covariance matrix $Q \in \mathbb{R}^{c \times c}$. The variable $v_t \sim N(0, R)$ models the output noise, drawn from a zero-mean Gaussian distribution with variance $R \in \mathbb{R}$. The current state depends linearly on the previous state and the current input, and the output depends linearly on the current state. Thus, the noise variables help to model the inherent system noise, as well as to compensate for the errors due to the linearity assumptions. Equation 3 helps to capture the inherent first-order behavior of the cache states, as discussed earlier. A graphical representation of this input-output state-space model (named State-space Modeled Cache Prediction (SMCP)) is shown in Fig. 2. Here, our objective is to predict the output at time $(k+1)$ while being at time k , even before receiving input U_k . To do so, we still learn the parameters based on the model described by Eqns. 3 and 4 using the training data. While this would only allow us to make a prediction for Y_k after U_k is received, we use the prediction for Y_k as a *predictor* for Y_{k+1} , under the assumptions that the cache state does not change much over a single window, and the request windows have some local homogeneity. Thus, given

the learnt parameters, prediction occurs as follows:

$$\begin{aligned} Y_k &= C X_k + v_k \\ Y_{k+1} &\leftarrow Y_k(\text{predictor}) \end{aligned}$$

While there exist various methods for the estimation of parameters of a stochastic state-space model [10], we train parameters based on the method proposed in [13], which produces maximum likelihood estimates of parameters. The implementation, which entails an expectation-maximization (EM) algorithm [9] together with Kalman filtering [2] for iterative estimation of the parameters of the state-space model, is made available online¹ by the authors. We utilize this implementation for learning parameters A, B, C, Q, R , and the multivariate Gaussian parameters (μ_0, Σ_0) for the initial value of state vector, X_0 . The model requires the size c of the state vector to be specified, discussed further in sec. 4.

Training and testing are performed on non-overlapping data, and proceed as follows. The training set is split into equal length segments of size L_{tr} (truncating if necessary) and used for the parameter estimation. Because the dynamics are learnt at the granularity of length L_{tr} sequences, we surmise that the state vector may become increasingly inaccurate in the prediction of longer length sequences at a stretch. Therefore, we partition the test set in a similar way as well into length L_{tr} sequences in order. Prediction occurs independently for each sequence, taken in order. Thus for each sequence, an initial state vector is first drawn $X_0 \sim N(\mu_0, \Sigma_0)$, following which the prediction proceeds dynamically as in Eqns. 3 and 4. We note that for consecutive sequences, the cache state at the end of one sequence should be the same as the initial state of the next one. The caveat is that if this state prediction (say X_{last}) is poor, then making $X_0 = X_{last}$ for the second sequence onwards may be an error carried through the chain of sequence, which may cause results to diverge completely (as we observe in some experiments). Instead, we take a middle path in the assignment of X_0 , by computing a linear combination of the last state predicted in the previous sequence with a sample drawn from $N(\mu_0, \Sigma_0)$, that is,

$$X_0 = \alpha X_{init} + (1 - \alpha) X_{last} \quad (5)$$

where $X_{init} \sim N(\mu_0, \Sigma_0)$. Here, $(1 - \alpha)$ measures the relative confidence we have in the last state predicted by our model, making α a system parameter that must be pre-specified in some way. Simply looping through a set of reasonable discrete values for alpha can yield a choice, as we show in Sec. 4. In essence, the alpha parameter pulls the last value for the state vector toward the mean value of the state vector (estimated by the model) to prevent divergence over time.

¹<http://www.gatsby.ucl.ac.uk/~zoubin/software.html>

4 Experimental Evaluation

4.1 Experimental Setup and Performance Metrics

We used real HTTP traces obtained from [1] during the last week of September 2007, for our evaluation. The traces consisted of about 1.35 million requests for static files. Half of it was used to train the models and the remainder was used to evaluate their evaluation. We simulated the behavior of a single web cache (of size 8 GB) with various cache-replacement policies (such as Least Recently Used (LRU), Least Frequently Used (LFU), Random and various skewed random policies etc.). The simulator was written in C++ and training and testing phases were done using Matlab. Representative features were extracted from the workload over windows of successive requests, chosen based on the intuition that they would have correlation with respect to hits and misses in the cache. These features included the following, all of which showed significant correlation with hit ratios in a prior study:

1. Mean of size for the files requested
2. Variance of size for the files requested
3. % of requests for unique files
4. Cumulative size of all unique files

Although we use linear models, in order to be able to capture any non-linear correlation between the features and the correlation, we add the squares, cubes, and cube-roots of each of these five features to the feature set. Effectively, the input vectors consist of 16 features, i.e., $U_t \in \mathbb{R}^{16}$.

The output $Y_t \in \mathbb{R}$ is the hit ratio over a window, which is the quantity we attempt to predict here ahead of time. For a request window of length L , if the number of hits at time t is h_t , then $Y_t = \frac{h_t}{L}$, which also means that $Y_t \in [0, 1]$.

4.2 Results

In order to evaluate the relative performance of our model, we computed the prediction with a representative set of alternative methods, as listed here: (1) **Random**: Random values drawn from a normal distribution having same mean and variance as the training output data; (2) **Last Value Prediction**: The most recent actual observed value used as prediction for current output; (3) **Linear Regression (LR)**.

We employed two different widely used cache replacement policies for our evaluation - Least Recently Used (LRU) and Least Frequently Used (LFU). For various window-sizes, comparison with the above methods showed that SMCP outperforms consistently all of them and LR

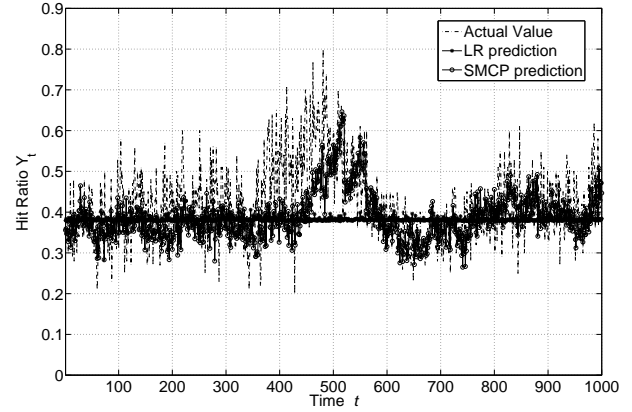


Figure 3. Prediction performance of SMCP and LR method for LRU

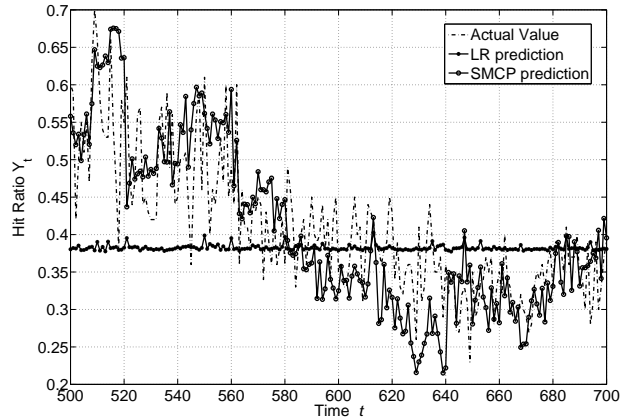


Figure 4. Interval [500-700] zoomed in for LRU

comes closest to SMCP. For example, for window-size of 500, the SMCP method predicts hit ratios within 0.1 (absolute value) of their actual value 77.5% of the times, as against 67.3% in the LR case, for the LRU cache replacement policy. The mean absolute error achieved by SMCP is 14.5% of the original values, as compared to 17.3% in the LR case. Hence, we are presenting the results showing comparison between SMCP and LR only in this section.

Figures 3 and 5 show the hit-ratio (for entire test-duration), actual and predicted (by SMCP and LR), where the replacement policies were LRU and LFU, respectively. For these experiments, SMCP parameters are tuned for best performance, with the dimensionality of state vector $c = 2$, and the $\alpha = 0.75$, which is the combination weight in Eqn. 5. The simulated time has been shown on X-axis in these figures which increase with the arrival of subsequent requests. The time-varying nature of the requests and consequent variation of internal cache state is shown by varying nature of the actual hit-ratio. SMCP, being a

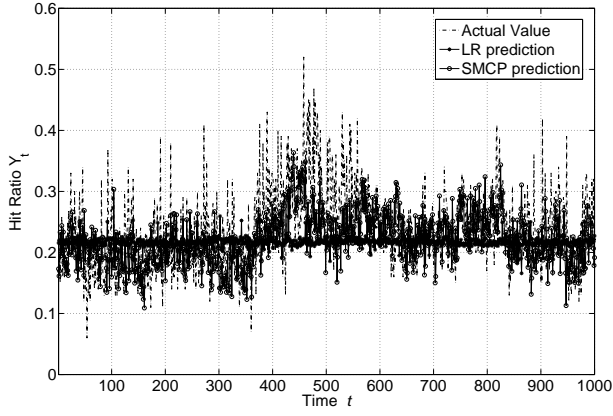


Figure 5. Prediction performance of SMCP and LR method for LFU

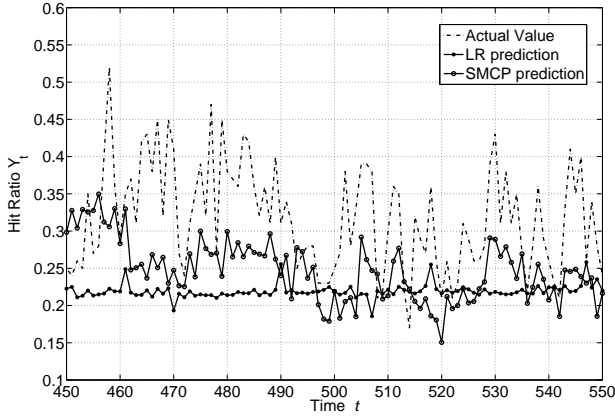


Figure 6. Interval [450-550] zoomed in for LFU

time-dependent model, captures the internal state dynamics and can adapt to the changing behavior of the cache, as shown in these figures. On the other hand, with LR, the learning process associates inputs with outputs directly, without any consideration to the cache state. The absence of such a machinery prevents LR from producing different outputs for the same input under different cache states, as evident from its prediction having very less deviation from a static mean value. Although, these results have been presented for a window-size of 500, similar performances for other window-sizes verify the limitation of LR to capture the cache dynamics. In general, the performances improve over larger window sizes for all cases due to the averaging effect. A larger window in turn works to the advantage of the application, giving the system greater time for resource reallocation.

We also show the goodness of prediction, as achieved by these two methods (for specific test-duration) in figures 4 and 6. For LRU and LFU, the X-axis corresponds to the duration [500-700] and [450-550] of simulated time, respec-

tively. Here also we notice that LR is insensitive to varying cache state. We observe from the graphs that the global as well as the local trends in the hit ratio tend to be captured well with the SMCP model, with prediction errors being tolerable in many cases. The graphs also tend to indicate that the system may continue good prediction at even larger window sizes as well. We believe these results showcase the potential of simple linear dynamical systems such as SMCP, for predictive modeling of this nature.

In terms of computational complexity, once the parameters are learnt, the SMCP method is comparable to the LR method during run-time, because both involve only a small set of matrix multiplications. This indicates that the performance improvement comes without additional overhead on the system. In summary, while we are unable to verify how the abstract state vector X_t corresponds with actual cache states at a given time, the results do indicate a clear role played by the state vector which help produce superior performance.

4.3 Sensitivity Analysis

We conclude the experimental results section with an evaluation of how the SMCP system parameters affect the prediction performance. Root Mean Square Error (RMSE) has been used as the metric for comparison and is defined as

$$RMSE = \sqrt{\frac{1}{T} \sum_{t=1}^T (Y_t^{actual} - Y_t^{predicted})^2}$$

where T is the length of the test sequence. The effect of these parameters are inter-dependent, hence an exhaustive evaluation is not possible. We focus on representative cases that highlight the dependencies of RMSE on them, with data generated using the LRU replacement policy. For each case, 10 runs are made, with mean and standard deviation of errors over them plotted.

(a) *State vector size c* : With $\alpha = .75$, $L = 300$, and $L_{tr} = 50$, we vary the state vector size c , and plot the results in Fig. 7 (a). We observe that performance is best at $c = 2$, and there is progressive improvement beyond that, but with additional computational overhead. This seems to suggest that two variables are sufficient to capture the internal cache state.

(b) *Parameter α* : With $c = 2$, $L = 500$, and $L_{tr} = 50$, we vary the parameter α , and plot the results in Fig. 7 (b). Performance seems best at $\alpha = 0.75$, with progressive degrading beyond it. This seems to suggest that taking the initial vector X_{init} estimated by the model, and the most recent state vector X_{last} in 3 : 1 ratio produces best performance, and that giving greater importance to X_{init} than this causes rise in error.

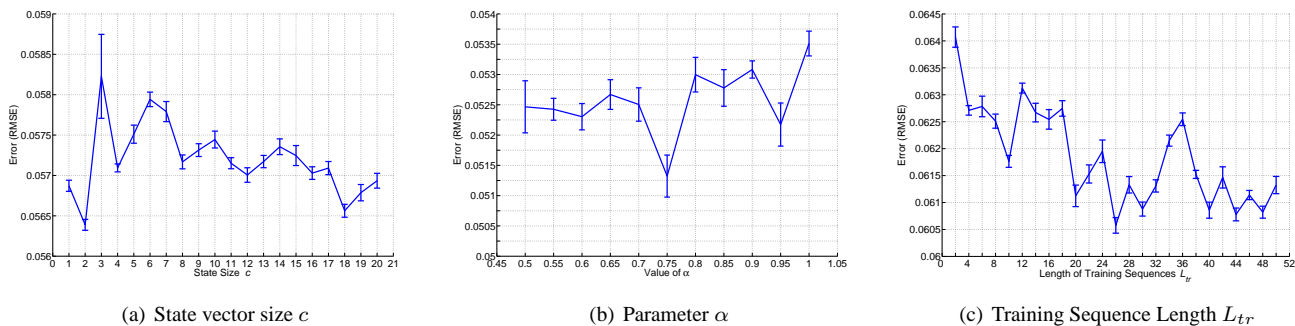


Figure 7. Effect of the system parameters on the RMSE of prediction.

(c) *Training Sequence Length L_{tr}* : With $c = 2$, $L = 200$, and $\alpha = 0.75$, we vary the training length L_{tr} , and plot the results in Fig. 7 (c). Performance seems best at $L_{tr} = 26$, with a general improvement seen from smaller to larger lengths. Note that after the training, the only role played by L_{tr} is that of resetting the state vector according to Eqn. 5. Thus we expect a correlation between L_{tr} and α , which we also observe when generating a surface plot over both of them.

In a real-world setting, these parameters can potentially be tuned through an automated process by looping through possible discrete values. Care must be taken to tune dependent parameters together, e.g., L_{tr} and α . Further, we also observed that the use of the non-linear terms of the main features to extend the input feature set U_t actually helps improve prediction performance. This indicates non-linear dependencies that some of the features may have with the state and the hit ratio.

5 Conclusions

Since predicting the hit rate of a Web cache is crucial for predicting client performance as well as for making resource provisioning decisions, this paper explored statistical techniques for doing that. Relying on the intuition that the internal dynamics of a cache could be captured by a first-order time-dependent process, where the new cache state depends entirely on the current state and the incoming request, we developed a model called SMCP, based on the well-studied linear Gaussian state space model, to observe, characterize, and predict the hit rates at a Web cache. A comparison with three time-independent models, including one based on Linear Regression (LR), validated our intuition for the need to employ a time-dependent model. In our experiments, SMCP predicts hit ratios within 0.1 (real value) of their actual value 77.5% of the times for LRU and 65% of the times for LFU. Secondly, SMCP captures the time-varying behavior more accurately than done by several time-independent models.

We plan to employ the model for different types of caches in future, such as storage-cache, file-cache etc. to evaluate its applicability. Issues like online training would be addressed to handle time-of-day effect for caches.

References

- [1] Web Caching project. <http://www.ircache.net>.
- [2] B. D. O. Anderson and J. B. Moore. *Optimal Filtering*. Prentice-Hall, 1979.
- [3] M. Arlitt, R. Friedrich, and T. Jin. Performance Evaluation of Web Proxy Cache Replacement Policies. *Lecture Notes in Computer Science*, pages 193–206, 1998.
- [4] M. F. Arlitt and C. L. Williamson. Web server workload characterization: The search for invariants. In *Proc. of the ACM SIGMETRICS*, pages 126–137, 1996.
- [5] L. Breiman, J. H. Friedman, A. Olshen, and C. J. Stone. *Classification and regression trees*. Wadsworth, CA, 1984.
- [6] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker. Web Caching and Zipf-like Distributions: Evidence and Implications. In *Proc. of the INFOCOM*, pages 126–134, 1999.
- [7] I. Cohen, J. C. Chase, M. Goldszmidt, T. Kelly, and J. Symons. Correlating Instrumentation Data to System States: A Building Block for Automated Diagnosis and Control. In *Proc. of OSDI*, 2004.
- [8] M. Crovella, M. Taqqu, and A. Bestavros. *Heavy-Tailed Probability Distributions in the World Wide Web*. A Practical Guide To Heavy Tails, chapter 1, Chapman & Hall, New York, 1998.
- [9] A. Dempster, N. Laird, and D. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society, Series B*, 39(1):1–38, 1977.
- [10] J. Durbin and S. J. Koopman. *Time series analysis by state space methods*. Oxford University Press, England, 2001.
- [11] A. P. Foong, Y. Hu, and D. M. Heisey. Web Caching: Locality of References Revisited. In *Proc. of the IEEE International Conference on Networks*, 2000.
- [12] S. Gadde, J. Chase, and M. Rabinovich. Web caching and content distribution: a view from the interior. *Computer Communications*, 24(2), February 2000.

- [13] Z. Ghahramani and G. E. Hinton. Parameter Estimation for Linear Dynamical System. Technical report, Department of Computer Science, University of Toronto, 1996.
- [14] M. Goldszmidt and B. Sabata. Research Issues in Applying Pattern Recognition and Statistical Models to System Management and Modeling. *Algorithms and Architectures for Self-Managing Systems*, pages 29–34, 2003.
- [15] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference and Prediction*. Springer, New York, NY, 2001.
- [16] T. Kelly, I. Cohen, M. Goldszmidt, and K. Keeton. Inducing models of black-box storage arrays. Technical Report HPL-2004-108, HP Labs, 2004.
- [17] R. Powers, M. Goldszmidt, and I. Cohen. Short term performance forecasting in enterprise systems. In *Proc. of the ACM SIGKDD*, pages 801–807, 2005.
- [18] V. N. Vapnik. *The Nature of Statistical Learning Theory*. Springer, New York, 1995.
- [19] M. Wang, K. Au, A. Ailamaki, A. Brockwell, C. Faloutsos, and G. Ganger. Storage device performance prediction with CART models. In *Proc. of the ACM SIGMETRICS*, pages 412–413, 2004.
- [20] A. Wolman, G. M. Voelker, N. Sharma, N. Cardwell, A. R. Karlin, and H. M. Levy. On the scale and performance of cooperative web proxy caching. In *Proc. of SOSP*, pages 16–31, 1999.