A randomized cache replacement scheme approximating LRU

Konstantinos Psounis
Dept. of Electrical Eng.
Stanford University
Stanford, CA 94305
email:
kpsounis@leland.stanford.edu

Balaji Prabhakar
Dept. of Electrical Eng. and
Computer Science
Stanford University
Stanford, CA 94305
email: balaji@isl.stanford.edu

Dawson Engler
Dept. of Electrical Eng. and
Computer Science
Stanford University
Stanford, CA 94305
email: engler@csl.stanford.edu

Abstract —

A randomized algorithm is proposed for approximating the Least Recently Used (LRU) scheme for page replacement in caches. In its basic version the proposed algorithm performs as follows: When a new page is to be evicted from the cache, the algorithm randomly samples N pages from the cache and replaces the least recently used page from the sample. We then study the following enhancement of the basic version: After replacing the least recently used page from the sample, the next $M < N$ least recently used pages are retained for the next iteration. And when the next replacement is to be performed, the algorithm obtains $N-M$ new samples from the cache, and replaces the least recently used page from the $N-M$ new samples and the M previously retained. Both the basic and enhanced versions perform very well compared to existing random replacement schemes. Rather surprisingly, we find that the enhanced scheme can be exponentially better compared to the basic scheme for very small values of $M \geq 1$. As may be expected, we find that as M becomes large the performance becomes worse. This suggests that, for a given N, there is an optimal value of M [1].

I. Introduction

The page replacement problem in caches pertains to the eviction rule for deciding which page currently in the cache should be evicted to make room for a new page. If all the page requests are known in advance, the best strategy is to evict that item whose next request occurs furthest in the future. This offline strategy is known as the MIN algorithm [5]. Typically, it is not possible to know future requests. Algorithms that assume no knowledge of future requests and base their decisions only on past requests are called online algorithms. The optimum online algorithm is known to be the Least Recently Used (LRU) algorithm [5]. LRU works by replacing that page in the cache whose most recent request occurred furthest in the past. We shall refer to this distinguished page as the *oldest* page. Heuristically, LRU's strategy is based on the assumption that the probability a given page will be accessed in the future is proportional to how recently it was accessed for the last time in the past.

The implementation of LRU requires keeping track of the age of all pages. Usually, this is done by a linked list or a stack. However, this entails a large amount of work since, whenever there is a cache access, up to six pointers need to be updated [7]. Due to its complexity and need for hardware support LRU is not implemented in most of today's systems [7]. Instead, in practice, heuristic algorithms that approximate LRU are used. But simplicity has come at the cost of performance.

A class of algorithms known for their simplicity and good performance are the so-called randomized algorithms. For cache replacement a particularly simple algorithm is the Random Replacement (RR) algorithm. The RR algorithm draws *one* page at random from the cache and replaces it [5]. A more complicated algorithm that performs better than RR is the Marker algorithm [2]. This algorithm associates a marker bit to each item of the cache and initializes its value to zero. When an item from the cache is accessed, the marker bit is set to one. The replacement strategy randomly chooses *one* page from among those whose marker is zero and evicts it. A further improvement of the marker algorithm is proposed in [4].

We propose to combine the benefits of both the LRU and the RR schemes. To this end the basic version of our scheme draws N pages from the cache and evicts the *oldest page in the sample.* We then refine this scheme by observing that by carrying the M next oldest samples from one iteration to the next, tilts the distribution of the age of the sample towards the older side and thus one expects the refinement to perform better. More precisely, the refinement works as follows: After replacing the least recently used page from the sample, the next $M < N$ least recently used pages are retained for the next iteration. And when the next replacement is to be performed, the algorithm obtains $N - M$ new samples from the cache, and replaces the least recently used page from the $N-M$ new samples and the M previously retained.

Taking the probability that the page being replaced is *not* from the oldest n^{th} percentile of the pages in the cache as a measure of performance, we find through analysis and simulation that there is indeed an improvement when $M > 0$. Rather surprisingly, the improvement can be *exponential* for values of M as small as 1, 2 or 3. Further, for typical values of N and n, the performance hardly improves as M increases from 3. In fact, as M grows beyond $N/2$, we observe that the performance degrades linearly. This suggests that there is an optimal value of M for which the above probability is minimized and thus the performance is best.

The rest of the paper is organized as follows. Section II presents the details of the algorithm, and Section III presents simulation results of its performance. In Section IV an analytical model is derived, its solution is computed and compared with the results of simulation. Section V investigates the question of how many samples one should keep at each iteration of the algorithm in order to get the optimum performance, and Section VI outlines a proof of the argument that there always exists such an optimum. Finally, Section VII concludes the paper.

[1] This research is supported in part by a Stanford Graduate Fellowship, a TERMAN Fellowship, and a DARPA grant.

II. A discussion of the algorithm

The basic version of the algorithm performs as follows. Whenever a page is to be evicted, N samples are drawn at random from the population and the oldest (least recently used) of these is evicted. An error is said to have occured if the evicted page *does not* belong to the oldest n^{th} percentile of all the pages in the cache, for some desirable values of n. Thus, the goal of the algorithms we consider is to minimize the probability of error. With a slight abuse of language we shall say that a page is *old* if it belongs to the oldest n^{th} percentile.

It is useful to conduct a quick analysis of the basic version of the algorithm described above so as to have a benchmark for comparison. Accordingly, suppose that all the pages are divided into $100/n$ bins according to age and N pages are sampled uniformly and independently from the cache. Then the probability of error equals $(1-n/100)^N$, [2] which approximately equals $e^{-nN/100}$. By increasing N this probability can be made to approach 0 exponentially fast.

For example, when $n = 5\%$ by choosing N to equal 60 the probability of error is around 0.05. One asks whether it is possible to get the same or better performance by drawing fewer samples. We find that it is indeed possible to do this by carrying good samples from one iteration to the next; rather surprisingly, the improvement in performance can be *exponentially* better.

We now describe the general procedure for carrying samples from one iteration to the next. As before, we begin by randomly choosing N samples. After replacing the least recently used page from the sample, the next $M < N$ least recently used pages are retained for the next iteration. And when the next replacement is to be performed, the algorithm obtains $N - M$ new samples from the cache, and replaces the least recently used page from the $N - M$ new samples and the M previously retained. This procedure is repeated whenever a page needs to be evicted. In pseudo-code we have:

```
if (eviction) {
   If (first_iteration) {
      sample(N);
      evict_oldest;
      keep_oldest(M);
   } else {
      sample(N-M);
      evict_oldest;
      keep_oldest(M);
   }
}
```

One potential drawback of the enhanced version ($M > 0$) as compared to the basic version ($M = 0$) is that it is possible for a page that is retained to be accessed between iterations. Such a page would no longer be old and the quality of the pages retained degrades. However, we have observed that the chance of this event occuring are very small [3], and hence assume that pages which are retained are not accessed between iterations. Our assumption is also supported by the general philosophy of any efficient replacement policy: The older a page is, the less likely it is to be accessed. Since, by design, the samples retained between iterations are the oldest possible, we expect that this issue will not significantly affect the performance of the algorithm or our calculations.

[2] Although the algorithm samples without replacement, the values of N are so small compared to the overall size of the cache that $(1-n/100)^N$ almost exactly equals the probability of error. A typical cache has 32K pages and the samples acquired are usually around 60.

[3] Simulations with real page request traces show that our scheme is very close to LRU, which supports our observation.

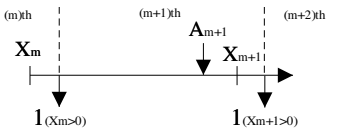

Fig. 1: Eviction takes place prior to resampling.

III. Simulations

This section presents the results of various simulations that indicate how the randomized algorithm performs.

Recall that a page is said to be *old* if it belongs to the oldest n^{th} percentile of all pages in the cache. We will work with a total sample size of N, of which $M (0 \le M < N)$ were retained from the previous iteration. Of all the N samples some will be old, belonging either to the M retained from the previous iteration or to the $N - M$ fresh samples. We are interested in estimating (through simulation and analysis) the probability of error, which is the probability that none of the N pages in the sample is old.

We proceed by introducing some helpful notation. Of the M samples retained at the end of the $(m-1)^{th}$ iteration, let Y_{m-1} $(0 \le Y_{m-1} \le M)$ be the number of old pages. At the beginning of the m^{th} iteration, the algorithm chooses $N - M$ fresh samples. Let A_m, $0 \le A_m \le N - M$ be the number of old pages coming from the $N - M$ fresh samples. In the m^{th} iteration, the algorithm replaces one page out of the total $Y_{m-1} + A_m$ available (so long as $Y_{m-1} + A_m > 0$) and retains M pages for the next iteration. Note that it is possible for the algorithm to discard some old pages because of the memory limit of M that we have imposed.

Define $X_m = \max(M + 1, Y_{m-1} + A_m)$ to be number of "useful" old pages; that is, these are precisely the old pages that the algorithm would ever replace at eviction times. If $X_m = 0$, then the algorithm commits an error at the m^{th} eviction. It is easy to see that X_m is a Markov chain and satisfies the recursion

$$X_m = \max(M + 1, X_{m-1} - 1_{(X_{m-1}>0)} + A_m),$$

and that A_m is binomially distributed with parameters $N - M$ and $n/100$. Figure 1 is a schematic of the above embedded Markov chain.

In the rest of this section we present plots showing how the error, $P_0 = P(X_m = 0)$, varies with M. The probabilities are taken after the Markov chain has equilibirated. In order for the scheme to have reasonably good performance, the number of samples should be at least equal to the number of bins. Thus, we have chosen $N \ge 100/n$.

Figure 2 shows a collection of plots of P_0 versus M for different values of N and n. The minimum value of P_0 is also written on top of each figure. We note that given N and n there are values of $M > 0$ for which the error probability is very small compared to its value at $M = 0$. We also observe that by increasing the number of samples, N, the error probability can be made to be as close to zero as desired. And there

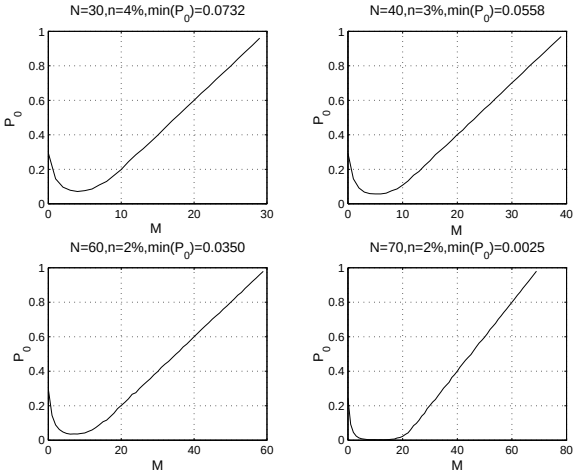

Fig. 2: Probability of error (P_0=probability not an old page is replaced) versus number of pages retained (M).

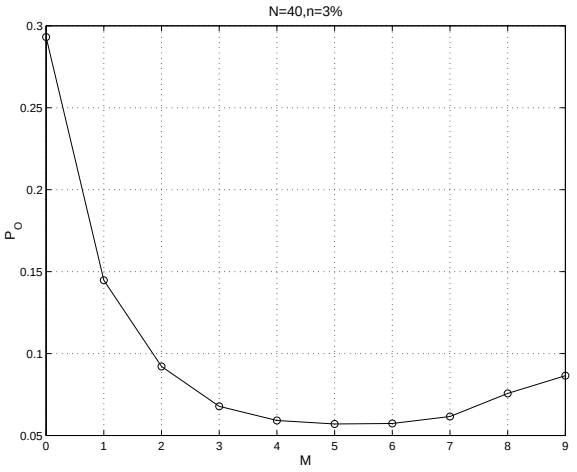

Fig. 3: Rate of decrease of P_0 as M increases

is no need for N to be a lot bigger than the number of bins $100/n$, since even for $N = 2 \cdot 100/n$ the minimum probability of error is extremely small.

Figure 3 zooms in on P_0 for small values of M. We notice that for small values of M there is a huge reduction in the error probability and that the minimum is achieved for a small M. As M increases further the performance deteriorates linearly. In this particular example, the optimum appears for $M = 5$ and there is no significant improvement for $M > 3$, while for $M > 6$ the performance deteriorates.

The exponential improvement for small M can be intuitively explained as follows. For concreteness, suppose that $M = 1$ and that the Markov chain X_m has been running from time $-\infty$ onwards (hence it is in equlibrium at any time $m \geq 0$). The relationship

$$\{X_m = 0\} \subset \{A_m = 0; A_{m-1} \leq 1\}$$

immediately gives that $P(X_m = 0) \leq P(A_m = 0)[P(A_{m-1} = 0) + P(A_{m-1} = 1)]$. Supposing that $N \geq 3 \cdot 100/n$, $P(A_m = 0) \approx e^{-3}$ and $P(A_m = 1) \approx 3e^{-3}$. Therefore $P(X_m = 0) \leq 4e^{-6}$.

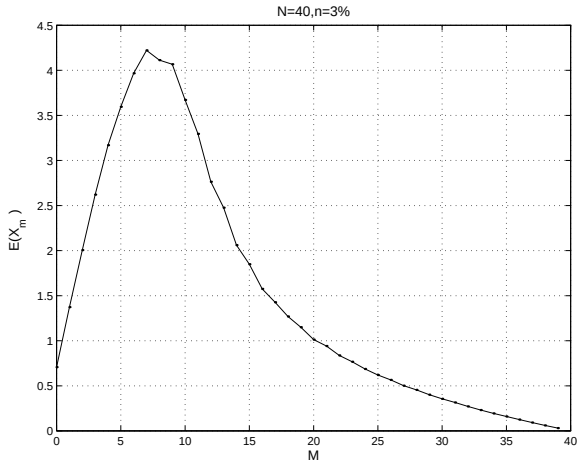

Fig. 4: Average value of X_m as a function of M

Compare this number with the case $M = 0$, where $P(X_m = 0) = P(A_m = 0) \approx e^{-3}$, and the claimed exponential improvement is apparent.

Figure 4 is a plot of the average value of X_m as a function of M. This information could be used to avoid sampling at every time page replacement. For example, if for a particular value of M, $E(X_m)$ is pretty high, then one could take fresh samples every other iteration.

A key point to be deduced from the above plots is that an acceptable performance can be achieved with very small values of M and reasonably small values of N. From an implementation point of view, this is important since it shows that it is not necessary to sample a lot and it is enough to remember very little.

IV. Analysis

In this section we derive and solve a model that describes the behavior of the algorithm precisely. We also compare the results of the model with the simulation results.

The system is modeled by the Markov chain, X_m, which tracks the number of old pages in the sample just prior to the m^{th} page replacement. For a fixed N and n, let $p_k(M) = P(A_m = k)$, $k = 0, \ldots, N - M$, denote the probability that k old pages are acquired during a sampling. When it is clear from the context we will abbreviate $p_k(M)$ to p_k. A_m is binomially distributed with parameters $N - M$ and $n/100$:

$$p_k = \left(\begin{array}{c} N-M \\ k \end{array} \right) \cdot (n/100)^k \cdot (1 - n/100)^{N-M-k}$$

Let T_M denote the transition matrix of the chain X_m for a given value of M. The form of the matrix depends on whether M is smaller or larger than $N/2$. Since we are interested in small values of M, we shall suppose that $M \leq N/2$[4]. It is

[4]The plots in Section III suggest that the M at which P_0 is a minimum is less than $N/2$.

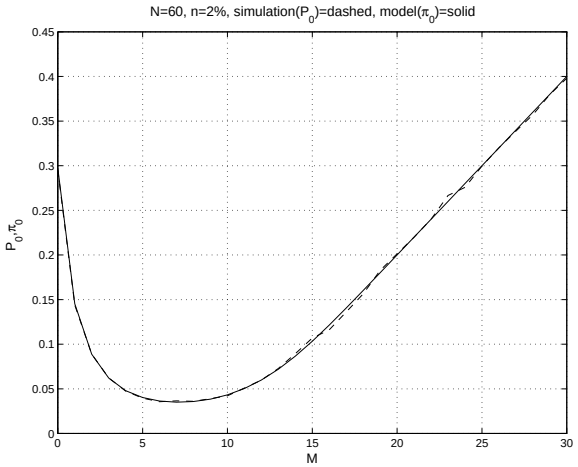

Fig. 5: A plot of the probability of error versus M, simulation (dotted curve, P_0) and analysis (solid curve, π_0).

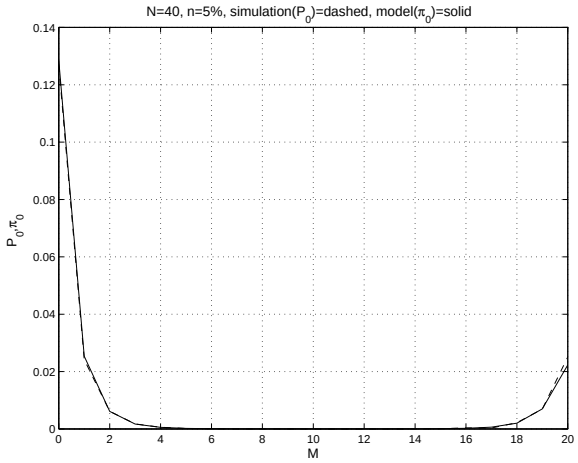

Fig. 6: Comparison between simulation and analysis for relatively large N and n.

immediate that T_M is irreducible and has the general form

$$T_M = \begin{pmatrix} p_0 & p_1 & p_2 & \dots & p_M & 1-\sum_{i=0}^{M} p_i \\ p_0 & p_1 & p_2 & \dots & p_M & 1-\sum_{i=0}^{M} p_i \\ 0 & p_0 & p_1 & \dots & p_{M-1} & 1-\sum_{i=0}^{M-1} p_i \\ 0 & 0 & p_0 & \dots & p_{M-2} & 1-\sum_{i=0}^{M-2} p_i \\ \vdots & & & \ddots & & \vdots \\ 0 & 0 & 0 & \dots & p_0 & 1-p_0 \end{pmatrix} \quad (1)$$

As may be inferred from the transition matrix, the Markov chain models a system with one deterministic server, binomial arrivals, and a finite queue size equal to M (the system's overall size is $M+1$). An interesting feature of the system is that as M increases, the average arrival rate, $E(A_m) = (N-M)n/100$, decreases linearly and the maximum queue size increases linearly.

Let $\pi = (\pi_0, \dots, \pi_{M+1})$ denote the stationary distribution of the chain X_m. Let $A = (a_{ij})$ be an $(M+2)\times(M+2)$ matrix, with $a_{ij} = 1$ for all i, j. Let $a = (a_i)$ be a $1 \times (M+2)$ matrix with $a_i = 1$ for all i. Since T_M is irreducible, $I - T_M + A$ is invertible [6] and

$$\pi = a \cdot (I - T_M + A)^{-1}. \quad (2)$$

Figure 5 compares the probability of error obtained from simulation, P_0, to that obtained by analysis, π_0, for various values of M. The slight difference between the two lines in the figure are due to simulation error, since the simulation results depend slightly on the seed used in the random number generator. Additionally, due to the nature of the scheme, no matter how many iterations we run, the convergence of the simulation is oscillatory and not monotone. Figure (6) presents another example. Here, we choose $N = 40$ and $100/5 = 20$ and thus expect the scheme to work very well. Indeed, for a wide range of values of M the probability of error is very close to zero. The minimum π_0 achieved is 1.6763^{-5}.

A further improvement on the scheme may result by varying M on the fly, based on how good samples we happen to get at each iteration. If the samples are very good (old) it makes sense to keep all of them while if they are bad (young) it makes sense not to keep any of them. To this end, it is interesting to investigate how far we are from the best possible policy. Ideally, one could sample until one gets an old page that belongs to the oldest bin, use it, and stop sampling. This is impossible in practice since it is not possible to know if a page belongs to the bin with the oldest pages. However, one could use this ideal scheme to compare the average number of samples, $N^{'}$, it requires, to the number of samples, N, required by our scheme. It is easy to see that $N^{'}$ is the mean of a geometric distribution and equals $1/(n/100) = 100/n$. To compensate for the fact that the ideal scheme achieves zero probability of error, one might compare N to $N^{''} = N^{'} \cdot \min(P_0)$. For example, in Figure 2, $N = 30$, 40, 60, 70; $N^{'} = 25$, 33.3, 50, 50; and $N^{''} = 23.2$,31.5,48.3, and 49.9.

V. On the optimal value of M

In this section we investigate optimal values of M for given N and n. That such an M always exists follows from the convexity of $\pi_0(M)$, which is established in the next section. Formally, the optimal value of M is defined as

$$M^* = \arg\min\{\pi_0(M)\}.$$

We note that even though the form of the transition matrix, T_M, allows one to write down an expression for $\pi_0(M)$, there is no closed form solution from which one might calculate M^*. Thus, we numerically solve Equation (2), compute $\pi_0(M)$ for all $M \le N/2$, and read off M^* for various values of N and n, as done in Table 1. This table is to be read as follows: For example, suppose N=30 and $n = 4\%$, the minimum value of π_0 is 0.073172 and it is achieved at $M^* = 4$.

From Table 1 it can be concluded that $\pi_0(M^*)$ is extremely small in certain cases, but it is achieved at relatively large values of M^*. Practically, in these cases it makes sense to use a value of $M = M^+ < M^*$ such that $\pi_0(M^+)$ is very close to $\pi_0(M^*)$. Table 2 presents suitable values of M^+ by requiring that

$$M^+ = \min\{M \le M^* : |\pi_0(M) - \pi_0(M^*)| < 10^{-3}\}.$$

The above discussion presents results regarding the optimum value of M given N and n. We now give some insights as to why there always exists such a value and motivate the next section.

N	$min(\pi_0)$			M^*		
	n=5	n=10	-	n=5	n=10	-
20	0.19456	0.00129	-	2	5	-
	n=4	n=8	-	n=4	n=8	-
30	0.073172	2.4454^{-6}	-	4	9	-
	n=3	n=6	n=9	n=3	n=6	n=9
40	0.055794	8.0595^{-8}	4.6629^{-15}	5	12	16
	n=2	n=4	n=6	n=2	n=4	n=6
50	0.13538	1.8678^{-6}	9.5368^{-14}	4	13	18
60	0.035002	8.3933^{-11}	-	7	19	-
70	0.0025402	-	-	11	-	-
80	3.1553^{-5}	-	-	16	-	-

Tab. 1: Optimum values of π_0 and M for various N and n

N	$min_{practical}(\pi_0)$			M^+		
	n=5	n=10	-	n=5	n=10	-
20	0.19456	0.0016899	-	2	4	-
	n=4	n=8	-	n=4	n=8	-
30	0.073172	0.0003229	-	4	3	-
	n=3	n=6	n=9	n=3	n=6	n=9
40	0.055794	0.00026642	0.00070757	5	3	1
	n=2	n=4	n=6	n=2	n=4	n=6
50	0.13538	0.00045789	0.00019338	4	4	2
60	0.035002	0.00036471	-	7	3	-
70	0.0035109	-	-	8	-	-
80	0.00090908	-	-	6	-	-

Tab. 2: Practically optimum values of π_0 and M for various N and n

Given an arbitrary N and n, let A and B be two instantiations of the scheme proposed, for M and $M+1$ respectively. Let λ_A be the average arrival rate of old pages from resampling in system A and λ_B be the arrival rate of old pages from resampling in system B. Obviously, $(N-M)n/100 = \lambda_A > (N-M-1)n/100 = \lambda_B$ and thus system A on average gets more old pages from resampling than system B. However, the queue size of system A is smaller than that of B by one place. In other words, there will be some cases where Q_A will be full and old pages will be dropped, whereas Q_B will be able to accommodate an extra old page from a previous iteration or from resampling. When M increases from 0 to 1, the positive effect from the increase in the queue size is greater than the negative effect from the decrease in the arrival rate (for typical values of N and n). As M increases it is less likely that overflows occur and the dominating phenomenon is the decrease of the arrival rate. This trade-off between high arrival rate and high queue size causes π_0 to be a convex function of M, and thus there is an optimal value of M at which π_0 is minimized.

Figure 7 demonstrates the convexity of π_0 as a function of M for different values of n and a fixed value of N. We already established in Section III the exponential decrease of π_0 for small M, when the samples are good. The linear increase of π_0 for large M, evident from Figure 7, is explained as follows: As M increases, the average arrival rate decreases and the queue size increases. As a result, the queue never overflows and the only phenomenon into play is the linear decrease of the arrival rate. For a queue that never overflows, $\pi_0 = 1-\lambda/\mu$ and thus π_0 increases linearly as a function of M.

Heuristically, one can make the following observation regarding the value of M^* for typical values of N and n. As the ratio $N/(100/n)$ increases, $\pi_0(M^*)$ decreases and M^* increases. In other words, the more samples there are compared to the bins, the smaller the minimum is and the older the samples tend to be. Thus, it makes sense to retain more of them for future iterations, resulting in a larger M^*.

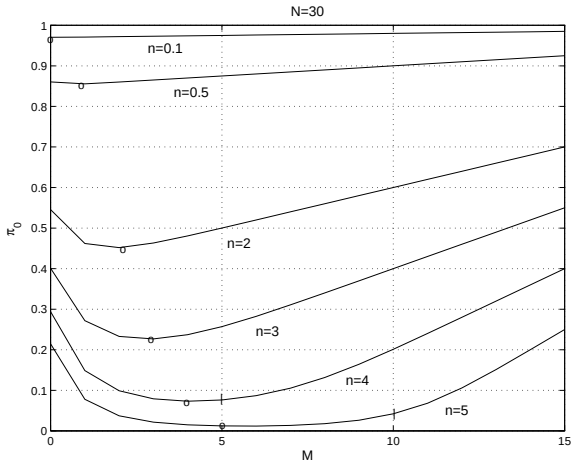

Fig. 7: Convexity of π_0 as a function of M.

VI. On the convexity

In this section we outline a proof of the fact that $\pi_0(M)$ is a convex function of M, from which the existence of an optimum value of M follows.

As remarked earlier, $\pi_0(M)$ cannot be expressed as a function of the elements of T_M in a closed form. Thus, it is not possible to establish its convexity directly. We shall therefore relate $\pi_0(M)$ to the quantity, $D(t,M)$, which counts the number of overflows in the time interval $[0,t]$ from a buffer of size M. We shall establish the convexity of $\pi_0(M)$ by establishing that $D(t,M)$ for our system is convex in M for each $t>0$. The convexity of $D(t,M)$ follows from two lemmas presented below. Due to limitations of space we can neither present the proofs of the lemmas nor elaborate the exact nature of the connection between $\pi_0(M)$ and $D(t,M)$ in this paper.

Consider a queueing system with a buffer of size M. Suppose that the buffer is empty at time 0. Let $D(t,M)$ be the number of overflows from the buffer in the interval $[0,t]$. We want to examine the behavior of $D(t,M)$ for different buffer sizes. Note that $D(t,M)$ is obviously a decreasing function of M since the larger the queue, the less the number of overflows.

Lemma 1 *$D(t,M)$ is a convex function of M.*

Recall that our system is modelled as a queueing system with one deterministic server and binomial arrivals. The average arrival rate $\lambda(M) = (N-M)n/100$. Therefore, the arrival process depends on M. As we vary the buffer size M, the arrivals also vary. However, in Lemma 1 the arrival and departure processes of the queueing system are assumed to remain unchanged for the various values of M. Thus, Lemma 1 does not imply the convexity of $D(t,M)$ directly.

Lemma 2 *$D(t,M)$ is a convex function of M when the average arrival rate $\lambda(M) = -a \cdot M + b$ for $a, b > 0$.*

Lemma 2 establishes the convexity of $D(t,M)$ for our system. Finally, from Lemma 2 the convexity of $\pi_0(M)$ is established:

Theorem 1 *The probability of error $\pi_0(M)$ is convex in M.*

VII. Conclusions

In this work we have introduced a randomized algorithm for approximating LRU. Two versions of the algorithm are studied through simulation and analysis. We find that carrying a small amount of information regarding good samples from one iteration to the next, leads to a dramatic improvement in performance. By a judicious of parameters (the total number of samples, N, and the number of good samples, M, retained from one iteration to the next) we find that LRU can be approximated as closely as desired.

We are currently implementing versions of the schemes mentioned here in real caches. Both versions of the algorithm assume that each item of the cache is time stamped with the last time that was accessed. For page caches, updating time stamps can be very expensive. Thus, approximations similar to the clock algorithm [7] may be needed for a practical implementation. For web caches updating time stamps is cheap. Exact LRU is possible in this case either by using a stack and updating pointers, or by using the time stamps directly. Thus, an approximation of LRU is not crucial in the web caches case. However, it has recently been shown that web caches schemes which take into account the size and cost of a document outperform LRU [1]. These algorithms (e.g. the greedy dual-size algorithm [1]) require extra data structures to be maintained that increase their cost a lot in comparison to LRU. The scheme introduced in this paper can approximate these algorithms without the need for complex data structures.

In general, our scheme can be used efficiently whenever there is a large population of objects from which the "best" is to be chosen according to some criterion.

References

[1] P.Cao and S.Irani, *Cost-aware WWW proxy caching algorithms*, In proceedings of the USENIX Symposium on Internet Technologies and Systems, Monterey, CA, Dec. 1997.

[2] A.Fiat, R.Karp, M.Luby, L.McGeoch, D.Sleator and N.Youmg, *Competitive paging algorithms*, Journal of Algorithms, 12:685-699, 1991.

[3] Donald Gross and Carl M. Harris, *Fundamentals of Queuing Theory*, Wiley Interscience, 1998.

[4] L.McGeoch and D.Sleator, *A strongly competitive randomized paging algorithm.*, Algorithmica, 6:816-825, 1991.

[5] R. Motwani and P. Raghavan, *Randomized Algorithms*, Cambridge University Press, 1995.

[6] J. Norris, *Markov Chains*, Cambridge University Press, 1997.

[7] A. Silberschatz and P. Galvin, *Operating System Concepts (Fifth Edition)*, Addison Wesley Longman, 1997.