Local Context Matching for Page Replacement

Technical Report UCI-ICS 99-37
Department of Information and Computer Science
University of California, Irvine

Xianping Ge, Scott Gaffney, Dimitry Pavlov, Padhraic Smyth
Information and Computer Science
University of California, Irvine
Irvine, CA 92697-3425
{xge,sgaffney,pavlovd,smyth}@ics.uci.edu

September, 1999

Abstract

In this paper we investigate the application of adaptive sequence prediction techniques the problem of page replacement in computer systems. We demonstrate the ability of a method based on local context-matching to substantially outperform well-known existing alternatives (such as least recently used, or LRU) in terms of minimizing page faults for common UNIX programs. Page traces track the sequences of memory addresses made by a program into its (paged) virtual memory space. We show that it is possible to characterize these types of sequences efficiently using only local information. This local characterization translates into the ability to achieve high prediction accuracy on these sequences motivating the new paging algorithm. The method empirically outperforms LRU with reductions of up to 70% in page faults in terms of reducing the gap between LRU and the optimal offline strategy.

Trace no.	Program name	Version	No. of Distinct Pages
1-6	gnuplot	3.50.1.17	85
7-12	latex	C 3.14t3	80
13-14	gunzip	1.2.4	31
15-16	gzip	1.2.4	33
17-20	paging	Sun release 4.1	27
21-24	grep	Sun release 4.1	23
25	find	Sun release 4.1	35

Table 1: Information describing 25 Unix traces [4]

1 Introduction

Computers often present a logical view of an executing program's memory space as consisting of a single contiguous block of memory that the program can address in a straightforward manner. In fact, most computers consist of many levels of memory which they manipulate so as to present a uniform logical view to an executing program. In this paper we will consider the case of a two-level memory store with the faster area of memory as cache and the slower area of memory as secondary memory (e.g., RAM). Under this scenario, cache is usually organized into fixed-size blocks (or pages) which are filled with information from secondary memory as needed. The sequence of pages that a program requests during its execution is commonly called its page trace. Table 1 describes a set of page traces for UNIX for different programs (Fiat and Rosen, 1997). The number of distinct pages corresponds to the number of symbols in the sequence alphabet for a given trace. Typically these traces are on the order of 10^6 symbols long.

A paging algorithm's main task is to decide which pages from secondary memory are to be moved into main memory for immediate access by an executing program. If a program requests a page that does not reside in main memory, a *page fault* is said to occur. It is the job of the paging algorithm to fetch the faulted page and decide which page currently residing in main memory will be evicted to make room for the incoming page. As such, the main operation for these types of paging algorithms is to decide which page to evict from main memory. Paging is an important and broad application in computer systems design and optimization, often occurring at different levels in the hardware hierarchy of a computer system.

2 Background on Paging

Paging algorithms are traditionally classified as being either online or offline. Online paging algorithms are given the page trace one page access at a time and must perform their processing as they go. Offline algorithms, on the other hand, are given the entire page trace to analyze in any fashion they wish. However, they must still go through the page trace, one page at a time, and determine which pages they will have in main memory at any instant in time. The difference is that they know exactly which pages will be accessed in the future, and so they will be able to pick good candidate pages for eviction, although they will incur page faults because they are only allowed to evict pages from main memory when a page fault occurs. Note that because offline algorithms require future information they cannot be implemented in practice. It is well known that the optimal offline algorithm (MIN) evicts the page in main memory that will be accessed the furthest in the future (providing a bound on the performance of any online algorithm). If such a page will not be accessed again, then we will be guaranteed to not incur a page fault on that page ever again.

The optimal *online* algorithm is difficult (or impossible) to specify. However, there has been much work in the theoretical computer science community on proving the optimality of various paging algorithms given some set of assumptions about the way the page traces are generated [2][7]. Most of this work focuses on building up the framework to allow the comparison and evaluation of various online algorithms. Many of the novel algorithms that have been proposed are not truly online (e.g., they need a description of the possible page traces) or may be too expensive (in terms of complexity) for practical use. Perhaps even more important is that almost every paging algorithm that has been proposed gets "beaten" by the simple page replacement strategy known as LRU.

Most common operating systems employ a page replacement strategy known as LRU (least recently used). LRU seeks to minimize page faults by evicting the page in main memory that has not been accessed for the longest period of time (the least recently used page). In this way, LRU is attempting to emulate the optimal offline algorithm MIN. LRU behaves in a highly local and adaptive manner, and its extreme simplicity and small complexity make it hard to improve on. If performance is measured by the number of page faults that an algorithm incurs while paging, then it has been shown that LRU outperforms just about all other known algorithms [7].

Fiat and Rosen [4] recently developed a paging algorithm based on *dynamic access graphs* [2]. Dynamic access graphs are built up in an online fashion. They represent the sequence of page requests that have been seen so far by assigning every page to a node in the graph, and connecting two nodes with a weighted edge that (essentially) represents how often the pages occur together. Fiat and Rosen use these graphs to decide which page should be evicted upon page replacement by using a weighted distance measure in the graphs (they evict the page that is farthest away from the faulted page in terms of graph distance). They report experimental results showing their method consistently beats LRU (comparing number of page faults) on a large set of page request sequences. While this method is interesting because it beats LRU, it is relatively complex to implement and unlikely to replace LRU in practice.

3 A Simple Approach to Characterizing Page Traces

3.1 Local Context Matching

Assume that we are given a discrete sequence of length m with an alphabet of size n. The context of the symbol in the ith position, s_i, is the k previous symbols $s_{i-k}, \ldots, s_{i-1}$. To predict s_i, we can simply search back in our sequence for the *first* occurrence of the context of s_i. If such a context is found, we define our prediction for s_i to be the symbol which appears immediately after the matched context. If there exists a context that fails to be matched, then we make no prediction at all. The number of symbols that we are allowed to search back while looking for matching contexts is limited by the history length (this might be set to control the locality or is determined by the number of symbols that are kept in memory).

The resulting prediction will be strongly dependent on the sequence's local behavior since the very first match determines our prediction. Thus, the prediction will be highly adaptive. There are many models, such as online kth order Markov models, that perform similar types of predictions based on the entire sequence that has been seen so far. These models are global in nature and as such they will be returning predictions which are less locally adaptive. Thus, if the nature of the sequence suddenly changes, these models will be predicting symbols using "old" information.

Adaptive online sequence prediction approaches include (for example) the TDAG algorithm [5]. This dynamically builds a Markov tree based data structure that essentially keeps track of some number of contexts of length 1 to some number d, and predicts symbols based on observed frequencies at each node in the graph. In the area of text compression the PPM (Prediction by Partial Match) compression algorithm [3] is probably closest in spirit to what we propose in this paper. PPM maintains frequency counts for the symbols that follow all possible contexts of lengths 1 to some number k, and uses this information to predict symbols appearing in the sequence given the current context. Algorithms such as PPM and TDAG are similar to our local context matching technique because they do not make predictions based on the entire sequence. PPM periodically throws away the learned model and starts over from scratch providing some form of adaptability. However, TDAG, PPM and full online Markov modeling are all too complex for this application in the sense that an impractically large number of frequency counts must be maintained and updated.

3.2 Predictability of Page Traces

We analyzed the 25 traces originally used in the experiments of [4]. The traces were collected by running common UNIX programs (e.g., `grep`) on designated input while collecting their page traces. The trace number, program name, version of the program, and number of distinct pages in each trace is given in Table 1. The one-step ahead prediction accuracies (using the adaptive local prediction method of Section 3.1) were in the range of 85 to 99%, which is rather high given the number of symbols in each alphabet (here, 23 to 85). (Details of the one-step ahead prediction experiments are not shown here given space limitations: page replacement experiments are described in detail later in the paper). We also investigated the entropy of kth-order substrings (all subsequences of length $k, 1 \leq k \leq 6$) and found that the entropy is extremely low (i.e., there is a relatively small number of commonly occurring subsequences). These results suggest that there is significant deterministic high-order structure in these page traces, and that local context matching picks it up. The existence of common substrings can probably be explained by the general nature of the data, e.g., program execution traces with many potentially repeated loops.

3.3 Extending LRU by Exempting Probable Pages

We can extend LRU page replacement strategy by restricting the set of pages that LRU can evict from main memory. The algorithm, LRUe (LRU with exemptions), predicts a set of pages that are likely to be accessed in the near future (given the current context) and marks these as being *exempt* from eviction at the current page fault. LRUe then evicts the page that has not been accessed for the longest period of time from the set of *evictable* pages.

LRUe attempts to mimic MIN by inferring the future page accesses that MIN, being an offline algorithm, is aware of. LRUe does this by employing the local context matching technique from Section 3 to find a matching context in the recent past. If a matching context is found, then LRUe examines the next w pages that occur after this matching context, and exempts any pages in main memory that appear in this subsequence. At most $c-1$ pages from main memory may be exempted, where c is the number of pages in main memory. We will, henceforth, refer to c as the *capacity* of main memory, and to w as the *exemption window length*. Because LRUe employs this simple local context matching scheme, it is able to efficiently compete with LRU, and is as adaptable as LRU itself.

In terms of practical implementation, LRUe must then search back in history for contexts at every page fault. The searching can be made efficient by maintaining a hash table containing the observed contexts of length k that are contained in the current history. We can store the signatures (e.g., a CRC) of the contexts in the hash table instead of the strings themselves, saving both time and space. The time to compute the signature can be compensated by directly using the signature as the hash value itself. With this data structure, the LRUe paging strategy is competitive with LRU in terms of complexity.

4 Experimental Results

In this section we present our results from experiments that were run to test the performance of LRUe as a paging algorithm. For all of the following experiments, the context length is set to 5.

4.1 LRUe vs. LRU

Our first set of experiments were run on the 25 page traces collected by Fiat and Rosen. In these tests we compared the number of page faults that LRUe incurred compared to the number that LRU incurred. In Figure 1a we see the page fault rate of the optimal offline algorithm (MIN), and the ratio of the number of page faults incurred by LRU, and LRUe to those incurred by MIN. These results were obtained on trace number 1 and are reported for varying capacity size. The history length is fixed at $h = 10,000$ and the exemption window length is set at $w = 25$. On average, LRUe reduces the gap between LRU and MIN by more than 70%. For a capacity of 3, LRUe is only 5% worse than MIN. This shows that, in fact, LRUe is closely following MIN's behavior by exempting pages in main memory. In Figure 1b we see the same tests run on all of the 25 traces for a fixed capacity of 10. These results paint the same picture as above.

In Figures 1c and 1d we run the same tests but on a data set obtained from the New Mexico State University Trace R3000 Data Set [1]. The results that we see here are analogous to those on the previous data set. This suggests that we are not capturing behavior only exhibited by a single (biased) data set.

4.2 LRUe vs. Dynamic Access Graphs

In Figure 2, we compare the performance of LRUe to the dynamic access graph based method described in Section 3.1. These results were obtained on trace number 1 from the Fiat and Rosen

Capacity	MIN	ratio to MIN	
		LRU	LRUe
3	0.36	1.15	1.05
4	0.24	1.18	1.10
5	0.20	1.22	1.13
7	0.14	1.38	1.12
10	0.09	1.53	1.06
15	0.02	3.18	1.51
20	3e-4	1.79	1.63

(a) Trace 1 of [6]

Capacity	MIN	ratio to MIN	
		LRU	LRUe
3	0.188	1.21	1.03
4	0.119	1.79	1.04
5	0.055	2.43	1.10
7	7.42e-3	1.63	1.36
10	3.31e-3	2.01	1.96
15	1.71e-4	2.05	1.92
20	4.67e-5	1.39	1.39

(c) Trace 008 of the NMSU R3000 dataset

Trace no.	MIN	ratio to MIN	
		LRU	LRUe
1	0.0867	1.53	1.06
2	0.0641	1.36	1.11
3	0.0806	1.54	1.16
4	0.0868	1.53	1.06
5	0.0643	1.36	1.11
6	0.0866	1.53	1.06
7	0.0269	1.64	1.38
8	0.0457	1.59	1.30
9	0.0262	1.67	1.38
10	0.0261	1.66	1.39
11	0.0449	1.59	1.30
12	0.0243	1.69	1.39
13	0.0002	1.51	1.59
14	0.0003	1.58	1.68
15	0.0026	1.61	1.62
16	0.0017	1.55	1.55
17	0.0002	1.81	1.76
18	0.0001	2.40	1.65
19	0.0002	1.77	1.71
20	0.0001	2.31	1.54
21	0.0003	2.25	1.61
22	0.0003	2.30	1.53
23	0.0003	2.23	1.61
24	0.0003	2.30	1.54
25	0.0925	2.09	1.33

(b) All 25 traces of [6], capacity=10

Trace no.	MIN	ratio to MIN	
		LRU	LRUe
008	0.0033	2.01	1.96
013	0.014	1.76	1.48
015	0.01	1.65	1.53
022	0.014	1.67	1.53
023	0.0032	1.63	1.3
026	0.013	1.37	1.33
034	0.004	2.61	1.6
039	0.00049	1.44	1.42
047	0.0005	1.01	1.01
048	0.00017	1.54	1.46
052	0.0087	2.01	1.84
056	0.0022	1.93	1.59
072	0.002	1.79	1.68
077	0.0024	1.46	1.43
078	0.00025	1.77	1.53
085	0.038	1.58	1.42
089	0.00072	1.55	1.44
090	0.004	1.84	1.64
093	0.0004	1.47	1.35
094	0.0082	1.77	1.65

(d) All 20 traces of the NMSU R3000 dataset, capacity=10

Figure 1: Paging algorithm results: page fault rate for MIN, and the ratio of the number of page faults for LRU and LRUe to that of MIN.

Capacity	MIN	ratio to MIN		
		LRU	LRUe	access graph
5	191855	1.22	1.13	1.20
10	85199	1.53	1.06	1.32
15	16586	3.18	1.51	2.26
20	311	1.79	1.63	1.71
25	172	1.80	1.70	1.68
30	101	1.53	1.50	1.32
35	69	1.58	1.58	1.59
40	56	1.50	1.50	1.45
45	46	1.48	1.48	1.59
50	36	1.42	1.42	1.22
55	29	1.41	1.41	1.24
60	24	1.50	1.50	1.17
65	19	1.47	1.47	1.21
70	14	1.36	1.36	1.29
75	9	1.11	1.11	1.11
80	4	1.00	1.00	1.00

Figure 2: Performance of LRUe vs. access graph based method on Trace 1 of [6].

data set and are reported for varying capacity size. The history length is fixed at $h = 10,000$ and the exemption window length is set at $w = 25$. From these results we see that when the MIN page faults rate is large, LRUe can reduce the gap between the dynamic access graph based method and MIN by a significant amount. However, when the capacity gets so large that almost no page faults are incurred by MIN, LRUe doesn't have much to improve on, and tends to make about 1 or 2 more page faults than the dynamic access graph based method.

4.3 Sensitivity of LRUe to Parameter Values

We also investigated how sensitive LRUe is to changes in its parameter values. These experiments were run using all of the 25 traces of the Fiat and Rosen data set with varying capacity. In Figure 3 we see that LRUe is quite robust with respect to any changes in the values of either or its context length or history length: similarly, the performance of LRUe relative to LRU was also relatively "flat" (not shown due to space limitations).

5 Conclusions

We have found that page request sequences empirically tend to be relatively predictable and have low entropy. This observation motivated the development of a simple adaptive local prediction scheme, in turn leading to the design of a new paging algorithm (LRUe) that extends the classic LRU replacement strategy. LRUe outperforms LRU, on average, by more than 70% (in terms of excess page faults relative to the optimal offline algorithm).

History	Context length				
length	3	4	5	6	7
2000	1.13	1.10	1.10	1.10	1.10
5000	1.12	1.10	1.10	1.09	1.09
10000	1.12	1.09	1.09	1.09	1.09
20000	1.12	1.09	1.09	1.09	1.09

Figure 3: Ratio of the number of page faults incurred by LRUe to MIN.

References

[1] New mexico state university trace database. Available at `ftp://tracebase.nmsu.edu/pub/traces/`.

[2] A. Borodin, S. Irani, P. Raghavan, and B. Schieber. Competitive paging with locality of reference. In *Proceedings of the Twenty Third Annual ACM Symposium on Theory of Computing*, pages 249–259, New Orleans, Louisiana, 6-8 May 1991.

[3] J.G. Cleary and I.H. Witten. Data compression using adaptive coding and partial string matching. *IEEE Trans. Communications*, 32(4):396–402, April 1984.

[4] A. Fiat and Z. Rosen. Experimental studies of access graph based heuristics: Beating the LRU standard? In *Proceedings of the Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 63–72, New Orleans, Louisiana, 5-7 January 1997.

[5] P. Laird and R. Saul. Discrete sequence prediction and its applications. *Machine Learning*, 15(1):43–68, April 1994.

[6] Z. Rosen. Access graph based heuristics for on-line paging algorithms. Master's thesis, Computer Science Department, Tel-Aviv University, December 1996. Available at `http://www.math.tau.ac.il/~rosen/cola.html`.

[7] N. Young. On-line caching as cache size varies. In *Proceedings of the Second Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 241–250, San Francisco, California, 28-30 January 1991.