On Adaptive Replacement Based on LRU with Working Area Restriction Algorithm

Edson T. Midorikawa, Ricardo L. Piantola, Hugo H. Cassettari
Department of Computer Engineering and Digital Systems
Polytechnic School, University of São Paulo
05508-900 – São Paulo – SP – Brazil
+55-11-3091-5617
edson.midorikawa@poli.usp.br, piantola@uol.com.br, hugohc@terra.com.br

ABSTRACT
Adaptive algorithms are capable of modifying their own behavior through time, depending on the execution characteristics. Recently, we have proposed LRU-WAR, an adaptive replacement algorithm whose objective is to minimize failures detected in LRU policy, preserving its simplicity and low overhead. In this paper, we present our contribution to the study of adaptive replacement algorithms describing their behavior under a number of workloads. Simulations include an analysis of the performance sensibility with the variation of the control parameters and its application in a multiprogrammed environment. In order to address LRU-WAR weakness as a global policy, we also introduce LRU-WARlock. The simulation results show that substantial performance improvements can be obtained.

Categories and Subject Descriptors
D.4.2 [**Operating Systems**]: Storage Management – *Virtual memory,* D.4.8 [**Operating Systems**]: Performance – *Modeling and prediction, simulation*

General Terms
Algorithms, Management, Performance

Keywords
Virtual memory, LRU, demand paging, adaptive replacement

1. INTRODUCTION
One of the most important aspects of a paged virtual memory system is the replacement policy. This policy defines the selection criterion applied to choose one or more pages for memory eviction when another page has to be loaded in memory and there is no space available. The main difficulty is to decide which page is the less important one, so that it can be evicted. The use of an efficient strategy is primordial because a good memory management is an essential feature of high performance systems [20].

There are many different strategies in choosing such a page, but most of them use the past as a prediction of the future. For example, LRU – Least Recently Used – replaces the page that has not been accessed for the longest time, because some correlation between recent past and near future memory accesses is assumed. Today operating systems usually implement some approximation of the LRU algorithm. This policy presents a relatively efficient performance for most applications, compared to other traditional algorithms like FIFO – First In, First Out, MRU – Most Recently Used, and LFU – Least Frequently Used. But LRU presents anomalous behavior with certain workloads.

Jiang and Zhang [14] point out some examples to illustrate LRU deficiencies in certain access patterns with weak locality:

- Sequential accesses over a large number of pages, such as "sequential scans" through a large file, may cause replacement of commonly referenced pages.
- Accesses inside loops with working set size slightly larger than the available memory may replace pages that would be reused soon.
- LRU can not distinguish pages with different access frequency or can not efficiently manage an irregularly accessed page.

On the other hand, it is well known that sequential and loop access patterns can be efficiently managed by MRU. So, it would be interesting to apply the most efficient replacement algorithm for each access pattern [8]. This approach has been used in adaptive page replacement algorithms [27].

The application of the adaptability concept in page replacement algorithms can provide important potential benefits, such as the correction of well-known deficiencies of traditional algorithms.

This paper presents some results obtained from the studies of our research group about adaptive replacement algorithms. After a comprehensive study of the LRU model, we proposed a new adaptive algorithm, named LRU-WAR – LRU with Working Area Restriction [5]. Its objective is to minimize the LRU failures under some access patterns without increasing the overhead of the memory management system. Simulations include an analysis of the performance sensibility with the variation of the control parameters [23] and its application in a multiprogrammed environment. In order to address LRU-WAR weakness as a global policy, we also introduce LRU-WARlock [24] that includes access frequency analysis using a profile based strategy.

The rest of the paper is organized as follows. Section 2 presents a brief overview of related works in adaptive replacement strategies and introduces our research. In Sections 3 and 4, we present the LRU-WAR and LRU-WARlock algorithms, along with results of trace driven simulations. Finally, we finalize the paper in Section 5, presenting some conclusions as well as ongoing and future work.

2. RELATED WORKS

Over the last decade, the scientific community has proposed many different adaptive page replacement algorithms. Most of them adopt as starting point the traditional and relatively efficient LRU algorithm. In situations where it presents good performance, its original behavior is maintained; otherwise, an alternative behavior is adopted.

The SEQ algorithm [12] can be considered an adaptive version of LRU that tries to correct the performance loss caused by the presence of linearly sequential memory accesses. When it identifies one or more memory reference sets to numerically adjacent addresses, the algorithm adopts a pseudo-MRU replacement strategy, otherwise maintaining the original LRU criterion.

Another algorithm is EELRU – Early Eviction LRU [28], which was proposed as an attempt to mix LRU and MRU, based only on the positions on the LRU queue that concentrate most of the memory references. This queue is only a representation of the main memory using the LRU model, ordered by the recency of each page. EELRU detects potential sequential access patterns analyzing the reuse of pages. One important feature of this algorithm is the detection of non-numerically adjacent sequential memory access patterns.

Another important algorithm is LIRS – Low Inter-reference Recency Set [14]. Its objective is to minimizing the deficiencies presented by LRU using an additional criterion named IRR (*Inter-Reference Recency*) that represents the number of different pages accessed between the last two consecutive accesses to the same page. The algorithm assumes the existence of some behavior inertia and, according to the collected IRRs, replaces the page that will take more time to be referenced again. This means that LIRS does not replace the page that has not been referenced for the longest time, but it uses the access recency information to predict which pages have more probability to be accessed in a near future.

There are some adaptive variations based on the 2Q algorithm [16] that try to explore efficiently the frequency and recency of the resident pages. Two examples of algorithms that follow this strategy are ARC – Adaptive Replacement Cache [19] and CAR – Clock with Adaptive Replacement [2]. ARC uses two LRU queues to manage the pages: L1 holds pages accessed only once, i.e. pages with no reuse at the moment, and L2 keeps the other pages, those that were reaccessed at least once. Both queues maintain data regarding some replaced pages (ghost cache), improving its historical knowledge about recently accessed pages. The algorithm has only one control parameter, named *p*, that is self-tuned without any external intervention. This parameter dynamically controls the replacement point in L1 and L2.

CAR can be considered an online version of ARC. It is also based on the Clock algorithm [10], but uses two complementary queues holding information about each page in memory. Both queues are similar to those used in ARC, but are controlled by pointers like Clock. The first queue has replacement priority, and information about recent replaced pages is also maintained in the data structures. The size of both queues is managed dynamically. CAR is also self-tuning, adapting its parameter to increase the hit ratio, and balancing between pages with high recency and frequency.

A variation of CAR, named CART – CAR with Temporal filtering [2] includes an additional mechanism to distinguish constantly referenced pages from pages with "localized reuse", i.e. that have many accesses for a short period of time. This filter classifies pages as "long-term utility" or "short-term utility". Pages from the last group are replaced first when they become inactive.

Another algorithm based on Clock is Clock-Pro [15]. It can be considered an online version of LIRS, adopting three circular and independent pointers to manage only one LRU queue. It seeks to distinguish pages with more or less reuse probability, classifying them as "hot" or "cold". This classification is based on the recent reuse of each page and only cold pages are replaced. Clock-Pro also preserves a ghost cache holding recent replaced pages as an additional reference data structure.

Three other algorithms, DEAR – Detection-based Adaptive Replacement [8], AFC – Application/File-level characterization [9] and UBM – Unified Buffer Management [17], analyze the memory accesses looking for some specific patterns, including sequential accesses. They adopt a different replacement criterion for each pattern. For example, DEAR applies MRU for sequential accesses and LRU or LFU for other patterns.

Recent adaptive algorithms use Artificial Intelligence techniques in order to help them in the adaptation. For example the FPR [25] and FAPR [1] algorithms apply fuzzy inference techniques to manage the replacement priorities of the resident pages.

All these proposals bring important conceptual benefits to the traditional page replacement algorithms, but also present more complex implementations. In many cases, additional data structures to hold nonresident pages are necessary increasing space requirements. Some algorithms require data update in every memory access, making impracticable its real implementation.

2.1 Our Research

Our research group has been studying the application of adaptive strategies on memory management algorithms since we first proposed using application's access patterns to drive the memory management policies in the late 90s [20] [21]. In [22] we investigated the application of adaptive strategies in two variable space algorithms: Working Set and Page Fault Frequency.

More recently, we have developed LRU-WAR [5], an adaptive page replacement strategy based on fixed space algorithms. The standard criterion is LRU, but when the algorithm detects a strong sequential access tendency, it switches to the MRU algorithm. Performance evaluation studies based on trace-based simulations were applied to compare its performance against some other strategies, like LRU, SEQ, EELRU and LIRS. The good performance of LRU-WAR for many workloads motivated us to study new ways to improve it. One solution was tuning its control parameters for each workload [23]. Next, in order to be used in real systems, we analyzed the application of LRU-WAR as a global management policy. As a result, we proposed a new algorithm named LRU-WARlock [24]. The next sections present the main results obtained from our research to develop LRU-WAR.

3. THE LRU-WAR ALGORITHM

The LRU-WAR algorithm presents a new proposal to the solution of the bad performance problem of LRU under some workloads. Based on the maximum working set size [11] between two consecutive page faults, and on the variation of this size between subsequent page faults, the LRU-WAR algorithm (*LRU with Working Area Restriction*) [5] uses a new adaptive technique to address the poor LRU performance in the presence of sequential access patterns: the maximum temporary working set dimension as a decisive factor for the choice of the replacement policy employed at each moment.

3.1 Algorithm Description

LRU-WAR monitors the memory accesses and searches for the referenced page with least access recency between two page faults; in other words, it identifies the highest position in the LRU queue that has been referenced since the last page fault. This position W limits the *working area*.

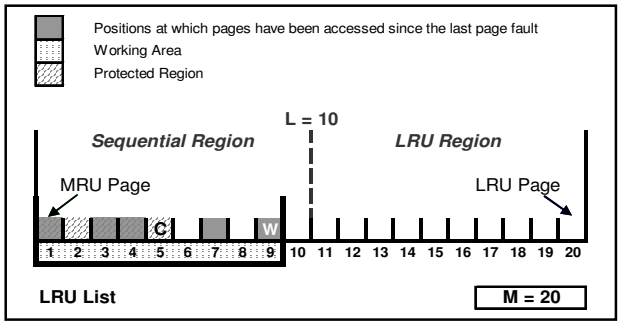


Figure 1. Logic partition of the LRU list used by the LRU-WAR algorithm.

The concept of "working area", exemplified in Figure 1, may be defined as the recency region where the current working set of a program is restricted, i.e., the initial portion of the LRU queue in which all memory accesses have been confined since the last page fault. The working area consists, therefore, in the part of the LRU queue between its first position – MRU page – and its W-th position.

Table 1 summarizes the three execution states associated with the LRU-WAR, where M corresponds to the memory size in number of pages (and, consequently, to the last position of the LRU queue).

Table 1. Execution states of the LRU-WAR algorithm.

Execution State	W	Replacement Policy	Replacement Point
LRU Tendency	> L	LRU	M
Sequential Tendency	≤ L		
Sequential Operating Mode	≤ L and stable	MRU-n	W+1

When the working area exceeds the "sequential region" – the initial partition of the LRU queue limited by the parameter L –, it is considered that the set of memory resident pages has been effectively reused and, hence, a *LRU tendency* (tendency that the LRU policy will work efficiently) is assumed. On the other hand, a *sequential* access *tendency* is detected when, between two page faults, the working area size becomes smaller than L. Until such a tendency is intensified, however, the LRU replacement strategy remains in use, avoiding errors due to hasty changes. When the sequential tendency becomes strong enough, indicating a very low reuse of resident pages in relation to the number of page faults, the LRU-WAR enters in its *sequential operating mode*, starting to utilize a MRU-type replacement strategy.

The parameter C of the algorithm (tendency Confirmation period[1]), defines the called "protected region". This small initial part of the LRU queue contains pages that should not be removed from memory, since they present great reuse potential (minimal expected temporal locality). The parameter C is also used as an additional period of time that must be taken so that the algorithm's execution state goes from sequential tendency to sequential operating mode, consequently reducing the risks related to a possible change in the replacement policy employed.

In practical terms, the LRU-WAR algorithm can be written in pseudo-code as in Figure 2. The default value for C is 5, while the parameter L is specified as MIN(50, M/2) [4].

The code in lines 2-10 is executed when a memory resident page is referenced (page hit). If such a page occupies a position P in the LRU queue that exceeds W, then the working area will increase until that position: W=P. In sequential operating mode, the page fault count is restarted. At this time, the execution state starts to be either LRU tendency or sequential tendency, depending on the size of the new working area. If a decision error is detected, the Tendency Confirmation (TC) period increases according to the number of page replacements in sequential operating mode. We assume that a decision error happens whenever: (1) W increases in sequential operating mode; (2) the page responsible for its increase is among the pages which were not replaced due to the last tendency confirmation period in sequential tendency; and (3) the reuse of this page occurred before it was expected to.

The code in 13-29 lines, in turn, describes the action of the algorithm when a page replacement needs to be made. In original (LRU) tendency, the least recently used page – which occupies the Mth position (last position) in the LRU queue – is replaced and the working area size is reset to zero. In sequential tendency, the LRU replacement criterion is also applied and the working area may only increase in size. Finally, if the algorithm is in sequential operating mode, the page that occupies the W+1th position in the LRU queue is replaced and the working area needs to remain stable in order to conserve this current execution state. It is important to point out that the working area size must always be greater than C (the size of the protected region), what is secured by lines 15 and 16 of the code. The number of page faults since the start of a sequential tendency is calculated by the counter INERTIA, while the counter N is responsible for the storage of the number of page faults in sequential operating mode.

The main idea of the LRU-WAR algorithm is to replace a page that tends to become inactive, i.e., a page that no longer belongs to the program's working set recently. An on-line version was also proposed: the 3P algorithm (*Three Pointers*) [7], which is based on the same idea and the same theoretical principles of the LRU-WAR, thus strongly suggesting the viability of its real implementation in modern operating systems.

[1] In Portuguese, this parameter is named "carência mínima".

```
LRU-WAR ( )
INPUT: The page access sequence.

1.  If the referenced page is in memory:
2.      P = position of the referenced page in the LRU queue;
3.      If P > W:
4.          If N > 0:            /* If the execution state is sequential operating mode */
5.              INERTIA = 0;     /* Finish the sequential operating mode */
6.              If P > W+1 and P ≤ W+TC+1 and (N ≤ M-P or N < 50):
7.                  TC = TC + N;     /* Increase the tendency confirmation (error) */
8.              N = 0;
9.          W = P;     /* Increase the working area size */
10.     Move the referenced page to the head of the LRU queue.
11. Else (the referenced page is not in memory):
12.     If memory is full:
13.         If W ≤ L:
14.             INERTIA = INERTIA + 1;
15.             If W ≤ C:
16.                 W = C + 1;
17.             If INERTIA ≥ W+TC:    /* Sequential operating mode */
18.                 If N < M or N < 50:
19.                     N = N + 1;
20.                 If TC > C:
21.                     TC = TC – 1;
22.                 Remove page at the W+1 position in the LRU queue;
23.             Else (INERTIA < W+TC):    /* Sequential tendency */
24.                 Remove page at the M position in the LRU queue;
25.         Else (W > L):    /* LRU tendency */
26.             INERTIA = 0;
27.             W = 0;
28.             N = 0;
29.             Remove page at the M position in the LRU queue;
30.     Insert the referenced page at the head of the LRU queue.
```

Figure 2. Pseudo-code of the LRU-WAR adaptive algorithm.

3.2 Performance Analysis

In order to evaluate the LRU-WAR performance, a trace-based simulation [30] environment, named Elephantools, was developed [6]. This environment allows using trace files from many sources and formats. For our evaluation tests, we have used the same trace files also used in [28] and [14]. Here we summarize the main results.

Table 2 presents the simulation configurations that were executed in our performance analysis. Each algorithm was analyzed using each trace file for different memory sizes. In summary, for each analyzed algorithm, a total of 527 simulations were executed.

Table 2. Traces and context description.

Trace	Unique Pages	Memory sizes simulated	Number of Simulations
Espresso	77	10, 11, 12, ..., 73, 74, 75	66
GCC	458	10, 15, 20, ..., 445, 450, 455	90
Gnuplot	7718	100, 200, 300, ..., 7500, 7600, 7700	77
Grobner	67	10, 11, 12, ..., 63, 64, 65	56
GS	558	10, 15, 20, ..., 545, 550, 555	110
Lindsay	521	10, 15, 20, ..., 510, 515, 520	103
P2C	132	10, 15, 20, ..., 120, 125, 130	25
		Simulations (total)	527

Table 3 presents a performance comparison between LRU-WAR and LRU showing the percentage of difference related to the number of page faults for each trace. Negative values indicate that the LRU-WAR algorithm presents a better performance than LRU, because it causes a smaller number of page faults. Positive values, obviously, indicate the contrary.

The “best case” column reflects the execution situation in which LRU-WAR obtains the best performance, in comparison with LRU. The opposite situation is illustrated by the “worst case” column – when LRU reaches the best performance. The arithmetic average of the differences collected with both algorithms, taking into consideration all the simulations described in Table 2, is calculated on the third column.

Thus, for example, we can interpret the data related to the Grobner application in the following way: LRU-WAR, in average, achieves a performance 3.78% better than LRU. In the execution with a 28-page memory, it shows a performance 11.75% better. However, it can cause a number of page faults up to 1.02% larger, which happens specifically with a 13-page memory size. But, in general, the average column indicates that LRU-WAR has a tendency of having better results than LRU. This result can be explained from the fact that Grobner presents a mixed memory access pattern where some sequential accesses can be detected and explored by LRU-WAR. Other similar results were obtained for Espresso, Gcc, Lindsay and P2c.

Table 3. LRU-WAR Performance in comparison with LRU.

Trace	Best Case	Worst Case	Average
	Diference % (Mem.)	Diference % (Mem.)	
Espresso	-7.94% (12)	0.02% (27)	-0.28%
GCC	-9.66% (185)	0.18% (245)	-1.24%
Gnuplot	-66.22% (7700)	-0.53% (100)	-33.38%
Grobner	-11.75% (28)	1.02% (13)	-3.78%
GS	-5.61% (155)	5.09% (300)	1.34%
Lindsay	-17.38% (. . .)	1.74% (130)	-8.60%
P2C	-0.72% (50)	0.04% (75)	-0.14%
VMTrace package	**-66.22%**	**5.09%**	**-6.58%**

The program Grobner represents a typical situation: when LRU has a good performance, LRU-WAR is usually better than LRU, but not much better. On the other hand, when LRU shows a very poor performance, LRU-WAR can present excellent results. One such example is Gnuplot. It exhibits an effective sequential memory access pattern that predominates in its entire execution (Figure 3). LRU-WAR presented better performance for all memory sizes, with improvements of up to 66.2% for a memory of 7700 pages.

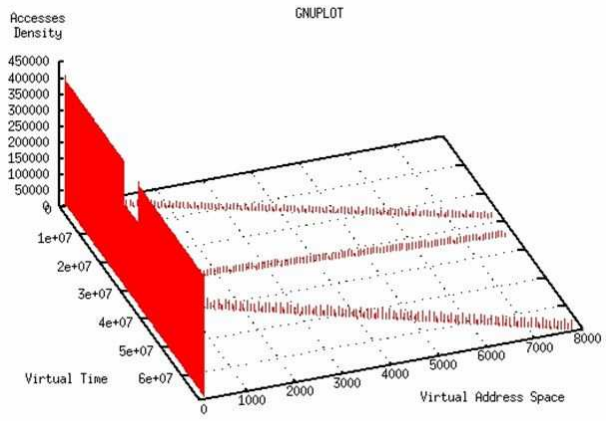


Figure 3 – Gnuplot memory access pattern.

From the analyzed traces, LRU-WAR presented the worst performance for the GS (Ghostscript) program. Although LRU-WAR presented performance improvements on a number of memory sizes, in average its performance was 1.34% worse than LRU. The reason was the absence of strong sequential access patterns in this trace.

Figure 4 shows the LRU-WAR performance over two applications which have a very well defined sequential access pattern (Gnuplot and Glimpse) and one with a good temporal locality (Lindsay). In these applications, LRU-WAR reaches an excellent performance.

The first trace belongs to Gnuplot, it is a plotting utility. LRU-WAR almost hit the best possible performance from OPT algorithm. It achieved improvements of up to 66.2% over LRU. LRU-WAR detected very quickly the memory sequential access pattern, and took advantage of it.

Lindsay is a hypercube simulator. This trace has a good temporal locality, so LRU also has a good performance. LRU-WAR presents a better performance, from the fact that it can detect some loop access patterns and enter in *sequential operating mode*.

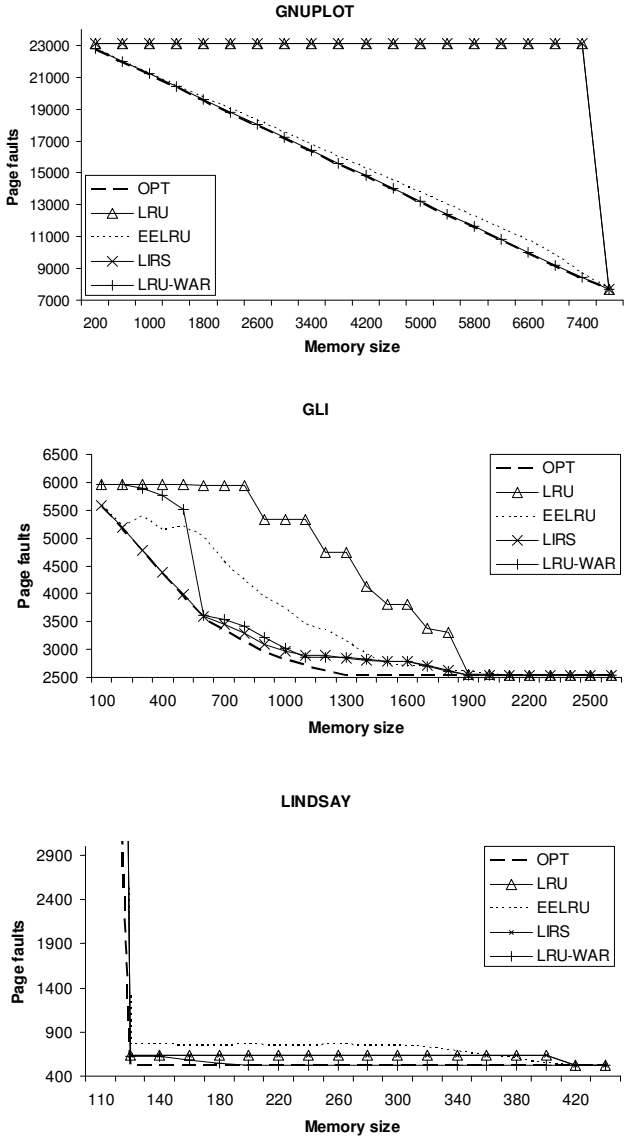


Figure 4. Replacement algorithms performance comparison with Gnuplot, Glimpse and Lindsay.

Glimpse is text information retrieval tool. Like Gnuplot, Glimpse has a very well defined sequential access pattern. Therefore, LRU-WAR had exceptional accuracy detecting such patterns in this trace.

These simulations show the LRU-WAR good working over LRU and other very nice adaptive algorithms. From the obtained simulation results, it is possible to conclude that LRU-WAR is substantially better than LRU when there are sequential access patterns and it is not much worse in any of the analyzed situations. In the best scenario, LRU-WAR has achieved a performance 66.22% better than LRU's. Nevertheless, its worst performance

has generated a difference of only 5.09% on the page faults, with regard to LRU, among a total of 527 simulations. More information and detailed analysis about LRU-WAR behavior can be found in [4], [5], [23] and [24].

3.3 Control Parameters Sensibility

In a program with sequential access patterns, LRU-WAR reaches an excellent performance. Previous papers show such a behavior in local memory management systems [5] [6]. The papers also show that it is possible to make it better by adjusting its parameters [23].

This subsection presents a study of the influence on the behavior and the performance of adaptive page replacement algorithms according to a variation of the control parameters. The obtained results for the LRU-WAR algorithm show that a dynamic control of these parameters can provide some interesting benefits in the execution of some applications.

3.3.1 Control Parameters

Adaptive page replacement algorithms are characterized, over all, by modifying its substitution criteria during execution. Such behavior is managed through control parameters previously defined and delimited, as the project and the particular strategy of each algorithm with intention to diminish to the maximum the occurrence of page faults. The control parameters are, therefore, an important part in the project of any adaptive algorithm. Table 4 has a brief description of the control parameters of the SEQ, EELRU, LIRS, CAR and LRU-WAR algorithms.

Table 4. Control Parameters of some algorithms.

Algorithm	Parameter	Brief Description
SEQ	L	Minimum sequence size so that it can shelter a substitution point.
	M	Potential substitution point of a sequence (MRU-M).
	N	Amount of considered pages to verify the growth tax of a sequence.
EELRU	E	Early eviction point.
	L	Late eviction point (LRU).
LIRS	L_{hirs}	Maximum percentage of HIRs pages in relation to the memory size.
CAR	P	Limit value of L1 queue.
LRU-WAR	C	Protected Region size.
	L	Sequential Region size.

3.3.2 Performance Analysis

We present here the results of an analysis of control parameters variation in the LRU-WAR algorithm. A complete description of the study can be found in [23]. We use here the program Grobner, described along with some details. The program Grobner is characterized by possessing some parts with sequential behavior interlaced with references to many other pages, using almost all pages in its virtual memory space. LRU-WAR performance with default parameters is slightly better than LRU (figure 5). In such a way, algorithms that try to detect sequential access behavior can achieve good results.

When analyzing *tendency confirmation period* variation effect (C parameter), we verify that the smaller the C value is, the faster LRU-WAR can detect the Grobner sequential memory access behavior and, therefore, we can notice improvement in performance (figure 5). The reason for this behavior is that, after being in *sequential tendency*, the smaller the *tendency confirmation period* is, the faster the algorithm enters *sequential operating mode*.

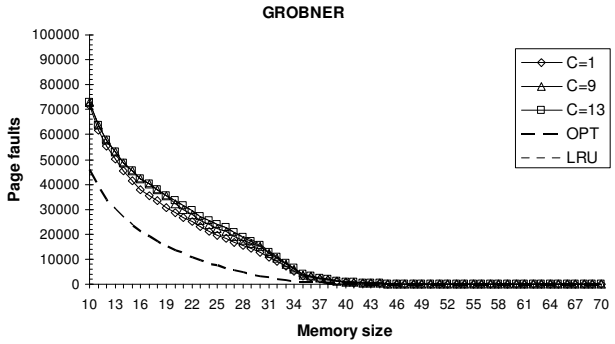


Figure 5. Performance analysis under parameter variation.

If C=1, LRU-WAR surpasses LRU in all memory sizes until 44 pages, when the page faults converge to all studied C values. The best improvement was achieved for 35-page memory size, with almost 21% of performance gain compared with LRU. In comparison with LRU-WAR default values, *tendency confirmation period* C=1 also presents a better performance, with gains up to 13% for 38-page memory. For C=9 and C=13, LRU-WAR shows a behavior tendency towards to LRU, however, still getting satisfactory results in some memory sizes.

For the simulation that studies the L parameter variation, we defined fixed values for the Sequential Region size until 10, due to the program total number of pages.

The results are presented in figure 6. The L parameter variation only changes the moment in which the algorithm changes its state from *LRU tendency* to *sequential operating mode.* This results a similar behavior in all scenarios.

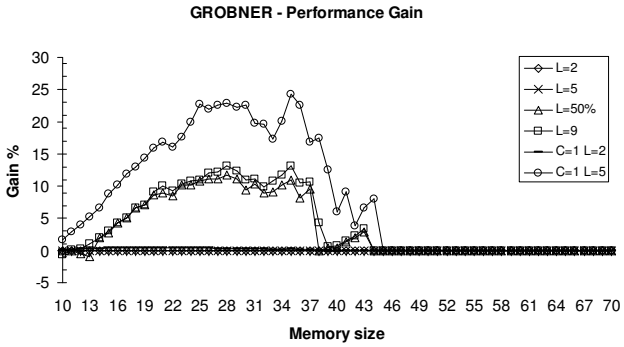


Figure 6. LRU-WAR comparison with LRU.

Next, we decided to test a variation with both parameters simultaneously due to the performance gain achieved with individual C and L variation. The best result was obtained with C=1 e L=5 with a performance gain of almost 24% in comparison

with LRU for 35-page memory. This value represents a gain of 15% with regard to LRU-WAR default parameters (figure 6). The obtained results reveal that it is possible to optimize the LRU-WAR performance, according to the program access pattern.

4. THE LRU-WARlock ALGORITHM

Operating systems generally use global policies for memory management. The adaptive strategies have as principle to adapt their behavior based on programs memory access patterns. Their use in multiprogrammed environment has not been properly examined in previous studies. This subsection intends to present a strategy to adjust LRU-WAR in order to obtain a good performance in a global memory management system. The results indicate that the same strategy can be used in algorithms following the same adaptive principle that LRU-WAR does [24].

4.1 Using Adaptability in Global Policies

Every page replacement policy in a demand paging system must choose a victim in order to give place to a new referenced page when there is not any room left in the memory [3]. If the chosen page is always from the same process that has suffered the page fault, the policy used by the system is called local. Nonetheless, it is possible that the memory management system works in a global manner. A global replacement algorithm works with all the pages in the memory and chooses the victim independently from the process that has suffered the page fault [26].

There are some advantages in using global policies in memory management systems. Some processes may need more memory than others. In this case, the global policy can correctly measure the amount of memory for each process. The blocked processes – waiting for a service – keep pages in the memory using space that could be used by pages from other processes. In a system with a global policy, that is minimized.

Most of the "traditional" adaptive algorithms are inadequate for a global policy memory system. The design of those algorithms aims to analyze an access pattern of one application only. When the accesses are counted in a global way, that is, the memory references are collected from multiple applications, the algorithms behave inefficiently. A lot of those algorithms – including LRU-WAR – focus rather on the access recency than on the frequency. The access frequency analysis is one important factor for a global policy.

4.2 LRU-WAR Modification Proposal

The LRU-WAR modification main objective is to obtain a good performance in systems with a global memory policy management, where the detection of access patterns becomes more difficult. In order to accomplish this objective, the LRU-WARlock algorithm – that complements LRU-WAR – was created, without modifying its original idea of sequential memory access exploration.

The LRU-WARlock working principle has emerged from the idea of separating the accesses that do not belong to the sequential access pattern and deal with them in another way. The crucial point was finding out what disturbed the sequential references detection the most. The answer was in the LRU general good working. In a multiprogrammed environment with a global memory paging management system, the memory is composed by pages from different programs. In such scenario – with many working sets [11] coming from different programs in the memory – it is possible to find out several different access patterns. For instance: lots of pages with a few accesses and some pages with a great amount of references. The solution in this scenario would be to remove the working sets pages from the LRU-WAR sequential pattern detection, that is, to separate the most frequently accessed pages.

The mechanism used by LRU-WARlock is the reservation of part of the memory for the most accessed pages. The other part of the memory keeps the original LRU-WAR management, not considering the reserved part. A parameter called K, which is related to the percentage of the memory that is reserved to the pages with high access frequency, was created and can be directly controlled by the operating system.

When it comes to determine what the most accessed pages are, the profiling technique was the chosen one. Such technique makes it possible to capture information in run-time and use this knowledge in future executions. Some compilers have already been using this technique to provide optimization in the executable code generation. Intel C++ compiler uses the technique named Profile-Guided Optimization (PGO) [13] which indicates what parts of the application are the most frequently executed, directing thus the optimization focus.

4.3 Algorithm Description

In LRU-WARlock, memory pages are logically divided in two parts: one managed by non-modified LRU-WAR and a new part that uses the profiling-guided technique, adding the new control parameter K.

The part concerning LRU-WAR continues to monitor the memory access, between two consecutives page faults, using the working set maximum size as adaptability deciding factor. Three possible states of execution are defined as in the original LRU-WAR: *LRU tendency*, *sequential tendency* and *sequential operating mode*.

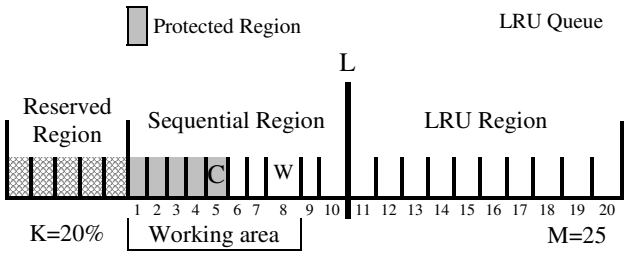


Figure 7. Logical partition of memory pages in the LRU-WARlock algorithm.

The algorithm adopts a new logical memory partition (Figure 7), with the Reserved Region, which is composed by the most referenced pages and is out of the entire processing executed by LRU-WAR code. The Reserved Region size is defined by the new control parameter K in percentage of the total of pages.

Figure 8 shows the algorithm pseudo-code, highlighting the additions to the original LRU-WAR code. The new conditions added to the code are in the lines 8 and 22: if the accessed page is neither in the memory nor in the profile, then LRU-WAR is executed; if the page is not in the memory but it is in the profile,

```
LRU-WARlock( )
INPUT: The page access sequence.

1.    If  referenced page is in memory:
2.          If "page locked" bit equals zero:
3.                If it is in Sequential Operating Mode:
4.                      Finalize Sequential Operating mode
5.                      Increase  Tendency Confirmation
6.                Increase  the Working Set
7.                Place referenced page in first queue position
8.    Else, if  referenced page is neither in memory nor in profile:
9.          If memory is full:  /*  total memory – reserved memory  */
10.               If it is in Sequential Operating Mode:
11.                     Remove page in (Working Set + 1) queue position
12.               Else, if it is in Sequential Tendency:
13.                     If the Working Set is lesser then L and exceed Tendency Confirmation
14.                           Beginning Sequential Operating Mode
15.                     Remove  page in LRU queue position
16.               Else:   /*  It is in LRU Tendency  */
17.                     If the Working Set is lesser then L:
18.                           Beginning Sequential Tendency
19.                     Decrease the Working Set and adjust Tendency Confirmation
20.                     Remove page in LRU queue position
21.         Load referenced page in first queue position
22.   Else, if it is not in memory and it is in profile:
23.         Load referenced page
24.         Set "page locked" bit to 1
```

Figure 8. Pseudo-code of the LRU-WARlock algorithm.

then the page is allocated in the Reserved Region. When the page is allocated in the Reserved Region, its "page locked" bit is activated. In this case, the page will not be removed from the memory until the program finishes its execution. In the pseudo-code, this task is in the lines 23 and 24.

The different treatment provided by LRU-WARlock – classifying the access to the memory according the access pattern – allows a better LRU-WAR adaptation in multiprogrammed environments. Another advantage is that if the accessed page is in the Reserved Region, it is not necessary to execute any adjustment in the parameters while LRU-WAR executes the steps from 3 to 7.

4.4 Performance Analysis

In this subsection we present an analysis of the adaptive page replacement algorithm LRU-WARlock. We start by describing the applications traces and tools used in the evaluation. After that, we present an analysis of the results obtained through the tests.

Table 5. Traces description.

Trace	Applications	Total of references	Total of pages
multi1	Cscope and cpp.	15858	2606
multi2	Cscope, cpp and postgres.	26311	5684
multi3	cpp, gnuplot, glimpse and postgres.	30241	7454

The LRU-WARlock evaluation was done by using three different applications traces: multi1, multi2 and multi3 (Table 5). The three traces [17] were selected because they contain simultaneous applications accesses, simulating a multiprogrammed environment. The tools used in the simulations are part of the Elephantools environment [4] [6].

Multi1 is composed by memory accesses from two applications. Cscope memory access pattern refers to looping with strong temporal locality and other different references. In Cpp, we can observe sequential memory reference blocks, along with other references. Trace multi1 interleaves accesses from both of these applications, one with looping references, and the other with sequential accesses. This access pattern jeopardizes in a great deal the LRU performance [14]. The individual programs present an access pattern adequate to LRU-WAR. However, as they are interleaved by multiprogramming, they lessen considerably the algorithm performance.

Table 6. Applications description.

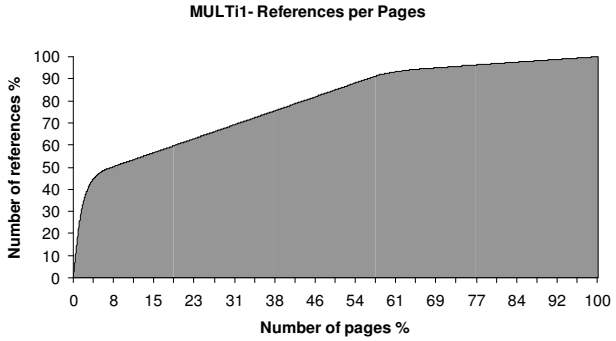
Application	Description
Cpp	GNU C compiler pre-processor.
Cscope	C source examination application.
Glimpse	Text information retrieval tool.
Gnuplot	Plotting program.
Postgres	Relational database system.

Trace multi2 has the same composition as trace multi1 with the addition of the program Postgres. Postgres presents a sequential and looping access patterns with non-constant periods. The increase of the multiprogramming level makes it more difficult to detect sequential patterns by LRU-WAR.

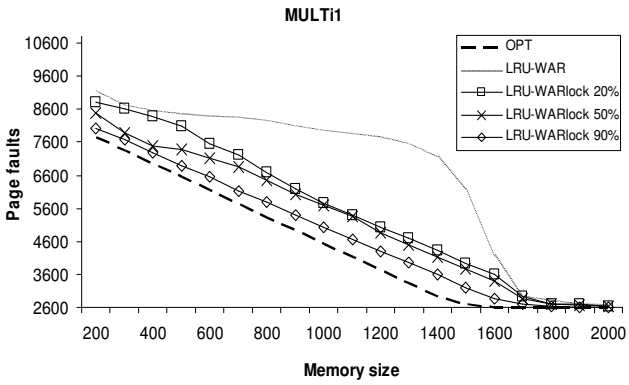
The third trace used, multi3, has the configuration slightly different from the first two. It is composed by four applications, so, memory reference pattern is very different. It includes Gnuplot that has a sequential access pattern, Cpp, Glimpse and Postgres. But also due to the increase of the multiprogramming level, it's much more difficult for LRU-WAR to detect sequential patterns.

4.4.1 Multi1

Since trace multi1 presents accesses of two interleaved applications in a global memory management system, LRU-WAR does not have a very good performance, even with the two programs that compose multi1 having sequential accesses (Figure 9).



(a) Page access frequency profile.



(b) Algorithms performance.

Figure 9. Multi1 analysis.

All the performance tests were done by changing the parameter K from 10 to 90%, but here we present only the results obtained using 20, 50 and 90%. Trace multi1 has less pages and less memory references in comparison with traces multi2 and multi3, because it has only two programs included in the trace. However, this does not make it less important.

In multi1, the page access frequency profile analysis show that only 7.5% of the pages are responsible for 50% of the memory accesses (Figure 9.a). When the parameter K=20, LRU-WARlock surpasses LRU-WAR in all memory sizes, with a performance improvement of almost 16%. The performance peak was obtained for a memory of 1400 pages, with an improvement of 39.3% comparing to LRU-WAR.

With the control parameter K=50, the Reserved Region occupies 50% of the memory. One interesting characteristic of LRU-WARlock is concerned to the small sized memories, because in this case it is guaranteed that half of the accesses are done with only 400 memory pages for a trace such as multi1, with 2606 different pages. The performance improvement was, in average, 19% and had a peak of 42% in comparison to LRU-WAR.

When the Reserved Region has a size of 90% of the memory, the highest performance continues to be the one with memory size of 1400 pages, which is 50% better than LRU-WAR performance. When it comes to the average performance, the difference is 26% (Figure 9.b).

An important fact was observed during the traces analysis and it concerns the amount of time in which LRU-WAR remains in the *Sequential Operating Mode*. In all the studied cases, the period in which the algorithm works detecting sequential accesses has increased. For the multi1, LRU-WAR remained about 11% of the time in sequential operation, while LRU-WARlock, with K=50, had increased it five times, being on this mode 56% of the time. With K=90, LRU-WARlock stayed 86% of the time in *Sequential Operating Mode*. Such a result shows that with this strategy it is easier to detect sequential accesses in the memory access, improving its efficiency.

4.4.2 Multi2

The most important characteristic of trace multi2 in the analysis comes from the fact that it is similar to multi1. The difference between them is the adding of the program Postgres, increasing, thus, the multiprogramming level. LRU-WAR performance lessens, because it becomes more difficult to distinguish access patterns with the inserting of new interleaved references. LRU-WARlock strategy also guarantees a good performance for this trace (Figure 10). Algorithms using techniques to separate patterns can obtain good results.

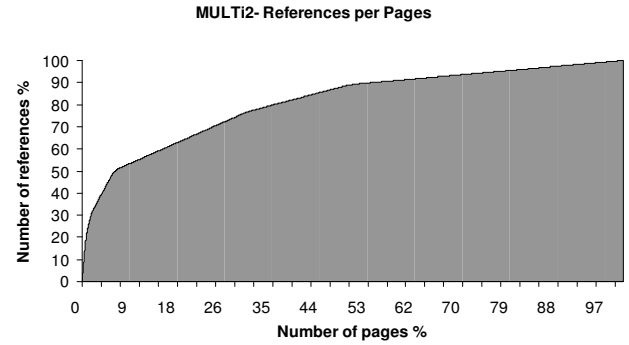
Pages access frequency profile analysis shows that half of the 26311 references are done by only 348 pages, in a total of 5684 pages, that is, only 6% of the pages are responsible for 50% of the memory access (Figure 10.a).

When K=20, LRU-WARlock surpasses LRU-WAR with all memory sizes, with an average improvement of 11%, not considering memory sizes bigger than 3200, when the page fault curves converge in all the analyzed algorithms (Figure 10.b). The largest performance improvement was obtained with a 2000-page memory. There was an improvement of 27% in comparison to LRU-WAR and almost 31% in comparison to LRU with 2200-page memory.

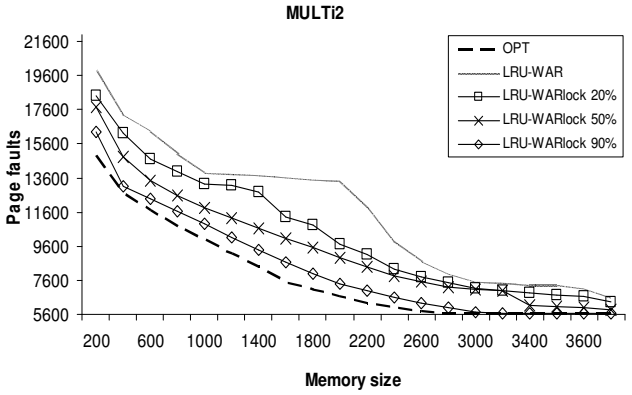
When K=50, working with half of the memory for *Reserved Region*, LRU-WARlock presents an interesting behavior. The curve, which we can see on the graphic, starts to be parallel to the OPT algorithm, lessening once more its identity with LRU. With this new characteristic, average improvement increases considerably to 18%, obtaining a better tolerance for all memory sizes. A clear example of this behavior is the improvement obtained with 1400-page memory. When K=20, there was an improvement of 7% and when K=50 with the same memory size

the improvement triples: 22% comparing to LRU-WAR. The performance peak was found in the simulation with 2000-page memory, when the improvement comparing to LRU-WAR was of 33% (when K=20, 27% of improvement).

When K=90, LRU-WARlock surpassed LRU-WAR with all memory sizes with a superior advantage of almost 30%. When it comes to the similarity of the test between the tested algorithm and the OPT, one fact must be well observed when there is a 400-page memory: LRU-WARlock performance was very close to the OPT theoretical performance, with a difference of only 3.35% of the total page faults. The best performance occurs with a 2000-page memory, being almost 45% better than LRU-WAR (Figure 10.b). These results show that LRU-WARlock is appropriate for systems with global memory policy.



(a) Page access frequency profile.



(b) Algorithms performance.

Figure 10. Multi2 analysis.

4.4.3 Multi3

Multi3 consists of four programs with diverse interleaved access patterns. In this case, LRU-WAR algorithm makes wrong decisions usually going on *Sequential Operating Mode* and, thus, it has a performance worse that LRU for some memory sizes. Since there is no temporal locality uniformity, pages that will be accessed in the near future are prematurely discarded. For this sequential reference LRU-WARlock also surpasses LRU-WAR (Figure 11).

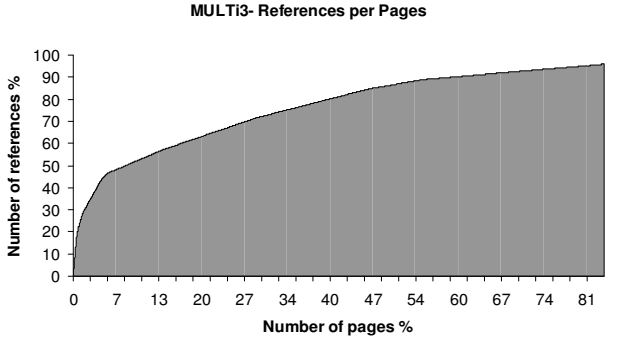
A variation of K has shown that the bigger the reserved area to the most frequent pages, the better is the sequential pages detection by the algorithm, creating, thus, a pattern division (Figure 11.b).

When K=20, the average improvement comparing to LRU-WAR is of 10.5% (only 0.5% less than when it is compared to multi2). The performance peak could be reached in the simulation, with memory size of 2800 pages, when the performance was almost 22% better than with LRU-WAR.

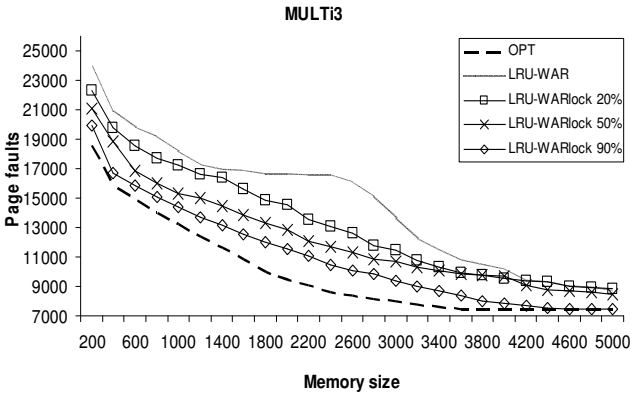
When K=50, the performance increase is linear comparing to K=20. There is an average improvement 17% better than LRU-WAR and a peak of almost 30% with 2600-page memory. The curve loses the characteristic inherited by the LRU and it lines up to the curve of the OPT algorithm.

When K=90, the average performance improvement is more constant: 26% when compared to LRU-WAR. The best improvement was obtained with 2600-page memory: 37%.

An interesting point is related to the memory size of 4000 pages, which had the best performance improvement with parameter K set to 20: it was multiplied by 3.85, having an improvement between 6% and 23%.



(a) Page access frequency profile.



(b) Algorithms performance.

Figure 11. Multi3 analysis.

This behavior can be explained by the fact that the traces' most accessed pages belong to each program's working set. Those pages are not many, but they are very much accessed. In the trace multi3, 50% of the 30241 references are related to only 617 pages of the 7454 pages that compose the trace, i.e. a little more than 8% of the total number of pages(Figure 11.a). This way, a lot of pages that will seldom be reaccessed will not be placed in the space allocated to the Reserved Region, which will make it easier the detection of sequential patterns.

5. CONCLUSION AND FUTURE WORK

This paper has presented a brief description of our research studies developing LRU-WAR, an adaptive page replacement algorithm. We included the concept of Working Area as well as a new technique to analyze applications' memory access pattern in order to identify sequential accesses.

Trace based simulation studies revealed the effectiveness of LRU-WAR, especially with applications that exhibit sequential access patterns. It was possible to obtain performance improvements of up to 66.2% over LRU for the Gnuplot application.

Another contribution is related to the study of control parameter sensibility in adaptive algorithms. Although the default values are set to obtain good performance for an average workload, our study demonstrated that, in certain workloads, it is possible to tune the parameters' values. Even for workloads that are not adequate for LRU-WAR, it was possible to get interesting improvements.

One of the most important contributions was to show that, in spite of having been designed for systems that use local memory management policies, the adaptive algorithms can be adjusted to have a good performance in systems with global memory management as well. Results show that LRU-WAR, when modified for a global management policy (LRU-WARlock) is a good alternative and can be dynamically adjusted according to the memory management system needs. LRU-WARlock presented performance improvements of up to 49.6% in comparison to LRU-WAR and 55.9% in comparison to LRU.

Besides access frequency, other kind of information could be given to the replacement algorithm to help memory management. Examples include the memory access pattern, the working set composition and the last reference of each page. Studies to verify which information is more relevant are already being conducted.

Some complementary studies can be developed in order to continue this work. A suggestion to consider is including in LRU-WAR a mechanism that deals with the access frequency in a dynamic way without profiling, and also that does not make its implementation unfeasible. A second suggestion is repeating the study developed with LRU-WARlock on other adaptive page replacement algorithms so that it is possible to find out a general model for adjusting the algorithms to work with global memory management policies.

With the current availability of multi-core processors, it is important to develop new algorithms to efficiently manage the memory allocated to parallel applications. Multithreaded heterogeneous applications have complex memory access characteristics: each thread can have a different access pattern and shared pages could be used for many cores during different periods of time. Current approaches deal with each thread separately. An integrated approach must be developed in order to manage parallel applications' pages as a unique set.

6. ACKNOWLEDGMENTS

We would like to thank the anonymous referees of our previous papers for their comments and suggestions. We are also grateful to Yannis Smaragdakis and Song Jiang to provide us with their traces and simulators used in their papers [28], [14] and [15].

7. REFERENCES

[1] Bagchi, S., Nygaard, M. 2004. A Fuzzy Adaptive Algorithm for Fine Grained Cache Paging. Proceedings of the 8th International Workshop (SCOPES'04), Amsterdam, The Netherlands, 200-213.

[2] Bansal, S. and Modha, D. S. 2004. CAR: Clock with Adaptive Replacement, In Proceedings of the USENIX Conference on File and Storage Technologies (FAST'04), San Francisco, 187-200

[3] Belady , L. A. 1966. A Study of Replacement Algorithms for Virtual Storage, IBM System Journal.

[4] Cassettari, H.H. 2004. Program Locality Analysis and Development of Adaptive Page Replacement Algorithms. MSc Thesis. Polytechnic School of University of São Paulo, São Paulo, SP, Brazil. *In Portuguese*.

[5] Cassettari, H.H. and Midorikawa, E.T. 2004. The LRU-WAR adaptive page replacement algorithm: exploring the LRU model with sequential access detection. In Proceedings of the XXIV Congress of the Brazilian Computer Society – I Workshop on Operating Systems (WSO 2004). SBC, Salvador, BA, Brazil, CD-ROM. *In Portuguese*.

[6] Cassettari, H.H. and Midorikawa, E.T. 2004. Workload characterization in studies of virtual memory management. In Proceedings of the Third Workshop on the Performance of Computational and Communication Systems (WPerformance 2004), Salvador, Brazil, CD-ROM. *In Portuguese*.

[7] Cassettari, H.H. and Midorikawa, E.T. 2005. The 3P page replacement algorithm: adding adaptability to the Clock policy. In Proceedings of the XXV Congress of the Brazilian Computer Society – II Workshop on Operating Systems (WSO 2005). SBC, São Leopoldo, RS, Brazil, CD-ROM. *In Portuguese*.

[8] Choi, J. et al. 1999. An Implementation Study of a Detection-Based Adaptive Block Replacement Scheme. In Proceedings of the USENIX Annual Technical Conference, 239-252.

[9] Choi, J. et al. 2000. Towards application/file-level characterization of block references: a case for fine-grained buffer management. In: Proceeding ot the 25th International Conference on Measurement and Modeling of Computer Systems, Santa Clara, CA. (SIGMETRICS'00), 286-295.

[10] Corbató, F. J. 1968. A paging experiment with the Multics system. In Honor of P. M. Morse, pp. 217–228, MIT Press, 1969. Also as MIT Project MAC Report MAC-M-384.

[11] Denning, P.J. 1968. The working set model for program behavior. Communications of the ACM, 11, 5 (1968), 323-333.

[12] Glass, G. and Cao, P. 1997. Adaptive Page Replacement Based on Memory Reference Behavior, In Proceedings of the ACM International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS'97), Seattle, 115-126.

[13] Intel. 2008. Profile-Guided Optimizations Overview. http://www.intel.com/software/products/compilers/docs/flin/main_for/mergedprojects/optaps_for/common/optaps_pgo_ovw.htm. Accessed: March 16, 2008.

[14] Jiang, S. and Zhang, X. 2002. LIRS: An Efficient Low Inter-Reference Recency Set Replacement Policy to Improve Buffer Cache Performance, In Proceedings of the ACM International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS'02), Marina Del Rey, 31-42.

[15] Jiang, S., Chen, F., Zhang, X. 2005. CLOCK-Pro: An effective improvement of the CLOCK replacement. In Proceedings of the 10th Annual USENIX Technical Conference, Anaheim, 2005. (USENIX'05), 323-336.

[16] Johnson, T., Shasha, D. 1994. 2Q: a low overhead high performance buffer management replacement algorithm. In Proceedings of the International Conference on Very Large Databases (VLDB' 94), Santiago, 439-450.

[17] Kim, J.M. et al. 2000. A low-overhead high-performance unified buffer management scheme that exploit sequential and looping references. In: Symposium on Operating System Design and Implementation, 4., San Diego. OSDI' 2000: Proceedings. San Diego: USENIX, 119-134.

[18] Lee, D. et al. 2001. LRFU: a spectrum of policies that subsumes the Least Recently Used and Least Frequently Used policies. IEEE Transactions on Computers, vol.50, n.12, 1352-1361.

[19] Megiddo, N. and Modha, D. S. 2003. ARC: A Self-Tuning, Low Overhead Replacement Cache, In Proceedings of the USENIX Conference on File and Storage Technologies (FAST'03), San Francisco, 115-130.

[20] Midorikawa, E.T. 1997. A new strategy for the memory management for high-performance systems. PhD Thesis. Polytechnic School of University of São Paulo. *In Portuguese*.

[21] Midorikawa, E.T., Sato, L.M.., and Zuffo, J.A. 1999 Communion: a new strategy for memory management in high-performance computer systems. Journal of Computer Science and Technology. 1, 1 (March 1999), 16p.

[22] Midorikawa, E.T., Gastaldo, D.L., Cassettari, H.H., and Gerbati, S.R.M. 2000 Adaptive page replacement algorithms for the next generation of virtual memory systems. In Proceedings of the Third Workshop of Computation (WorkComp'2000). São José dos Campos, Brazil. pp.101-107. *In Portuguese*.

[23] Midorikawa, E.T., Piantola, R.L., Cassettari, H.H. 2007. Control Parameters Influence in the Adaptive Page Replacement Algorithms Performance. In Proceedings of the XXVII Congress of the Brazilian Computer Society – IV Workshop on Operating Systems (WSO 2007). SBC, Rio de Janeiro, RJ, Brazil, CD-ROM. *In Portuguese*.

[24] Piantola, R.L., Midorikawa, E.T. 2008. Adjusting LRU-WAR for a Global Memory Management Policy. In Proceedings of the XXVIII Congress of the Brazilian Computer Society – V Workshop on Operating Systems (WSO 2008). SBC, Belém, PA, Brazil, CD-ROM. *In Portuguese*.

[25] Sabeghil, M. and Yaghmaee, M. H. 2006. Using fuzzy logic to improve cache replacement decisions. IJCSNS International Journal of Computer Science and Network Security, Seoul, v.6, n.3A, 182-188.

[26] Silberschatz, A., Galvin, P.B., and Gagne, G. 2005. Operating system concepts. Wiley, 7th edition.

[27] Smaragdakis, Y. 2004 General adaptive replacement policies, Proceedings of the 4th international symposium on Memory management, October 24-25, Vancouver, BC, Canada.

[28] Smaragdakis, Y., Kaplan, S., and Wilson, P. 1999. EELRU: Simple and Effective Adaptive Page Replacement, In Proceedings of the ACM International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS'99), Atlanta, 122-133.

[29] Smith, A.J. 1978. Sequentiality and prefetching in database systems. In Proceedings of the ACM Transactions on Database Systems, v.3, n.3, 223-247.

[30] Uhlig, R.A., Mudge, T.N. 1997. Trace-driven memory simulation: a survey. Proceedings of the ACM Computing Surveys, v.29, n.2, 128-170.