Algoritmo Adaptativo de Substituição de Páginas LRU-WAR: Exploração do Modelo LRU com Detecção de Acessos Seqüenciais

Hugo Henrique Cassettari, Edson Toshimi Midorikawa

Escola Politécnica da Universidade de São Paulo (EPUSP)
Departamento de Engenharia de Computação e Sistemas Digitais (PCS)
05508-900, São Paulo-SP

{hugo.cassettari,edson.midorikawa}@poli.usp.br

Abstract. *Page replacement algorithms have direct influence on the performance of computing systems which use paged virtual memory. Adaptive algorithms are capable of modifying their own behavior through time, depending on the execution characteristics. This paper presents a new adaptive algorithm for page replacement: LRU with Working Area Restriction (LRU-WAR), whose target is to minimize failures detected in LRU algorithm, preserving its low overhead.*

Resumo. *Algoritmos de substituição de páginas influenciam diretamente o desempenho de sistemas computacionais que utilizam memória virtual paginada. Algoritmos adaptativos têm a capacidade de modificar o seu comportamento ao longo do tempo, de acordo com as características de processamento observadas. Este artigo apresenta um novo algoritmo adaptativo para substituição de páginas, cujo objetivo é minimizar falhas detectadas no algoritmo LRU sem perder a sua simplicidade computacional: o LRU-WAR (LRU with Working Area Restriction / LRU com Confinamento da Área de Trabalho).*

1. Introdução

Um bom gerenciamento de memória é característica indispensável para um sistema operacional executar satisfatoriamente tarefas que exigem alto desempenho (MIDORIKAWA, 1997). Um dos aspectos mais importantes em um sistema de memória virtual paginada é a política de substituição de páginas utilizada. Esta política define os critérios de seleção empregados para a retirada de uma ou mais páginas da memória principal quando uma outra precisa ser alocada e não há espaço disponível.

O esquema de substituição de páginas mais utilizado em sistemas operacionais modernos é aquele que implementa alguma forma de aproximação do algoritmo LRU (*Least Recently Used* – o qual substitui a página residente acessada há mais tempo). Esta política se mostra relativamente eficiente na grande maioria dos casos, em comparação com outras políticas tradicionais como FIFO (*First In, First Out* – página carregada há mais tempo), MRU (*Most Recently Used* – página acessada há menos tempo), LFU (*Least Frequently Used* – página que recebeu menos acessos), etc. Isto porque os programas normalmente não apresentam padrões de acesso bem definidos, ou seja, é difícil prever que página será ou não acessada brevemente. Nessa situação, o emprego do critério LRU tende a ser o mais lógico dentre as políticas de substituição triviais. Afinal, se não há mecanismos suficientes para se prever qual página levará mais tempo para ser requisitada pelo processador, substitui-se, em caso de falta, a página que está há mais tempo sem ser acessada, aquela menos ativa no momento da execução.

Porém, Jiang; Zhang (2002) apontam situações em que certamente o algoritmo LRU apresenta deficiências:

- Acessos seqüenciais em um grande número de páginas de memória distintas;
- Acessos dentro de grandes *loops* (muitas páginas referenciadas por iteração);
- Freqüência irregular de acessos a uma mesma página.

Esta terceira situação não apresenta a mesma inércia comportamental quanto ao padrão de acessos que as duas primeiras. A tentativa de prever variações na freqüência de acessos às páginas

exige estruturas adicionais que armazenem informações históricas relevantes no decorrer da execução. A alta complexidade de implementação inerente a essa tentativa *(overhead)* pode torná-la desinteressante. Por outro lado, as duas primeiras situações – acessos seqüenciais e acessos dentro de grandes *loops* – podem ser facilmente detectadas em um processo, pois possuem uma característica extremamente bem definida: muitas faltas de página em um curto intervalo de tempo. Outra característica fundamental é que as páginas carregadas deixam de ser referenciadas rapidamente, sem que haja uma reutilização significativa enquanto permanecem residentes na memória.

O conceito de adaptatividade em algoritmos de substituição de páginas oferece benefícios importantes em potencial, como a correção de deficiências presentes nas políticas tradicionais – ou triviais – e a possibilidade de otimização nos critérios de substituição adotados. Após um amplo estudo nas propriedades do modelo LRU, chegamos à proposta de um novo algoritmo adaptativo: o LRU-WAR (*LRU with Working Area Restriction*), cujo objetivo é minimizar as falhas detectadas no algoritmo LRU sem sobrecarregar o sistema de gerenciamento de memória.

A Seção 2 deste artigo relata alguns dos principais algoritmos adaptativos para substituição de páginas, recentemente publicados na literatura, enquanto a Seção 3 detalha o algoritmo LRU-WAR. Resultados de desempenho obtidos em simulações realizadas com os algoritmos LRU e LRU-WAR são exibidos e analisados na Seção 4. Finalmente, a conclusão do trabalho é exposta na Seção 5.

2. Trabalhos Relacionados

Várias propostas de algoritmos adaptativos de substituição de páginas foram apresentadas à comunidade científica nos últimos anos, a maioria tendo como ponto de partida o tradicional e relativamente eficiente algoritmo LRU. Nas condições em que ele apresenta bom desempenho, sua atuação original é mantida; nas demais condições, uma alternativa de comportamento é desenvolvida.

O algoritmo SEQ (GLASS; CAO, 1997) pode ser considerado uma versão adaptativa do LRU que procura corrigir a perda de desempenho ocasionada pela presença de acessos linearmente seqüenciais. Quando identifica um ou mais conjuntos de referências a endereços de memória seqüencialmente adjacentes, o algoritmo implementa um critério de substituição pseudo-MRU, mantendo nas demais situações o critério LRU original.

Por sua vez, o algoritmo EELRU – *Early Eviction LRU* (SMARAGDAKIS; KAPLAN; WILSON, 1999), foi proposto como uma outra tentativa de mesclar o LRU e o MRU, baseado apenas nas posições da fila LRU em que as referências à memória se concentram. Esta fila nada mais é do que a própria representação da memória principal no modelo LRU, ordenada de forma decrescente sob o critério da recência de acesso às páginas. Analisando a reutilização de páginas residentes ou não, o algoritmo EELRU detecta potencialmente qualquer tipo de padrão seqüencial de acessos.

Outro algoritmo importante é o LIRS – *Low Inter-reference Recency Set* (JIANG; ZHANG, 2002). Seu objetivo é minimizar as deficiências apresentadas pelo LRU utilizando um critério adicional interessante: a chamada IRR (*Inter-Reference Recency*), que representa o número de páginas referenciadas entre os dois últimos acessos consecutivos a uma mesma página. O algoritmo pressupõe uma inércia comportamental e, de acordo com as IRRs coletadas, substitui a página que provavelmente levará mais tempo para ser novamente acessada. Isto significa que o LIRS não substitui necessariamente a página referenciada há mais tempo, mas ele utiliza o histórico recente desta informação para prever quais páginas têm maiores probabilidades de acesso em um futuro breve.

Existem ainda variações adaptativas do algoritmo 2Q (JOHNSON; SASHA, 1994), também originado a partir do modelo LRU, que tentam explorar eficientemente a relação conjunta de freqüência e recência dos acessos às páginas residentes. Os algoritmos ARC – *Adaptive Replacement Cache* (MEGIDDO; MODHA, 2003) e CAR – *Clock with Adaptive Replacement* (BANSAL; MODHA, 2004) seguem esta linha estratégica.

Três outros algoritmos procuram identificar certas características de execução nos processos e se adaptar dinamicamente a elas, alternando o critério de substituição de acordo com o padrão de

acessos vigente. São eles: DEAR – *DEtection-based Adaptive Replacement* (CHOI et al., 1999), AFC – *Application/File-level Characterization* (CHOI et al., 2000) e UBM – *Unified Buffer Management* (KIM et al., 2000).

As propostas aqui relatadas, entre outras, agregam benefícios conceituais importantes aos algoritmos tradicionais nos quais se baseiam, mas também possuem uma complexidade maior de implementação, muitas vezes exigindo estruturas adicionais que armazenem informações de páginas não residentes. Diferentemente, o algoritmo adaptativo LRU-WAR se destaca por detectar e tratar de maneira eficaz períodos com predominância de acessos seqüenciais (a principal falha da política LRU, base de seu desenvolvimento), mantendo praticamente a mesma complexidade estrutural e de implementação que o algoritmo LRU original.

3. O Algoritmo LRU-WAR

As políticas de substituição de páginas mais eficientes tentam explorar, cada uma ao seu modo, as características de localidade de referências inerentes aos programas, sobretudo localidade temporal. O algoritmo LRU, por exemplo, procura substituir páginas que não apresentem a tendência de acessos contínuos no momento de execução observado. No entanto, como foi ponderado na Seção 1, este algoritmo possui algumas deficiências, principalmente quando o programa em execução exibe padrões de acesso basicamente seqüenciais – e, conseqüentemente, baixa localidade temporal em muitas páginas de memória –, situação que acontece quando uma estrutura de dados (por exemplo vetor, matriz, lista ligada, árvore) é percorrida, elemento a elemento, por um longo intervalo de acessos. Isto pode ocorrer por diversos motivos: iniciação ou atualização de elementos, busca seqüencial, processamento consecutivo – como multiplicação de matrizes –, entre outros.

O algoritmo LRU-WAR (*LRU with Working Area Restriction*, ou *LRU com Confinamento da Área de Trabalho*) representa uma nova proposta adaptativa voltada a solucionar o mau desempenho do algoritmo LRU na presença de acessos com tais características. Baseado no tamanho máximo do *working set* estimado e validado entre duas faltas de página consecutivas, e na variação deste tamanho entre as faltas seguintes, o algoritmo utiliza um mecanismo ainda inexplorado pelas técnicas atuais: a dimensão máxima do *working set* temporário como fator decisivo de adaptatividade.

3.1. Idéia Geral

Working set – conjunto de trabalho –, conforme definido por Denning (1968), pode ser descrito como o conjunto das páginas acessadas pelo programa em um determinado intervalo de tempo. Subentende-se aqui, como *working set* temporário, o conjunto das páginas acessadas entre as duas últimas faltas de página ocorridas, qualquer que seja o instante do processamento. Em outras palavras: é o conjunto das páginas recentemente utilizadas pelo programa. Um novo conceito pode então ser proposto: definimos área de trabalho *(working area)* como sendo a região de recência onde o *working set* temporário está confinado, isto é, a porção inicial da fila LRU em que todos os acessos realizados entre as duas últimas faltas de página se concentraram.

O algoritmo LRU-WAR inova ao monitorar os acessos à memória e verificar, entre duas faltas consecutivas, a página referenciada com menor recência em relação ao seu último acesso; ou seja, ele identifica a posição mais alta da fila LRU que recebeu acessos nesse período. Tal posição, representada por W, limita a área de trabalho. Logo, a área de trabalho consiste no trecho da fila LRU compreendido entre a última página referenciada – página MRU, início da fila – e a página que ocupa a W-ésima posição da fila. Uma conclusão importante é que o *working set* temporário contém sempre, no máximo, as páginas que ocupam esta área.

A Figura 1 exemplifica um caso hipotético de execução, no qual W é identificado, assim como a localização da área de trabalho. É essencial salientar que as células da fila LRU preenchidas com a cor cinza indicam as posições que receberam acessos desde a última falta de página até o instante vigente. Isto não quer dizer que as páginas referenciadas continuam nestas mesmas posições. Tampouco quer dizer que as páginas que atualmente as ocupam foram acessadas no período. De

acordo com a política LRU, toda página referenciada é deslocada imediatamente para o início da fila. Os destaques em cinza especificam apenas um histórico das posições que tais páginas – não importa quais são – ocupavam no instante do seu respectivo acesso, antes da conseqüente reordenação na fila LRU e dentro do período de tempo considerado.

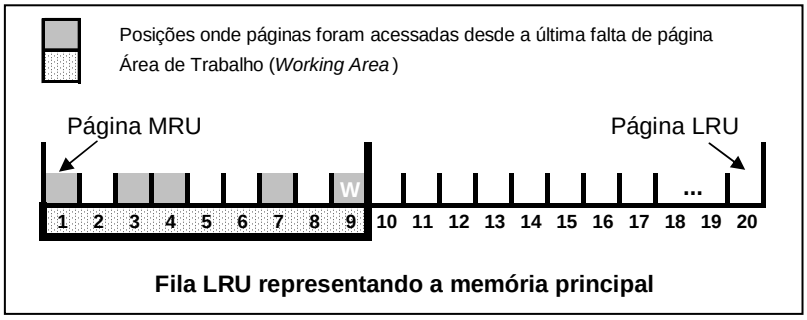

Figura 1. Exemplo de área de trabalho, identificando-se o limite W entre duas faltas de página consecutivas

O algoritmo LRU-WAR trabalha basicamente com dois critérios de substituição: o critério LRU, no qual a página acessada há mais tempo é retirada da memória, e o critério MRU-n, no qual a n-ésima página mais recentemente acessada é descartada. O critério de substituição padrão é o LRU, que só deixa de ser utilizado quando uma tendência de acessos seqüenciais se fortalece. O algoritmo detecta indícios de padrão seqüencial de acessos muito rapidamente, contudo mantém o critério LRU e aguarda durante um certo período de carência até que a tendência se intensifique. O critério de substituição então passa a ser o MRU-n nesta situação. A página substituída por tal critério é sempre aquela que ocupa a primeira posição da fila LRU fora da área de trabalho. Assim, n=W+1; o que permite especificar a versão MRU-n utilizada como sendo MRU-(W+1).

A idéia é substituir uma página que, por tendência, esteja se tornando inativa; isto é, uma página que deixou de pertencer ao *working set* temporário do processo há pouco tempo. Com este ponto de substituição precoce, um grande número de páginas permanece carregado na memória – todas aquelas que ocupam uma posição na fila LRU superior a W+1 – até que a tendência de acessos seqüenciais deixe de existir e o critério de substituição LRU volte a ser empregado.

3.2. Detalhes Operacionais

Para identificar tendências seqüenciais durante a execução de um programa, o algoritmo LRU-WAR verifica, a cada falta de página, se o tamanho da área de trabalho é pequeno o suficiente para sugerir que as páginas residentes estão sendo subutilizadas. Se for, assume a tendência seqüencial e monitora a variação de tamanho desta área durante as faltas seguintes. Permanecendo a área de trabalho pequena o suficiente por um certo número de faltas de página, entra em vigor o modo de operação seqüencial do algoritmo. Uma área de trabalho é considerada pequena quando está contida na região seqüencial, a qual corresponde às L (um dos dois parâmetro do algoritmo) primeiras posições da fila LRU. Consideramos o valor L=MIN[50,M/2] bastante razoável e adequado ao bom funcionamento do LRU-WAR, sendo M o tamanho de memória disponível. Trata-se de um valor pequeno, mas não tão pequeno ao ponto de invalidar a contribuição do algoritmo. Este foi o valor adotado em nossos experimentos.

A região LRU complementa a região seqüencial e consiste, obviamente, na porção final da fila LRU. Uma subregião também é identificada dentro da região seqüencial: a chamada região protegida, compreendendo as C primeiras páginas da fila; C (carência mínima) é o outro parâmetro do algoritmo, cujo valor original é C=5. Esta pequena região engloba algumas posições de recência consideradas baixas demais para conter a área de trabalho, o que permite dizer que a menor área de trabalho aceita pelo LRU-WAR é C+1. Logo, se $W \leq C$, o algoritmo atualiza o seu valor para W=C+1. A Figura 2 ilustra as divisões lógicas na fila LRU que orientam o algoritmo.

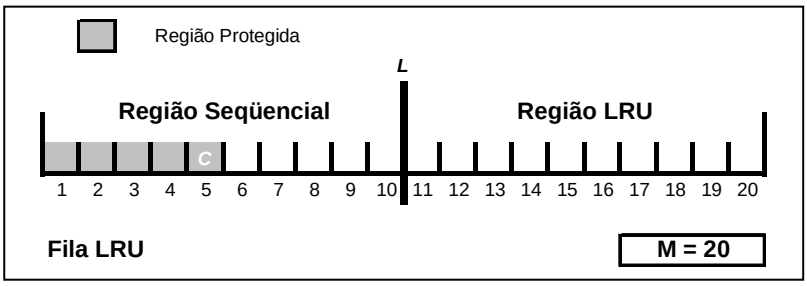

Figura 2. Divisão lógica da fila LRU utilizada pelo algoritmo LRU-WAR

Uma *tendência LRU* é assumida quando, entre duas faltas de página consecutivas, o tamanho da área de trabalho ultrapassa o tamanho da região seqüencial. Nesta situação, o algoritmo zera a posição W – que representa o tamanho da área de trabalho – no momento da falta. Se no próximo instante em que uma falta de página ocorrer, a área de trabalho for maior que a região seqüencial novamente, a tendência continua sendo LRU e a posição W volta a ser zerada. Caso contrário, uma *tendência seqüencial* é detectada – a ser confirmada pela imutabilidade no tamanho da área de trabalho durante as faltas subseqüentes.

O algoritmo LRU-WAR entra em *operação seqüencial* somente após:

- A ocorrência de um número de faltas de página proporcional ao tamanho da área de trabalho – e, portanto, equivalente a W – desde o instante em que a tendência seqüencial começou;
- A ocorrência de um número adicional de faltas de página, após a primeira condição ter sido satisfeita, referente a um tempo de carência complementar, utilizado como meio de detectar e prevenir erros de decisão do algoritmo.

Isto, é claro, se a área de trabalho permanecer na região seqüencial durante todo o período considerado, configurando uma baixa reutilização de páginas. Por exemplo, se a área de trabalho tem tamanho 10 em um determinado momento de execução (W=10), o critério MRU-n só é utilizado caso a área tenha se mantido menor ou igual a este tamanho durante as últimas 10 faltas de página, incluindo a atual, além do tempo de carência.

Fora de tendência LRU, ou seja, estando a área de trabalho contida na região seqüencial, a posição W somente aumenta, nunca diminui. Enfim: em tendência ou operação seqüencial, a área de trabalho não é sensível à diminuição de tamanho da região de recência em que o *working set* temporário está confinado. Ela só é sensível a um aumento neste tamanho. Isto acontece porque o objetivo de W é limitar a região de recência em que todos os acessos estiveram concentrados desde que a tendência seqüencial foi detectada. A Tabela 1 destaca os três estados de execução previstos pelo algoritmo e reitera os critérios de substituição empregados em cada caso.

Tabela 1. Estados de execução definidos pelo algoritmo LRU-WAR

ALGORITMO LRU-WAR			
Estado de execução	**Tamanho da Área de Trabalho**	**Critério de Substituição**	**Ponto de Substituição (Posição na Fila LRU)**
Tendência Original (LRU)	Maior que L	LRU	M
Tendência Seqüencial	Menor ou igual a L	LRU	M
Operação Seqüencial	Menor ou igual a L e estável	MRU-n	W+1

3.3. Descrição Prática do Algoritmo

O algoritmo LRU-WAR pode ser escrito na forma do pseudo-código a seguir. O tempo de carência TC consiste em um número de faltas de página adicional que o algoritmo respeita antes de operar em modo seqüencial. Apesar de a tendência seqüencial já ser bastante forte em dado momento, suficiente para o algoritmo começar a utilizar o critério de substituição MRU-n, o tempo de carência poupa algumas páginas que seriam substituídas de imediato por este critério, mantendo-as na memória.

```
1.  Se a página acessada está carregada na memória:
2.      P = posição da página na fila LRU;
3.      Se P > W:
4.          Se N > 0:                   /* Se está em operação seqüencial */
5.              INÉRCIA = 0;    /* Encerra a operação seqüencial */
6.              Se P > W+1 e P ≤ W+TC+1 e (N ≤ M-P ou N < 50):
7.                  TC = TC + N;    /* Detecta erro e aumenta TC */
8.              N = 0;
9.          W = P;    /* Aumenta área de trabalho */
10.     Reordena a página na primeira posição da fila.
11. Senão (a página acessada não está carregada na memória):
12.     Se a memória está cheia:
13.         Se W ≤ L:
14.             INÉRCIA = INÉRCIA + 1;
15.             Se W ≤ C:
16.                 W = C + 1;
17.             Se INÉRCIA ≥ W+TC:    /* Operação seqüencial */
18.                 Se N < M ou N < 50:
19.                     N = N + 1;
20.                 Se TC > C:
21.                     TC = TC – 1;
22.                 Remove a página na posição W+1 da fila;
23.             Senão (INÉRCIA < W+TC):    /* Tendência seqüencial */
24.                 Remove a página na posição M da fila;
25.         Senão (W > L):    /* Tendência LRU */
26.             INÉRCIA = 0;
27.             W = 0;
28.             N = 0;
29.             Remove a página na posição M da fila;
30.     Carrega a página acessada no início da fila.
```

O trecho compreendido entre as linhas de código 2 e 10 é executado quando uma página residente é acessada (*hit*). Se a página ocupa uma posição P na fila LRU maior que a área de trabalho, o tamanho da área é aumentado, passando a englobar esta posição: W = P. Em operação seqüencial, a contagem de faltas de página é reiniciada, encerrando tal fase. O modo de operação volta a ser, então, tendência LRU ou tendência seqüencial, dependendo do tamanho da nova área de trabalho. Se um erro de decisão é detectado, ou seja, uma das páginas preservadas da substituição seqüencial pelo tempo de carência foi a responsável pelo aumento de W, e a sua reutilização foi considerada muito rápida, incrementa-se o tempo de carência TC de acordo com o número de substituições realizadas em operação seqüencial.

O trecho de código logo abaixo, entre as linhas 13 e 29, descreve a atuação do algoritmo quando uma substituição de página precisa ser realizada. Em tendência original (LRU), a página acessada há mais tempo – que ocupa a posição M, última da fila – é substituída e a área de trabalho é zerada. Em tendência seqüencial, o critério de substituição LRU também é empregado e a área de trabalho apenas pode aumentar de tamanho. Finalmente, se o algoritmo está operando em modo seqüencial, a página que ocupa a posição W+1 da fila é substituída e a área de trabalho precisa permanecer estável para que a operação não seja interrompida. Ressalte-se que o tamanho da área de trabalho deve sempre ultrapassar o tamanho da região protegida, condição garantida pelas linhas 15 e 16. O número de faltas de página ocorridas desde o começo de uma tendência seqüencial é calculado através do contador INÉRCIA. Por sua vez, o contador N é o responsável pelo armazenamento do número de faltas de página computadas em operação seqüencial.

Uma versão *online* (factível) do algoritmo LRU-WAR também foi criada, com a mesma simplicidade algorítmica da versão *offline* (completa, mas teórica). Assim como ocorre nas implementações reais do LRU, esta versão *online* do LRU-WAR é apenas uma aproximação da versão *offline*. A grande diferença está na fila LRU, que não pode ser atualizada em todos os acessos, apenas durante os momentos de falta de página.

4. Avaliação de Desempenho

Um arquivo de *traces* consiste basicamente na listagem cronológica dos diversos acessos à memória realizados por um processo. Em outras palavras, descreve passo a passo o comportamento de um dado programa em termos de utilização da memória. Estes arquivos são obtidos através de um gerador de *traces*, que executa o programa alvo e coleta os acessos à memória realizados ao longo de seu processamento (UHLIG; MUDGE, 1997).

A partir dos dados coletados e armazenados nos arquivos de *trace*, diversas experiências podem ser efetuadas para se avaliar o desempenho de algoritmos de substituição de páginas, no contexto de todos os possíveis tamanhos de memória que se queira considerar. Utilizando esta técnica de avaliação, um conjunto de simulações foi organizado para averiguar o desempenho do algoritmo LRU-WAR e compará-lo com a política de substituição de páginas LRU original. O critério de desempenho utilizado é o número de faltas de página geradas na execução de um processo.

4.1. Simulações Realizadas

Os experimentos aqui relatados tiveram como carga de simulação os sete arquivos de *trace* que compõem o pacote VMTrace, originalmente aplicados por Smaragdakis; Kaplan; Wilson (1999) nas simulações do algoritmo EELRU, sendo também por eles disponibilizados. O nome deste pacote designa a ferramenta de geração de *traces* utilizada em sua captação, desenvolvida pelos próprios autores. A seguir são resumidamente descritos os programas representados por estes sete arquivos:

- *Espresso:* Simulador de circuito;
- *GCC:* Compilador C/C++ do projeto GNU, versão 2.7.2;
- *Gnuplot:* Gerador de gráficos do projeto GNU;
- *Grobner:* Programa matemático que trabalha com Bases de Gröbner;
- *GS:* GhostScript 3.33, interpretador PostScript;
- *Lindsay:* Simulador de hipercubo;
- *P2C:* Tradutor de programas em Pascal para C.

O programa Gnuplot é o único que apresenta predominância de acessos seqüenciais à memória, intercalados com referências a poucas páginas que exibem alta localidade temporal. Em contrapartida, os demais arquivos VMTrace se caracterizam principalmente pelo aspecto da localidade temporal, sem a presença de outros padrões regulares facilmente identificáveis (CASSETTARI; MIDORIKAWA, 2004).

A Tabela 2 descreve as baterias de simulação às quais os programas foram submetidos. A execução de cada programa é simulada com diferentes tamanhos de memória disponível. Tamanhos muito pequenos (abaixo de 10 páginas) e muito grandes (acima do número de páginas que o programa referencia em todo o seu processamento) foram desprezados, pois não permitem uma comparação efetiva entre os algoritmos – já que os resultados de desempenho são equivalentes nestas condições.

Tabela 2. Programas e contextos de memória considerados nas simulações

	Simulações Realizadas				
	Arquivo de *Traces*	**Páginas Distintas**	**Tamanhos de Memória Simulados**	**Interv. entre Tamanhos**	**Número de Simulações**
VMTrace	Espresso	77	10, 11, 12, ..., 73, 74, 75	1	66
	GCC	458	10, 15, 20, ..., 445, 450, 455	5	90
	Gnuplot	7718	100, 200, 300, ..., 7500, 7600, 7700	100	77
	Grobner	67	10, 11, 12, ..., 63, 64, 65	1	56
	GS	558	10, 15, 20, ..., 545, 550, 555	5	110
	Lindsay	521	10, 15, 20, ..., 510, 515, 520	5	103
	P2C	132	10, 15, 20, ..., 120, 125, 130	5	25
				Total de Simulações	527

A segunda coluna da Tabela 2 informa o número de páginas distintas referenciadas pelos programas; a terceira coluna relata os tamanhos de memória utilizados em cada simulação e a quarta coluna indica o intervalo entre estes tamanhos. A última coluna, por fim, contabiliza o número de

simulações efetuadas. Cada simulação avalia o desempenho de um algoritmo considerando apenas um único tamanho de memória. O conjunto das simulações associadas a um programa indica o desempenho médio que o algoritmo alcança com o mesmo.

4.2. Resultados Obtidos

A Tabela 3 demonstra percentuais de desempenho referentes à diferença entre o número de faltas de página gerado pelo algoritmo LRU e o número gerado pelo LRU-WAR. Valores percentuais positivos indicam que o novo algoritmo apresenta um desempenho pior que o do algoritmo LRU, pois provoca um número maior de faltas de página. Valores negativos, obviamente, indicam o contrário.

Tabela 3. Desempenho do algoritmo LRU-WAR em relação ao algoritmo LRU

ALGORITMO LRU-WAR: Diferença percentual no número de faltas de página em relação ao algoritmo LRU					
Arquivo de *Traces*	**Melhor Caso**		**Pior Caso**		**Média**
	Diferença % (Mem.)		**Diferença % (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%
Pacote VMTrace	**-66,22%**		**5,09%**		**-6,58%**

A coluna “melhor caso” reflete a situação de execução em que o algoritmo LRU-WAR obtém o melhor desempenho percentual comparado com o algoritmo LRU. A situação inversa é ilustrada pela coluna “pior caso”: quando o LRU obtém o melhor desempenho. A média aritmética das diferenças percentuais coletadas entre os dois algoritmos, levando-se em conta todas as simulações descritas na Tabela 2, é calculada na terceira coluna. Um exemplo: 56 simulações são realizadas para avaliar o desempenho médio de cada algoritmo na execução do programa Grobner e, portanto, 56 diferenças percentuais são calculadas entre números de faltas de página gerados pelo algoritmo LRU e números gerados pelo algoritmo LRU-WAR. A menor destas diferenças (maior redução no número de faltas de página) representa o melhor caso do LRU-WAR em relação ao LRU; a maior diferença representa o pior caso; e a média aritmética entre as 56 diferenças indica uma tendência geral.

Assim, podemos interpretar da seguinte forma os dados relativos ao programa Grobner: o LRU-WAR, em média, atinge um desempenho 3,78% melhor que o do algoritmo LRU. Porém, ele pode provocar um número de faltas de página até 1,02% maior, o que acontece especificamente com uma memória de tamanho 13. Em uma outra situação de execução (memória disponível de 28 páginas), entretanto, ele exibe um resultado 11,75% melhor. Logo, o LRU-WAR pode ser melhor ou pior do que o LRU no processamento de tal programa, dependendo do tamanho de memória disponível; mas a média sugere que ele tende a apresentar melhores resultados. Isto é confirmado pela Figura 3, cujos gráficos informam a variação percentual de faltas de página geradas pelo algoritmo LRU-WAR em relação às faltas geradas pelo algoritmo LRU nas simulações efetuadas com os *traces* Grobner e Gnuplot. O programa Grobner representa uma situação típica: o LRU-WAR é melhor do que o LRU, mas não muito melhor. Por outro lado, o programa Gnuplot exibe um padrão efetivo de acessos seqüenciais à memória que predomina em toda a sua execução; neste caso, o LRU apresenta um desempenho muito ruim, enquanto o LRU-WAR mostra excelentes resultados.

A conclusão a que se chega, a partir dos resultados práticos obtidos, é que o algoritmo LRU-WAR atende ao seu propósito: é substancialmente melhor do que o algoritmo LRU quando padrões de acesso seqüenciais vigoram e não é muito pior em nenhuma situação analisada. No seu melhor caso, o LRU-WAR alcançou um desempenho 66,22% superior ao do LRU. Não obstante, seu pior desempenho em relação ao mesmo gerou uma diferença percentual máxima de 5,09% no número de faltas de página, em um universo de 527 simulações realizadas.

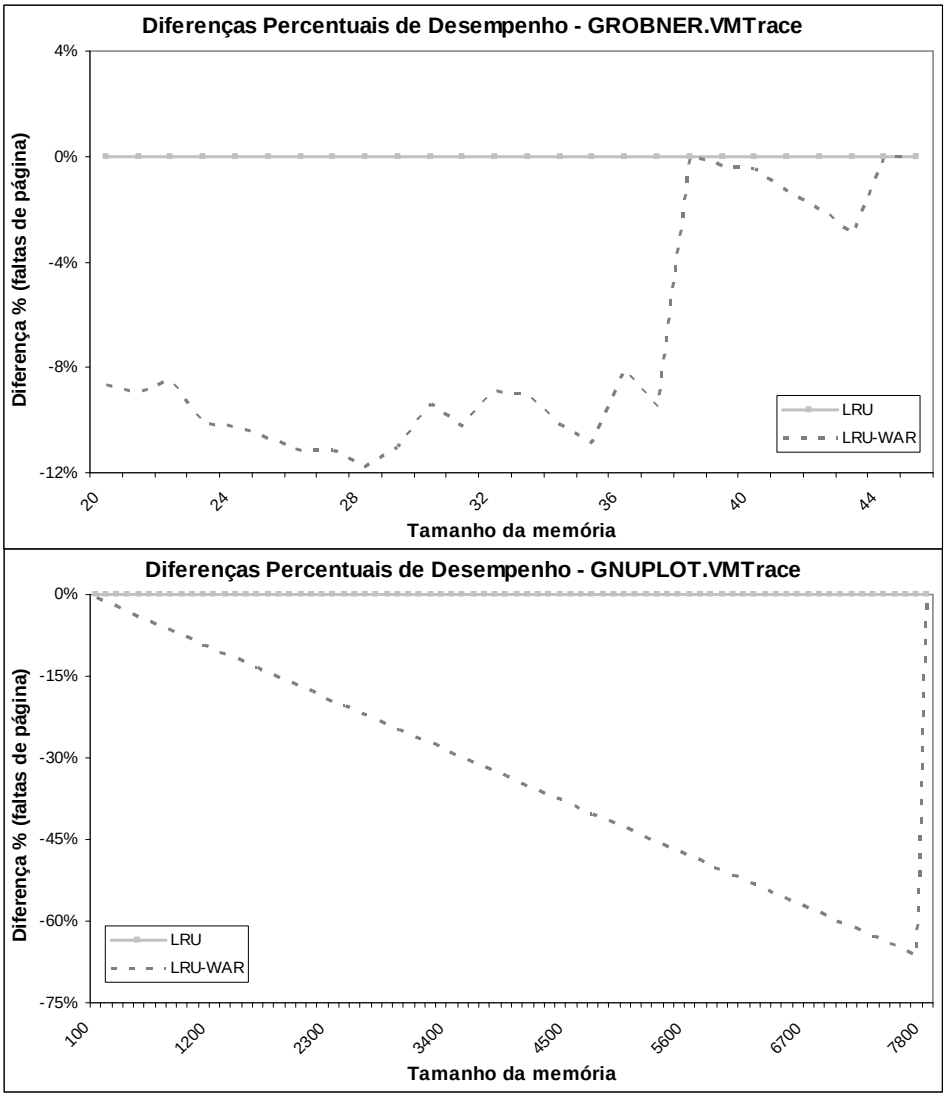

Figura 3. Variação de desempenho do algoritmo LRU-WAR em relação ao LRU

5. Conclusão

Após uma importante análise acerca das propriedades do modelo LRU e de algumas características inerentes a programas dos mais variados, verificamos que poderíamos reconhecer padrões de acesso seqüenciais, dinamicamente, somente identificando as posições na fila LRU ocupadas pelas páginas referenciadas ao longo do tempo por um processo. A nova técnica poderia, assim, contribuir para melhorar o desempenho do algoritmo LRU original na presença de tais padrões, visto que acessos seqüenciais tendem a romper a eficiência normalmente observada em sua atuação.

Vários aspectos operacionais foram então definidos, dando origem ao algoritmo adaptativo de substituição de páginas LRU-WAR (LRU com Confinamento da Área de Trabalho), uma proposta simples e inédita sintetizada neste artigo. Simulações demonstraram que o algoritmo LRU-WAR alcança desempenhos muito bons quando trabalha com programas que apresentam padrões de acesso basicamente seqüenciais. Ao mesmo tempo, o algoritmo se mostrou confiável, uma vez que os seus resultados não foram significativamente piores que os do LRU tradicional em nenhum caso analisado. Um estudo de avaliação de desempenho muito mais completo e detalhado está disponível em Cassettari (2004), que também justifica cada decisão de projeto e descreve com maior profundidade as motivações, parâmetros, estruturas e considerações operacionais que fundamentam e orientam o algoritmo LRU-WAR.

O objetivo final de nossos estudos é propor, a partir da versão *online* do algoritmo, um sistema de gerenciamento de memória completo e adaptativo. Este sistema seria responsável por um procedimento de particionamento dinâmico da memória entre os processos que viabilizaria o funcionamento eficiente da política LRU-WAR em ambientes com multiprogramação. O desenvolvimento e a implementação prática de tal estratégia em um sistema operacional de código aberto, provavelmente o Linux, é a seqüência natural deste trabalho.

Lista de Referências

BANSAL, S.; MODHA, D.S. CAR: Clock with adaptive replacement. In: CONFERENCE ON FILE AND STORAGE TECHNOLOGIES, 3., San Francisco, 2004. *FAST' 04:* Proceedings. San Francisco: USENIX, 2004. p.187-200.

CASSETTARI, H.H. *Análise da localidade de programas e desenvolvimento de algoritmos adaptativos para substituição de páginas*. 2004. 118p. Dissertação (Mestrado) – Escola Politécnica, Universidade de São Paulo. São Paulo, 2004.

CASSETTARI, H.H.; MIDORIKAWA, E.T. Caracterização de cargas de trabalho em estudos sobre gerência de memória virtual. In: WORKSHOP EM DESEMPENHO DE SISTEMAS COMPUTACIONAIS E DE COMUNICAÇÃO, 3., Salvador, 2004. *WPerformance' 04:* Anais. Salvador: SBC, 2004.

CHOI, J. et al. An implementation study of a detection-based adaptive block replacement scheme. In: ANNUAL TECHNICAL CONFERENCE, 4., Monterey, 1999. *USENIX' 99:* Proceedings. Monterey: USENIX, 1999. p.239-252.

____. Towards application/file-level characterization of block references: a case for fine-grained buffer management. In: INTERNATIONAL CONFERENCE ON MEASUREMENT AND MODELING OF COMPUTER SYSTEMS, 25., Santa Clara, 2000. *SIGMETRICS' 00:* Proceedings. Santa Clara: ACM, 2000. p.286-295.

DENNING, P.J. The working set model for program behavior. Communications of the ACM, v.11, n.5, p.323-333, 1968.

GLASS, G.; CAO, P. Adaptive page replacement based on memory reference behavior. In: INTERNATIONAL CONFERENCE ON MEASUREMENT AND MODELING OF COMPUTER SYSTEMS, 22., Seattle, 1997. *SIGMETRICS' 97:* Proceedings. Seattle: ACM, 1997. p.115-126.

JIANG, S.; ZHANG, X. LIRS: An efficient low inter-reference recency set replacement policy to improve buffer cache performance. In: INTERNATIONAL CONFERENCE ON MEASUREMENT AND MODELING OF COMPUTER SYSTEMS, 27., Marina Del Rey, 2002. *SIGMETRICS' 02:* Proceedings. Marina Del Rey: ACM, 2002. p.31-42.

JOHNSON, T.; SHASHA, D. 2Q: a low overhead high performance buffer management replacement algorithm. In: INTERNATIONAL CONFERENCE ON VERY LARGE DATA BASES, 20., Santiago, 1994. *VLDB' 94:* Proceedings. Santiago: Morgan Kaufmann, 1994. p.439-450.

KIM, J.M. et al. A low-overhead high-performance unified buffer management scheme that exploits sequential and looping references. In: SYMPOSIUM ON OPERATING SYSTEM DESIGN AND IMPLEMENTATION, 4., San Diego, 2000. *OSDI' 00:* Proceedings. San Diego: USENIX, 2000. p.119-134.

MEGIDDO, N.; MODHA, D.S. ARC: a self-tuning, low overhead replacement cache. In: CONFERENCE ON FILE AND STORAGE TECHNOLOGIES, 2., San Francisco, 2003. *FAST' 03:* Proceedings. San Francisco: USENIX, 2003. p.115-130.

MIDORIKAWA, E.T. *Uma nova estratégia para a gerência de memória para sistemas de computação de alto desempenho*. 1997. 193p. Tese (Doutorado) – Escola Politécnica, Universidade de São Paulo. São Paulo, 1997.

SMARAGDAKIS, Y.; KAPLAN, S.; WILSON, P. EELRU: simple and effective adaptive page replacement. In: INTERNATIONAL CONFERENCE ON MEASUREMENT AND MODELING OF COMPUTER SYSTEMS, 24., Atlanta, 1999. *SIGMETRICS' 99:* Proceedings. Atlanta: ACM, 1999. p.122-133.

UHLIG, R.A.; MUDGE, T.N. Trace-driven memory simulation: a survey. ACM Computing Surveys, v.29, n.2, p.128-170, 1997.