Ajustando o LRU-WAR para uma Política de Gerência de Memória Global

Ricardo L. Piantola, Edson T. Midorikawa

Departamento de Engenharia de Computação e Sistemas Digitais
Escola Politécnica da Universidade de São Paulo
05508-900 – São Paulo – SP – Brasil

piantola@uol.com.br, edson.midorikawa@poli.usp.br

Abstract. *Even with recent advances in Computer Architecture, the memory wall problem has not been solved yet. In order to lessen the performance difference between processor and memory, it is essential to create new memory management strategies, which are stronger in performance. Operating Systems generally use global policies for memory management. The adaptive strategies have as principle to adapt their behaviour based on programs memory access patterns. Its use in multiprogrammed environment has not been properly examined in previous studies. This paper intends to present a strategy to adjust LRU-WAR in order to obtain a good performance in a global memory management system. The results show that the same strategy can be used in algorithms following the same adaptive principle that LRU-WAR does.*

Resumo. *Mesmo com os avanços na área de arquitetura de computadores, ainda não foi resolvida a questão sobre o desempenho das memórias em relação aos processadores. Para atenuar esta diferença é imprescindível criar novas estratégias de gerência de memória, que sejam mais robustas em desempenho. Os Sistemas Operacionais geralmente utilizam políticas globais para a gerência de memória. As estratégias adaptativas têm como princípio adaptar seu funcionamento com base no padrão de acessos à memória dos programas. Sua utilização em ambiente multiprogramado ainda não foi adequadamente estudada em trabalhos anteriores. Este artigo procura mostrar uma estratégia para ajustar o LRU-WAR com a finalidade de obter bom desempenho em um sistema de gerência de memória global. Os resultados mostram que a mesma estratégia pode ser adequada em algoritmos que seguem o mesmo princípio adaptativo que o LRU-WAR.*

1. Introdução

"*Quanto mais processamos dados, mais dados surgem para serem processados*", está é uma das versões da dita Lei de Parkinson. O aumento da carga computacional tem motivado a evolução dos computadores. No contexto da computação moderna a centralização de recursos se tornou evidente com o atual ressurgimento das tecnologias de virtualização, como o VMware ESX Server e o sistema Viridian da Microsoft, e com a recente difusão de processadores multi-cores. A necessidade de uma gerência eficiente de recursos aumenta consideravelmente, em particular a gerência eficiente da memória principal.

Várias equipes de pesquisadores têm desenvolvido trabalhos no âmbito de algoritmos para gerência de memória virtual em sistemas operacionais. Há propostas recentes na

literatura sobre algoritmos adaptativos de substituição de páginas, como SEQ [Glass and Cao 1997], EELRU [Smaragdakis et al 1999], LIRS [Jiang and Zhang 2002], ARC [Megiddo and Modha 2003], CAR [Bansal and Modha 2004] e LRU-WAR [Cassettari and Midorikawa 2004b]. Contudo estes algoritmos não foram desenvolvidos visando seu uso para ambientes multiprogramados com o controle das páginas de várias aplicações simultaneamente.

O objetivo principal deste artigo é apresentar uma forma de ajustar o algoritmo adaptativo de substituição de páginas LRU-WAR para uma política de sistema de gerência de memória global, através da utilização da técnica *profiling* e da adição de um parâmetro de controle, que levou à criação de um novo algoritmo chamado LRU-WARlock.

Este artigo está organizado da seguinte forma. A seção 2 discute o uso da adaptabilidade em políticas globais, descrevendo o desempenho dos algoritmos adaptativos "tradicionais" nesses sistemas de gerência de memória. A seção 3 descreve a proposta de adaptação do LRU-WAR. A avaliação de desempenho é realizada sobre o algoritmo LRU-WARlock é apresentada na seção 4. A seção 5 finaliza o artigo e traz as principais conclusões, indicando alguns trabalhos futuros.

2. O Uso da Adaptabilidade em Políticas Globais

Toda política de substituição de páginas em um sistema de paginação por demanda deve escolher uma vítima para dar lugar a uma nova página referenciada, quando não há mais espaço disponível na memória. Caso a página escolhida seja sempre uma página do próprio processo que sofreu a falta de página, a política utilizada pelo sistema de memória é dita local. Porém, é possível que o sistema de gerência de memória trabalhe de uma forma global. Um algoritmo de substituição global atua com todas as páginas da memória, e escolhe a vítima independente do processo que sofreu a falta de página.

Existem algumas vantagens em utilizar políticas globais em sistemas de gerência de memória. Alguns processos podem precisar de mais memória que outros, neste caso a política global pode dimensionar corretamente a quantidade de memória para cada processo. Os processos que estão bloqueados, esperando algum serviço, mantêm páginas na memória consumindo espaço que poderia ser utilizado por páginas de outros processos. Em um sistema com política global isto é minimizado.

Os algoritmos adaptativos de substituição tentam adaptar seu comportamento de forma dinâmica conforme o padrão de referências à memória. Alguns dos algoritmos adaptativos mais conhecidos na literatura são: SEQ, EELRU, LIRS, ARC, CAR e LRFU [Lee at al 2001]. Algoritmos adaptativos recentes utilizam técnicas de Inteligência Artificial para ajudá-los na adaptação, como por exemplo, o FPR [Sabeghil and Yaghmaee 2006].

A maioria dos algoritmos adaptativos "tradicionais" é inadequada para um sistema de memória com política global. Os projetos desses algoritmos visam analisar um padrão de acesso de uma única aplicação. Quando os acessos são contabilizados de forma global, ou seja, as referências à memória são coletadas de múltiplas aplicações, estes algoritmos se

comportam de maneira ineficiente. Muitos desses algoritmos, incluindo o LRU-WAR, investem maior foco de suas análises na recência dos acessos e não na freqüência. A análise da freqüência de acessos é um fator importante para uma política global e é apresentada mais detalhadamente na próxima seção.

3. Proposta de Adaptação do LRU-WAR

O principal objetivo da adaptação do algoritmo LRU-WAR é fazer com que ele consiga obter bom desempenho em ambientes com política de gerência de memória global, onde se torna mais difícil a detecção de padrões de acessos. Para conquistar esse objetivo foi criado o algoritmo LRU-WARlock, que complementa o LRU-WAR, sem modificar sua idéia original da exploração dos acessos seqüências.

O princípio do funcionamento de LRU-WARlock surgiu da idéia de separar os acessos diferentes do padrão de acesso seqüencial e tratá-los de outra maneira. A questão crucial foi descobrir o que mais atrapalhava a detecção de referências seqüenciais. A resposta estava no bom funcionamento geral do LRU. Em um ambiente multiprogramado com sistema de gerência de memória global paginado, a memória é composta pelas páginas de diferentes programas. Neste cenário, com vários *working sets* [Denning 1968] de diferentes programas presentes na memória, é possível encontrar vários padrões de acesso diferentes: por exemplo, muitas páginas com poucos acessos e algumas páginas com alta quantidade de referências. A solução neste cenário seria remover as páginas dos *working sets* da detecção do padrão seqüencial do LRU-WAR, ou seja, separar as páginas acessadas mais freqüentemente.

O mecanismo utilizado pelo LRU-WARlock é a reserva de parte da memória para as páginas com a maior freqüência de acessos. A outra parte da memória mantém o funcionamento original do LRU-WAR desconsiderando a parte reservada. Foi criado um parâmetro de controle do algoritmo chamado K. O parâmetro K está relacionado com a porcentagem de memória que é reservada para as páginas com alta freqüência de acesso, e pode ser controlado diretamente pelo sistema operacional.

Com relação à determinação das páginas de maior freqüência, foi escolhida a técnica de *profiling*. Essa técnica possibilita capturar informações em tempo de execução da aplicação, e utilizar esse conhecimento em futuras execuções. Alguns compiladores já utilizam essa técnica para prover otimização na geração do código executável. O compilador Intel C++ utiliza a técnica *Profile-Guided Optimization* (PGO) [Intel 2008], que localiza quais partes da aplicação são mais freqüentemente executadas, direcionando assim o foco da otimização.

3.1. Algoritmo LRU-WARlock

O algoritmo LRU-WARlock está dividido logicamente em duas partes: o LRU-WAR sem modificação e uma nova parte que utiliza a técnica de *profiling* como auxílio, mais o novo parâmetro de controle K.

A parte referente ao LRU-WAR continua monitorando os acessos à memória, entre duas faltas de páginas consecutivas, utilizando a dimensão máxima da área de trabalho como fator decisivo de adaptabilidade. Existem três estados possíveis de execução definidos: tendência LRU, tendência seqüencial e operação seqüencial. Na operação padrão, tendência LRU, o critério adotado para a substituição de páginas é o LRU, o mesmo se aplica à tendência seqüencial. Quando são detectados acessos seqüenciais, o algoritmo entra em operação seqüencial, sendo adotado para a substituição de páginas MRU-n (com o parâmetro n igual a W+1, onde W é tamanho da Área de Trabalho – Figura 1).

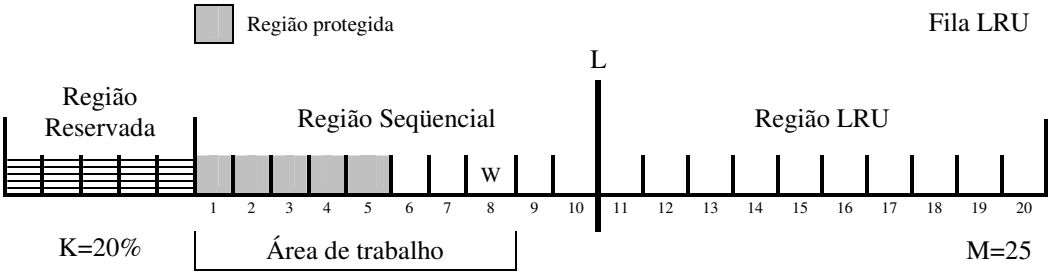


Figura 1. Divisão lógica do algoritmo LRU-WARlock.

A nova região da divisão lógica do algoritmo (Figura 1), a Região Reservada, é composta pelas páginas referenciadas com maior freqüência, e está fora de todo o processamento descrito na parte do LRU-WAR. O tamanho da Região Reservada é definido pelo novo parâmetro de controle K, em porcentagem do total de páginas.

A Figura 2 apresenta o pseudocódigo do algoritmo, onde são destacados os acréscimos ao código original do LRU-WAR.

```
1. Se a página referenciada está na memória:
2.     Se o bit "page locked" for igual a zero:
3.           Se está em Operação Seqüencial:
4.                 Termina a Operação Seqüencial
5.                 Aumenta o tempo de carência
6.           Aumenta Área de Trabalho
7.           Reordena a página na primeira posição da fila.
8. Senão, se não está na memória e não está nas páginas do profile:
9.     Se a memória está cheia (memória total – memória reservada):
10.          Se estiver em Operação Seqüencial
11.                Remove a página na posição (Área de Trabalho + 1) da fila;
12.          Senão, se estiver em Tendência Seqüencial:
13.                Se a Área de Trabalho for menor que L e excedeu a carência
14.                      Entrar em Operação Seqüencial
15.                Remove a página na posição LRU da fila;
16.          Senão (Está em Tendência LRU):
17.                Se Área de Trabalho for menor que L
18.                      Entrar em Tendência seqüencial
19.                Diminui Área de Trabalho e ajusta carência.
20.                Remove a página na posição LRU da fila;
21.    Carrega a página referenciada no início da fila.
22. Senão, se não está na memória e está no profile:
23.    Carrega a página referenciada
24.    Adiciona o bit "page locked" a página
```

Figura 2. Pseudocódigo do algoritmo LRU-WARlock.

As novas condições adicionadas ao código são referenciadas nas linhas 8 e 22: se a página acessada não está na memória e não está no *profile*, então executará o LRU-WAR ou se a página não está na memória e está no *profile*, então a página é alocada na Região Reservada. Quando a página é alocada na Região Reservada, seu bit "*page locked*" é ativado. Neste caso a página não será mais removida da memória até que o programa termine sua execução. No pseudocódigo essa tarefa está nas linhas 23 e 24.

Este tratamento diferenciado proporcionado pelo LRU-WARlock, classificando as referências à memória de acordo com o padrão de acesso, permite a melhor adaptação do LRU-WAR em ambientes multiprogramados. Outra vantagem parte do princípio em que se a página referenciada estiver na Região Reservada, não é necessário executar nenhum ajuste nos parâmetros, diferente do LRU-WAR que executa os passos de 3 a 7.

4. Análise do Desempenho

Nesta seção apresentamos a análise efetuada sobre o algoritmo adaptativo de substituição de páginas LRU-WARlock. Começamos com a descrição dos *traces* de aplicações e ferramentas utilizadas para a avaliação e, na seqüência, a análise dos resultados obtidos através dos testes.

4.1. Caracterização dos *Traces* Utilizados

A avaliação do LRU-WARlock foi efetuada usando três diferentes *traces* de aplicações, são eles: multi1, multi2 e multi3 (Tabela 1). Os três *traces* [Kim et al 2000] foram selecionados pelo fato de conterem acessos simultâneos de aplicações, simulando um ambiente multiprogramado. As ferramentas utilizadas nas simulações fazem parte do ambiente Elephantools [Cassettari and Midorikawa 2004a].

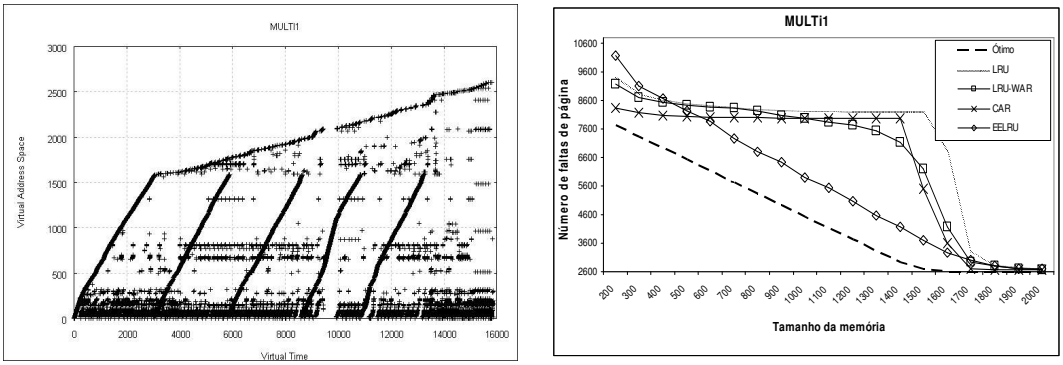
Passaremos a apresentar a composição dos *traces* utilizados nos testes, além de uma breve descrição dos padrões de acesso e comportamento com algoritmos de substituição da literatura.

Tabela 1. Descrição dos *traces* utilizados.

Trace	Descrição	Origem	Total de páginas
multi1	Execução simultânea das aplicações cscope e cpp.	LIRS	2606
multi2	Execução simultânea das aplicações cscope, cpp e postgres.	LIRS	5684
multi3	Execução simultânea das aplicações cpp, gnuplot, glimpse e postgres.	LIRS	7454

O *trace* **multi1** é composto pelas aplicações cscope e cpp. O cscope é uma ferramenta interativa de verificação de programa fonte escrita em linguagem C. Seu padrão de acesso à memória faz referências a *looping* com forte localidade temporal e outras referências de padrão diverso (Figura 3.a). O cpp é o pré-processador do GNU C, onde durante sua execução pode ser observado blocos de referências seqüenciais à memória em conjunto com outras referências. O *trace* multi1 intercala acessos dessas duas aplicações, uma com referências a

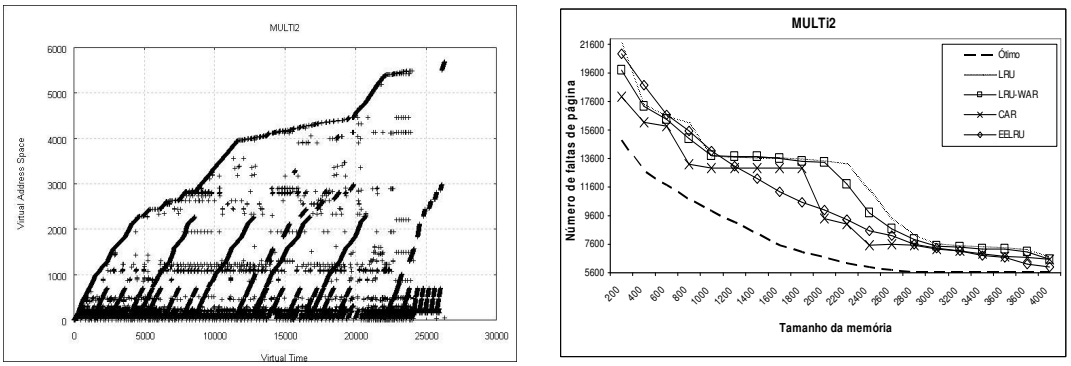
looping e a outra com acesso seqüencial. Este padrão de acesso prejudica muito o desempenho do algoritmo LRU [Jiang and Zhang 2002], isso acontece até que o espaço de memória disponível seja maior ou igual ao tamanho total das duas aplicações (Figura 3.b). Os programas individuais apresentam um padrão de acessos adequado ao LRU-WAR, porém como estão intercalados pela multiprogramação, diminuem consideravelmente o desempenho do algoritmo.



(a) Padrão de acessos à memória. (b) Desempenho de alguns algoritmos.

Figura 3. Características do *trace* multi1.

O *trace* **multi2** tem a mesma composição do *trace* multi1 com a adição do programa postgres. O postgres é um sistema de banco de dados relacional da Universidade da Califórnia. Apresenta um padrão de acessos seqüencial e *looping* com períodos não constantes. A Figura 4.a mostra o padrão de acessos do *trace* multi2. Contudo, ao adicionar o programa postgres, e comparar o LRU-WAR com o LRU (Figura 4.b) vemos que a melhoria no desempenho não consegue ultrapassar 13% com 2400 de tamanho da memória, contra 39% de melhoria com relação ao LRU no *trace* multi1 para memórias de tamanho 1600 (Figura 3.b). Esta é uma conseqüência visível, e o motivo está diretamente relacionado com o aumento do nível de multiprogramação, que deixa mais difícil a detecção de padrões seqüenciais pelo algoritmo LRU-WAR.

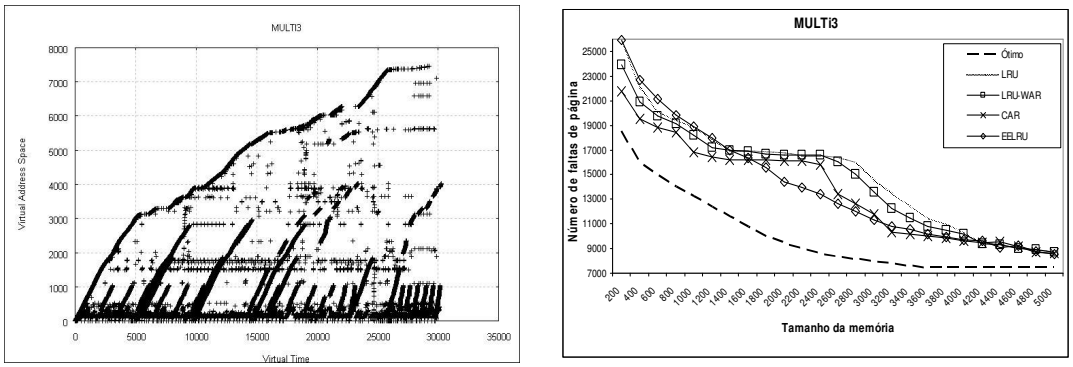


(a) Padrão de acessos à memória. (b) Desempenho de alguns algoritmos.

Figura 4. Características do *trace* multi2.

O terceiro *trace* escolhido, o **multi3**, tem a configuração um pouco diferente dos dois primeiros. É formado pelo cpp, prostgres, glimpse e gnuplot. O cpp está contido nos dois

primeiros *traces* e o prostgres no multi2. O glimpse é um utilitário usado na busca de informações em textos. Seu padrão de referência a memória é bem diverso. Já o gnuplot tem um padrão de acessos seqüencial bem definido. O gnuplot é um programa interativo de plotagem gráfica. É possível observar (Figura 5.b) que com o aumento da quantidade de programas contidos no multi3, a diferença de desempenho entre o algoritmo LRU e LRU-WAR diminui. Neste caso a diferença chega apenas a 8% com tamanho de memória de 3200 páginas. Este resultado é também devido ao aumento da multiprogramação, que cria dificuldades de detecção de padrões seqüenciais pelo algoritmo LRU-WAR.



(a) Padrão de acessos à memória. (b) Desempenho de alguns algoritmos.

Figura 5. Características do *trace* multi3.

4.2. Resultados Obtidos e Análises

Apresentamos nesta seção, cumprindo os objetivos definidos neste trabalho, as análises e os resultados do estudo do desempenho do algoritmo LRU-WARlock, a adaptação do LRU-WAR para sistemas com política de memória global.

Multi1

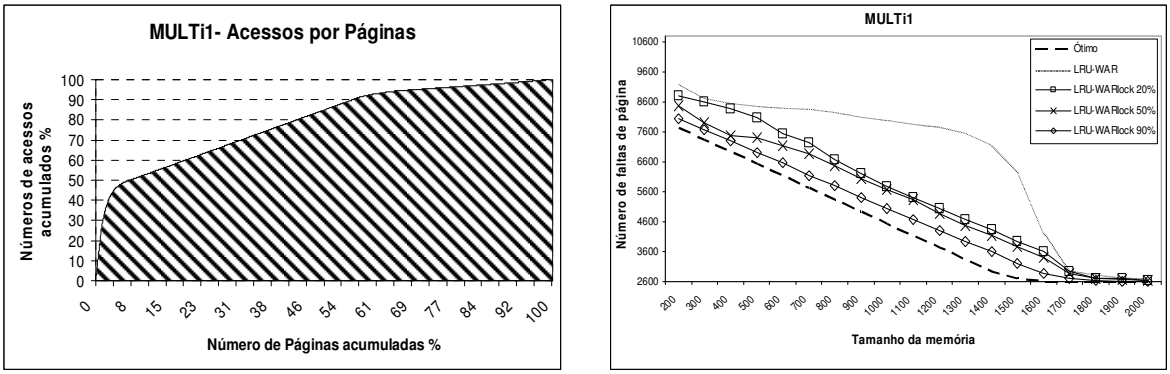
Em um programa com padrões de acessos seqüenciais, o algoritmo LRU-WAR atinge um excelente desempenho. Trabalhos anteriores demonstraram este comportamento em sistemas de gerência de memória local [Cassettari and Midorikawa 2004], e que também ainda é possível melhorá-lo ajustando seus parâmetros C e L dinamicamente [Midorikawa, Piantola and Cassettari 2007]. Como o *trace* multi1 apresenta acessos de duas aplicações intercaladas em um sistema de gerência de memória global, o LRU-WAR apresentou desempenho não muito bom, mesmo com os dois programas que compõem o multi1 tendo acessos seqüenciais (Figura 6.b).

Todos os testes de desempenho foram realizados variando o parâmetro K de 10 a 90%, mas apresentamos aqui somente os resultados para 20, 50 e 90%. O *trace* multi1 possui menos páginas e menos referências à memória em comparação com os *traces* multi2 e multi3. Isto pelo fato de ter somente dois programas inclusos no trace, porém isso não descaracteriza sua importância.

No multi1, a análise do perfil de freqüência de acessos às páginas mostra que somente

7,5% das páginas são responsáveis por 50% dos acessos à memória (Figura 6.a). Quando o parâmetro K=20, o LRU-WARlock supera o LRU-WAR em todos os tamanhos de memória, com ganho médio de quase 16%. O pico de desempenho foi obtido para uma memória de 1400 páginas, com um ganho de 39,3% em relação ao LRU-WAR.

Com o parâmetro de controle K=50, a Região Reservada ocupa 50% do total da memória. Uma característica interessante do LRU-WARlock é em relação às memórias de tamanho pequeno, pois neste caso é garantido que metade dos acessos são cobertos com apenas 400 páginas de memória, para um *trace* como o multil1, com 2606 páginas diferentes. O aumento de desempenho foi de, na média e de pico, respectivamente, 19% e 42% em relação ao LRU-WAR.



(a) Páginas versus Acessos (b) Variação do parâmetro K.

Figura 6. Gráficos do *trace* multi1.

Quando a Região Reservada tem tamanho 90% do total da memória, o maior desempenho continua sendo para o tamanho de memória igual a 1400, que é de quase 50% melhor que a desempenho do algoritmo LRU-WAR. O ganho em desempenho médio também é alto e quase chega a 26% (Figura 6.b)

Um fato importante que foi observado na análise dos *traces* foi com relação ao tempo em que o LRU-WARlock fica no modo de operação seqüencial. Em todos os casos estudados, o período em que o algoritmo trabalha com deteção de acessos seqüenciais aumentou. Para o multi1, o algoritmo LRU-WAR ficou em torno de 11% do tempo em operação seqüencial. Já o algoritmo LRU-WARlock para K=50, este tempo aumentou 5 vezes, chegando a 56%. Com K=90, o algoritmo LRU-WARlock fica em média 86% em operação seqüencial. Este resultado mostra que com essa estratégia fica mais fácil detectar acessos seqüenciais presentes nos acessos à memória, aumentando sua eficiência.

Multi2

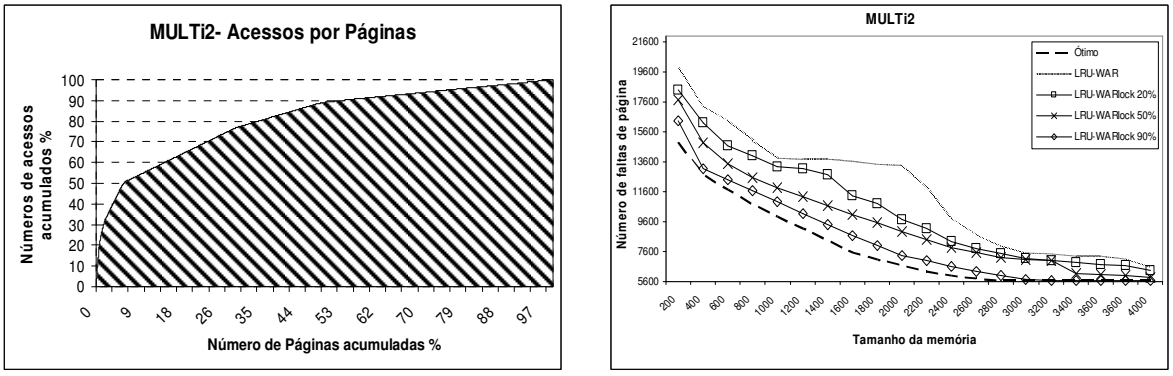
A maior característica do *trace* multi2 para a análise surge do fato de ser semelhante ao multi1. A diferença entre eles é a adição do programa postgres, desta forma aumentando o nível de multiprogramação. O desempenho do LRU-WAR diminui, pois se torna mais difícil ainda distinguir padrões de acessos com a inserção de novas referências intercaladas (Figura 4.a). A estratégia do LRU-WARlock também garante bom desempenho para este *trace*

(Figura 7.b). Deste modo, algoritmos que utilizam técnicas para separar padrões podem obter bons resultados.

A análise do perfil de freqüência de acessos às páginas mostra que metade das 26311 referências é feita por apenas 348 páginas de um total de 5684 páginas, ou seja, somente 6% das páginas são responsáveis por 50% dos acessos a memória (Figura 7.a).

Quando K=20, o LRU-WARlock supera o LRU-WAR em todos os tamanhos de memória, com uma média de ganho de 11%, desconsiderando tamanhos de memória maiores que 3200, quando então as curvas de faltas de página se convergem para todos os algoritmos analisados (Figura 7.b). O maior ganho de desempenho foi obtido para uma memória de 2000 páginas, com um ganho de 27% em relação ao LRU-WAR e quase 31% em relação ao LRU com memória de 2200 páginas.

Para K=50, trabalhando com metade da memória reservada para páginas de acessos freqüentes, o LRU-WARlock apresenta um comportamento interessante. A curva representada no gráfico começa a se alinhar com a curva do algoritmo Ótimo, diminuindo ainda mais sua identidade com a curva do LRU. Com essa nova característica, o ganho médio aumenta consideravelmente para 18%, obtendo uma tolerância maior para todos os tamanhos de memória. Um exemplo claro desse comportamento é o ganho obtido para memória de 1400 páginas, para K=20 foi de 7%, e para K=50 com mesmo tamanho de memória o ganho é triplicado, 22% em relação ao LRU-WAR. O ápice do desempenho se encontra na simulação com memória com tamanho de 2000 páginas, com ganho relativo ao LRU-WAR de 33%, comparado com o valor de 27% quando K=20.



(a) Páginas versus Acessos (b) Variação do parâmetro K.

Figura 7. Gráficos do *trace* multi2.

Quando K=90, o LRU-WARlock superou o LRU-WAR em todos os tamanhos de memória com uma vantagem média muito superior, de quase 30%. Um fato que deve ser bem observado, relativo à aproximação do algoritmo testado ao Ótimo, é quando a memória tem tamanho de 400 páginas. Por muito pouco o LRU-WARlock não se iguala ao desempenho teórico do algoritmo Ótimo, com uma diferença de apenas 3,35% do total de faltas. O maior desempenho continua sendo com tamanho de memória de 2000 páginas, com quase 45%

melhor que o LRU-WAR (Figura 7.b). Estes resultados mostram que o LRU-WARlock é bem adequado para sistemas com política de memória global.

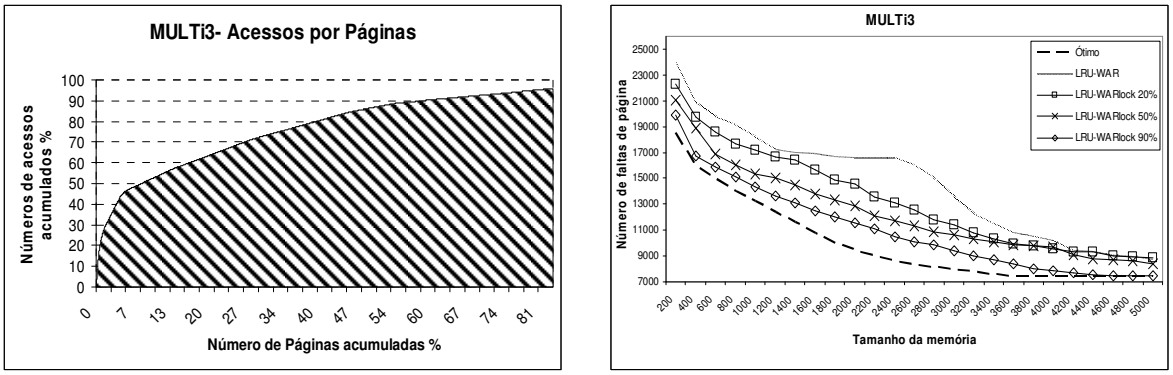
Multi3

Dos *traces* usados, aquele que tem o maior nível de multiprogramação é o multi3. São quatro programas com diversos padrões de acessos intercalados entre eles. Neste caso o algoritmo LRU-WAR toma decisões equivocadas entrando muitas vezes em operação seqüencial, e assim apresenta para alguns tamanhos de memória desempenho pior que o LRU. Como não existe uma uniformidade da localidade temporal, páginas que serão acessadas em um futuro próximo são descartadas prematuramente. Para esta seqüência de referências o LRU-WARlock também supera o LRU-WAR (Figura 8.b).

A variação do parâmetro K mostrou que quanto maior a área destinada às páginas mais freqüentes, melhor é a detecção das páginas seqüências pelo algoritmo, criando assim uma divisão de padrões. Quando o K=20 a média de ganho sobre o algoritmo LRU-WAR é de 10,5%, apenas 0,5 ponto percentual a menos comparado com o multi2. O pico de desempenho encontra-se na simulação com o tamanho de memória de 2800 páginas, com desempenho quase 22% melhor em comparação com o LRU-WAR.

Para K=50 o aumento no desempenho é linear em relação a K=20. Média de 17% de ganho em relação ao LRU-WAR e pico de quase 30% com o tamanho de memória 2600 páginas. A curva perde a característica herdada pelo LRU e se alinha à curva do algoritmo Ótimo.

Quando K=90 a média de ganho de desempenho é mais constante entre os tamanhos das memórias, com média de 26% comparado ao LRU-WAR. O maior desempenho obtido é sobre o tamanho de memória 2600 páginas, com 37% de ganho em relação ao LRU-WAR. Um ponto interessante da análise tem relação ao tamanho de memória de 4000 páginas, que teve o maior ganho em desempenho relativo ao parâmetro K=20, foi multiplicado por 3,85 vezes (ganho variando de 6% para 23%).



(a) Páginas versus Acessos (b) Variação do parâmetro K.

Figura 8. Gráficos do *trace* multi3.

Este comportamento pode ser explicado pelo fato de que as páginas mais acessadas dos *traces*, pertencem ao *working set* de cada programa. Essas páginas não são muitas, mas

tem uma freqüência de acesso muito grande. No *trace* multi3, 50% das 30241 referências são para somente 617 páginas das 7454 páginas que compõem o *trace*, ou seja, pouco mais de 8% do número total de páginas. Desta forma, a área não reservada da memória, que irá trabalhar com o algoritmo LRU-WAR, deve conter muitas páginas que são acessadas com pouca freqüência, facilitando assim, a detecção de padrões seqüenciais.

5. Conclusões e Trabalhos Futuros

Este trabalho apresentou um estudo sobre a adequação do algoritmo adaptativo LRU-WAR para políticas de gerência de memória global. Este estudo foi conduzido sobre o algoritmo LRU-WAR modificado, que foi chamado LRU-WARlock, o qual aplicou a técnica de *profiling* para determinação das páginas com maior freqüência de acessos e a inclusão de um novo parâmetro de controle K. Foram utilizados três *traces* com características de multiprogramação para os estudos de desempenho com políticas de gerência de memória global.

A contribuição mais importante deste artigo é mostrar que, apesar dos algoritmos adaptativos terem sido projetados para sistemas que utilizem políticas de gerência de memória locais, é possível ajustá-los para apresentarem também bom desempenho em sistemas de gerência de memória global. As análises revelaram que é imprescindível tratar o aspecto da freqüência nos acessos à memória, quando o sistema utilizar uma política de gerência de memória global. A prova desta tese está no fato observado em que, nos três *traces* estudados, menos de 9% do número total de páginas é responsável por 50% das referências á memória.

Para os três *traces* estudados, o algoritmo LRU-WAR entra em operação seqüencial, em média, em 13% do tempo. E o algoritmo LRU-WARlock fica em operação seqüencial em 41% para K=20 e chega a 89% para K=90. Combinado com o bom desempenho, pode-se afirmar que o LRU-WARlock passou a acertar mais as detecções de padrões seqüenciais, proporcionando uma melhoria de até 44% sobre o LRU-WAR.

Os resultados apontam que o LRU-WAR adequado para política de gerência global (LRU-WARlock) é uma boa alternativa de algoritmo adaptativo de substituição de páginas, e pode ser ajustado dinamicamente pelos seus parâmetros de acordo com a necessidade do sistema de gerência de memória. Além disso, o *overhead* de execução do algoritmo é menor porque o processamento responsável pela detecção de padrões do LRU-WAR é executado somente para aquelas páginas que não fazem parte da Região Reservada.

Alguns estudos complementares podem ser desenvolvidos para dar continuidade a este trabalho. Uma sugestão a considerar é incluir um mecanismo no LRU-WAR que trate a freqüência de acessos de forma dinâmica sem *profiling* e que não inviabilize sua implementação. Uma segunda sugestão é repetir o estudo realizado neste artigo para outros algoritmos adaptativos de substituição de páginas, de forma que se possa descobrir um padrão geral de adequação de algoritmos para políticas de gerência global da memória.

Além da freqüência de acessos, outras informações poderiam ser disponibilizadas ao algoritmo de substituição para auxiliar na gerência de memória, como por exemplo, o padrão de acessos a memória, composição do *working set*, última referência de cada página. Estudos sobre qual informação é mais relevante estão sendo conduzidos.

Referências

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

Cassettari, H. H. (2004). "*Análise da Localidade de Programas e Desenvolvimento de Algoritmos Adaptativos para Substituição de Páginas*." Dissertação de Mestrado. Escola Politécnica da Universidade de São Paulo, 2004.

Cassettari, H.H. and Midorikawa, E.T. (2004a) "*Caracterização de Cargas de Trabalho em Estudos sobre Gerência de Memória Virtual*", In Anais do III Workshop em Desempenho de Sistemas Computacionais e de Comunicação (WPerformance 2004), Salvador, BA.

Cassettari, H.H. and Midorikawa, E.T. (2004b) "*Algoritmo Adaptativo de Substituição de Páginas LRU-WAR: Exploração do Modelo LRU com Detecção de Acessos Seqüenciais*". In: Anais do I Workshop de Sistemas Operacionais (WSO 2004), Salvador, BA.

Denning, P.J. (1968) "*The working set model for program behavior*". In: Communications of the ACM 11, 5, pp. 323-333.

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

Intel. (2008). *"Profile-Guided Optimizations Overview"*. Disponível em http://www.intel.com/software/products/compilers/docs/flin/main_for/mergedprojects/optaps_for/common/optaps_pgo_ovw.htm. Acesso em 16/03/2008.

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

Kim, J.M. et al. "*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, 2000. *OSDI' 2000*: Proceedings. San Diego: USENIX, 2000 pp.119-134.

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, p.1352-1361.

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

Midorikawa, E.T., Piantola, R.L., Cassettari, H.H. (2007) "*Influência dos Parâmetros de Controle no Desempenho de Algoritmos Adaptativos de Substituição de Páginas*". In: Anais do IV Workshop de Sistemas Operacionais (WSO 2007), Rio de Janeiro, RJ.

Sabeghil, M. and Yaghmaee (2006), M. H. "*Using fuzzy logic to improve cache replacement decisions*". IJCSNS International Journal of Computer Science and Network Security, Seoul, v.6, n.3A, pages182-188.

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