Algoritmo de Substituição de Páginas 3P: Acrescentando Adaptatividade ao Clock

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. *Adaptive algorithms are capable of modifying their own behavior through time, depending on the execution characteristics. Recently, we proposed the LRU-WAR, an adaptive page replacement algorithm whose target is to minimize failures detected in LRU policy, preserving its simplicity. In this paper, we present a new online approach inspired by the LRU-WAR: the 3P adaptive algorithm. By detecting sequential accesses in an effective way, the new algorithm combines the high performance of LRU-WAR with the low overhead inherent in the Clock algorithm. The performance gain from Clock to 3P has achieved 69% in simulation experiments.*

Resumo. *Algoritmos adaptativos têm a capacidade de modificar o seu comportamento ao longo do tempo, de acordo com as características de processamento observadas. Recentemente, propusemos o LRU-WAR, um algoritmo adaptativo para substituição de páginas cujo objetivo é minimizar falhas detectadas na tradicional política LRU sem perder a sua simplicidade computacional. Neste artigo, apresentamos uma versão online inspirada no LRU-WAR: o algoritmo adaptativo 3P. Detectando acessos seqüenciais de maneira eficiente, o novo algoritmo alia o bom desempenho do LRU-WAR à baixa complexidade de implementação inerente ao algoritmo Clock. Em simulações realizadas, o ganho de desempenho do algorimo 3P em relação ao Clock original chegou a 69%.*

1. Introdução

Um dos aspectos mais importantes em sistemas com 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, ou seja, quando uma falta de página acontece. A dificuldade está em decidir qual é a página menos importante para a execução do processo no momento da substituição e, portanto, qual página deve deixar a memória. O uso de uma política eficiente nesta decisão se torna fundamental, visto que um bom gerenciamento de memória é característica indispensável para um sistema operacional executar satisfatoriamente tarefas que exigem alto desempenho (MIDORIKAWA, 1997).

O esquema de substituição de páginas mais empregado 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 triviais como FIFO, MRU, LFU, etc. Dentre suas versões *online*, a mais simples é representada pelo algoritmo Clock (CORBATÓ, 1969), ou Algoritmo do Relógio, que consiste em uma aproximação do LRU um pouco menos precisa, porém adequada para implementação em sistemas reais devido à baixa sobrecarga (*overhead*) que acrescenta.

Mas há situações em que o LRU certamente apresenta deficiências, as quais são “herdadas” naturalmente pelo algoritmo Clock. Uma situação clássica é quando o programa em execução exibe uma predominância de acessos seqüenciais envolvendo muitas páginas de memória, o que pode ocorrer por diversos motivos: iniciação ou atualização de elementos, processamento consecutivo – como multiplicação de matrizes –, busca seqüencial, entre outros. Um padrão de acessos à memória

deste tipo pode ser facilmente detectado em um processo devido a uma característica bem definida: muitas faltas de página acontecendo em um curto intervalo de tempo. Outra característica importante é que as páginas carregadas deixam de ser referenciadas rapidamente, sem que haja uma reutilização pontual significativa.

O conceito de adaptatividade em algoritmos de substituição de páginas oferece benefícios potenciais importantes, como a correção de deficiências presentes nas políticas tradicionais, ou triviais, e a possibilidade de otimização dos 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* / LRU com Confinamento da Área de Trabalho), cujo objetivo é minimizar as falhas detectadas no algoritmo LRU sem sobrecarregar o sistema de gerenciamento de memória (CASSETTARI; MIDORIKAWA, 2004).

A evolução do algoritmo LRU-WAR é aqui descrita: o algoritmo *online* 3P (*3 Pointers* / 3 Ponteiros). Esta versão aproximada possui como grande diferencial o fato de se adequar totalmente à forma de operação do algoritmo Clock, característica que garante a viabilidade de sua implementação prática em sistemas operacionais reais. Isto significa que, além de o algoritmo 3P realizar operações de substituição somente quando uma falta de página ocorre, o peso computacional de tais operações é mínimo, variando de uma ordem O(1) no melhor caso – e provavelmente na maioria dos casos – até O(n) no pior, exatamente como acontece com o algoritmo Clock e seus derivados. Por outro lado, a eficiência de substituição do 3P é equivalente à eficiência do LRU-WAR em média, evidenciando uma ótima relação custo/benefício ou, mais precisamente, sobrecarga/eficiência.

A Seção 2 deste artigo comenta o estado atual de desenvolvimento no campo dos algoritmos adaptativos para substituição de páginas, destacando as principais alternativas *online* existentes e apresentando uma visão superficial do LRU-WAR, ponto de partida para a criação do algoritmo adaptativo *online* 3P, o qual é detalhado na Seção 3. A Seção 4 apresenta uma análise dos resultados de desempenho obtidos em simulações realizadas com os algoritmos Clock, CAR, CART, LRU-WAR e 3P, validando a nova proposta. A conclusão do trabalho é formalizada na Seção 5.

2. Algoritmos Adaptativos para Substituição de Páginas

Diversos algoritmos adaptativos para substituição de páginas foram publicados na última década, baseados, em sua maioria, no tradicional algoritmo LRU. Os algoritmos SEQ (GLASS; CAO, 1997) e EELRU (SMARAGDAKIS; KAPLAN; WILSON, 1999), por exemplo, utilizam ora o critério de substituição LRU, ora o critério MRU-n, na execução de um determinado processo. O objetivo comum de ambos é manter o comportamento do LRU em situações normais de processamento, e inverter o critério de substituição quando acessos seqüenciais são detectados, já que o MRU tende a apresentar um ótimo desempenho nestes casos – e somente nestes casos – e o LRU, um péssimo desempenho. O SEQ detecta exclusivamente acessos seqüenciais em páginas com endereços adjacentes de memória, enquanto o EELRU é um algoritmo muito mais genérico e, em média, mais eficiente. O algoritmo LRU-WAR segue na mesma linha do EELRU, contudo seu custo de implementação é muito menor, diminuindo a sobrecarga inserida no sistema.

Três outros algoritmos, DEAR (CHOI et al., 1999), AFC (CHOI et al., 2000) e UBM (KIM et al., 2000), procuram identificar padrões de acesso à memória dinamicamente nos programas e, assim, utilizar um critério de substituição ótimo de acordo com cada padrão, incluindo acessos seqüenciais. As demais propostas analisam, em última instância, a freqüência de reutilização recente das páginas residentes. Destacam-se nesta linha os algoritmos LRFU (LEE et al., 2001), LIRS (JIANG; ZHANG, 2002) e ARC (MEGIDDO; MODHA, 2003), este último criado como uma versão adaptativa do algoritmo 2Q (JOHNSON; SASHA, 1994), o qual procura melhorar o desempenho do LRU através do isolamento, em uma fila separada, de páginas residentes não reutilizadas. Os algoritmos LRFU e LIRS, por sua vez, valem-se de mecanismos de controle adicionais na tentativa de proteger páginas com reutilização freqüente de uma eventual substituição, ainda no âmbito do modelo LRU puro. O primeiro acrescenta contadores de acesso às páginas e o segundo armazena um histórico dos dois últimos acessos realizados em cada página residente.

Todas estas propostas agregam benefícios conceituais valiosos aos algoritmos tradicionais nos quais se inspiram, mas consistem basicamente em contribuições teóricas, inviáveis em sistemas reais pela sobrecarga que acarretam. Como também ocorre com a maioria das políticas triviais, os algoritmos adaptativos citados necessitam efetuar operações – complexas, por vezes – em cada acesso à memória, onerando o sistema de maneira proibitiva. A solução para este problema é o desenvolvimento de versões aproximadas *online*, com sobrecarga aceitável, a partir dos algoritmos adaptativos originais. No entanto, o bom desempenho de tais algoritmos normalmente se deve a uma decisão de substituição precisa, embasada por mecanismos internos que monitoram continuamente os acessos à memória realizados pelo processo. Logo, a conversão de algoritmos de substituição em soluções práticas nem sempre é uma tarefa possível.

2.1. Algoritmos Baseados no Modelo Clock

Para que um algoritmo de substituição de páginas seja considerado *online*, viável em sistemas reais, é importante que o mesmo execute os seus procedimentos exclusivamente nos momentos em que uma falta de página acontece. Em tais momentos, o processo fica em espera enquanto uma página é trazida da memória secundária para a memória principal; conseqüentemente, o algoritmo de substituição pode atuar sem sobrecarregar excessivamente o sistema.

Em termos de algoritmos adaptativos, apenas três propostas foram publicadas neste contexto:

- CAR – *Clock with Adaptive Replacement* (BANSAL; MODHA, 2004): Versão *online* do algoritmo adaptativo ARC. Utiliza duas filas, complementares entre si, para armazenar os identificadores de cada página carregada na memória. Uma das filas engloba páginas residentes nas quais não se percebe uma tendência de reutilização pontual, enquanto a outra fila é dinamicamente preenchida por páginas que demonstram tal tendência. Ambas são controladas por ponteiros que se movimentam como no algoritmo Clock. A primeira fila tem prioridade na substituição, isto é, a probabilidade de uma página ser substituída na primeira fila é maior do que na segunda. Páginas substituídas permanecem nas filas por um certo tempo após deixarem fisicamente a memória, servindo como uma base histórica recente para a orientação do algoritmo. O tamanho das filas, gerenciado de modo dinâmico, indica qual delas deve ter uma página removida em caso de falta de página.
- CART – *CAR with Temporal filtering* (BANSAL; MODHA, 2004): Variação do algoritmo CAR que inclui um mecanismo adicional para diferenciar páginas referenciadas constantemente daquelas muito referenciadas em um único período. Utilizando um novo bit como filtro, o algoritmo classifica as páginas como sendo de "reutilização localizada" ("*short-term utility*") ou de "reutilização constante" ("*long-term utility*"). Páginas do primeiro grupo têm prioridade de substituição quando se tornam inativas na memória. Tanto a lógica como a estrutura operacional do algoritmo CAR são mantidas, entretanto uma complexidade maior de implementação é acrescentada.
- Clock-Pro (JIANG; CHEN; ZHANG, 2005): Versão *online* do algoritmo adaptativo LIRS que, utilizando três ponteiros circulares e independentes sobre uma única fila LRU, procura distinguir páginas com maior ou menor probabilidade de reutilização, classificadas respectivamente como "quentes" ou "frias" ("*hot*" ou "*cold*"). Esta classificação é baseada na reutilização recente de cada página e somente páginas "frias" são substituídas quando uma falta de página ocorre. De maneira similar aos algoritmos CAR e CART, o Clock-Pro mantém na fila algumas páginas recém-substituídas como referencial adicional temporário.

Outros algoritmos de substituição *online* podem ser citados, como por exemplo: GClock – *Generalized Clock* (SMITH, 1978), WSClock (CARR; HENNESSY, 1981) e *Two-Handed Clock* (LEVY; LIPMAN, 1982). Estas variações do Clock, todavia, comportam-se de modo fixo na presença de qualquer eventual padrão de acessos à memória; ou seja, não são algoritmos adaptativos e, como o próprio Clock, podem apresentar desempenhos muito ruins em algumas situações pouco favoráveis às premissas consideradas em seu desenvolvimento.

2.2. Algoritmo LRU-WAR

O algoritmo adaptativo LRU-WAR (CASSETTARI; MIDORIKAWA, 2004), resumido nesta seção, destaca-se por detectar e tratar de forma eficiente períodos com predominância de acessos seqüenciais à memória, mantendo praticamente a mesma complexidade estrutural e de implementação que o algoritmo LRU tradicional, do qual se originou. Inspirado no conceito de *working set*, que pode ser descrito como o conjunto das páginas acessadas pelo programa em um determinado intervalo de tempo (DENNING, 1968), o LRU-WAR inova ao monitorar a fila LRU e verificar, entre duas faltas de página consecutivas, a página referenciada com menor recência em relação ao seu último acesso. Ou seja, o algoritmo identifica a posição mais alta da fila LRU que recebeu acessos no período, sendo que tal posição, representada por W, limita a chamada "área de trabalho".

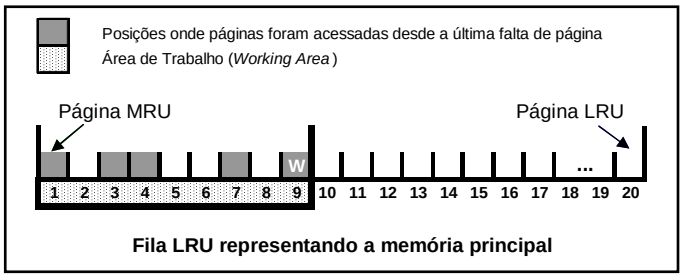

Figura 1. Exemplo de área de trabalho identificada entre duas faltas de página consecutivas.

Definimos o conceito de área de trabalho, exemplificada na Figura 1, como sendo a região de recência onde o *working set* temporário (conjunto das páginas acessadas desde a última falta de página ocorrida) está confinado, isto é, a porção inicial da fila LRU em que todos os acessos realizados se concentraram desde a falta anterior. A área de trabalho consiste, pois, no trecho da fila LRU compreendido entre a primeira posição – página MRU – e a W-ésima posição da mesma.

Dois critérios de substituição são empregados pelo algoritmo LRU-WAR: o critério LRU, segundo o qual a página acessada há mais tempo é retirada da memória, e o critério MRU-n, pelo 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 muito rapidamente, de acordo com o tamanho momentâneo da área de trabalho identificada, contudo mantém o critério LRU e aguarda durante um certo período de carência até que a tendência se intensifique. Nesta situação, o critério de substituição passa a ser o MRU-n. A Tabela 1 resume os três estados de execução previstos pelo LRU-WAR, onde M corresponde ao tamanho total da memória e, portanto, à última posição da fila LRU. O parâmetro L estabelece um tamanho mínimo para que a área de trabalho seja considerada grande o suficiente para indicar uma reutilização de páginas residentes adequada ao bom funcionamento da política LRU.

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

A idéia principal é substituir uma página que, por tendência, esteja se tornando inativa: uma página que deixou de pertencer à área de trabalho há pouco tempo. O ponto de substituição precoce (MRU-n) é sempre definido e controlado de maneira adaptativa pelo algoritmo, de acordo com uma potencial variação de tamanho observada no *working set* temporário do processo. Com a detecção eficiente de padrões de acesso seqüenciais, a substituição sistemática de todas as páginas residentes – que comumente acontece na presença de tais padrões – pode ser finalmente evitada. Simulações demonstraram uma melhora de desempenho, por parte do LRU-WAR, de até 75% em relação ao algoritmo LRU (CASSETTARI, 2004).

3. O Novo Algoritmo 3P

Embora o algoritmo LRU-WAR apresente uma versão *online* relativamente trivial, a qual atualiza a fila LRU somente quando há uma falta de página, tal versão ainda exibe um custo de implementação proibitivo devido justamente às operações de atualização na fila. A complexidade de uma substituição LRU-WAR é invariavelmente de ordem O(n). O desenvolvimento de uma variação *absolutamente online*, com custo de implementação aceitável, fazia-se necessário. Esta foi a principal motivação que nos levou à proposta do novo algoritmo adaptativo 3P (3 Ponteiros).

3.1. Idéia Geral

O intuito fundamental do algoritmo 3P é o mesmo que o do algoritmo LRU-WAR: identificar a presença de acessos seqüenciais à memória e trabalhar eficientemente em tal situação, sem acrescentar sobrecarga significativa ao sistema. Outrossim, ambos os algoritmos mantém a "recência entre acessos a uma mesma página" como critério básico para orientar a sua decisão de substituição nos demais casos. Logo, o LRU-WAR e o 3P são algoritmos que seguem exatamente o mesmo conceito, porém enquanto o primeiro enfoca sobretudo o aspecto da precisão teórica, o segundo enfoca essencialmente o aspecto da viabilidade prática de implementação.

O esquema de atuação do 3P pode ser visualizado na Figura 2. Como o próprio nome do algoritmo evidencia, três ponteiros são utilizados para percorrer a fila circular que representa a memória principal: CLOCK, ANTECIPADO e APAGADOR. Os três ponteiros se movimentam em conjunto e, conseqüentemente, a distância entre eles permanece sempre a mesma. A chamada "área jovem" corresponde às posições da fila situadas entre o ponteiro ANTECIPADO e o ponteiro APAGADOR. Em tendência ou operação seqüencial, é nesta área que se localizam as páginas "jovens", recém-carregadas, em período de teste de reutilização.

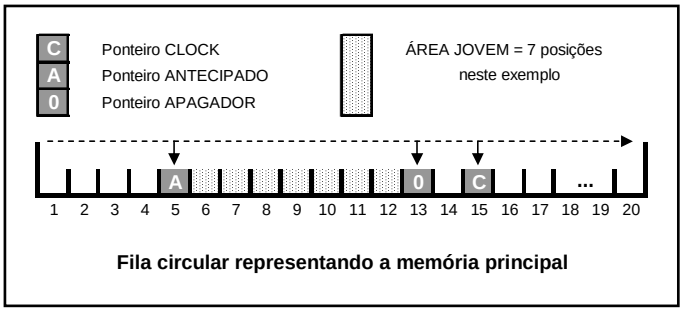

Figura 2. Estruturas e esquema básico de funcionamento do algoritmo 3P.

Um novo bit associado a cada página – adicionado e controlado por software – auxilia o algoritmo em suas operações. Quando setado, o "bit jovem", como é denominado, indica que uma página recém-carregada ainda não passou nenhuma vez pelo ponteiro ANTECIPADO. O papel deste ponteiro é servir como um ponto de substituição precoce, sem que uma maior complexidade computacional seja inserida no algoritmo. O ponteiro APAGADOR filtra a presença de reuso imediato (acessos que eventualmente se repetem quando a página é carregada, por um período de tempo muito curto, inclusive quando padrões seqüenciais vigoram), cuja ocorrência é muito comum e pode ser perniciosa: reuso imediato não significa necessariamente localidade temporal. Por sua vez, o ponteiro CLOCK funciona e se movimenta como o ponteiro tradicional do algoritmo Clock. Os outros dois ponteiros acompanham seus movimentos em uma única direção.

Fazendo uso adequado do conjunto de ponteiros descrito, o 3P identifica situações em que: (1) muitas faltas de página acontecem em um curto intervalo de tempo; e (2) as páginas carregadas deixam de ser referenciadas rapidamente, sem que haja uma reutilização pontual relevante. Como levantado nas seções anteriores, estas situações indicam uma forte tendência de que o padrão de acessos à memória é predominantemente seqüencial no período em questão, o que sugere a utilização de um critério de substituição alternativo, mais especificamente o critério MRU-n. O ponto de substituição antecipado habilita o emprego de tal critério.

3.2. Estados de Operação

O algoritmo 3P admite os mesmos três estados de execução consolidados pelo algoritmo LRU-WAR: tendência original (ou tendência LRU), tendência seqüencial (quando detecta indícios de acessos seqüenciais) e operação seqüencial (quando a tendência seqüencial se fortalece substancialmente). No decorrer da execução de um programa, o algoritmo verifica, a cada falta de página, se a página apontada pelo ponteiro CLOCK possui o bit de acesso setado. Em caso positivo, assume a tendência original, pois aparentemente há uma reutilização de páginas satisfatória. Em caso negativo, inicia a tendência seqüencial e nela permanece durante um certo período de carência, desde que as páginas apontadas por CLOCK continuem apresentando bits de acesso zerados. Vencido o período de carência – que representa a ocorrência de um determinado número de faltas de página –, entra em vigor o modo de operação seqüencial do algoritmo, impondo exigências para as páginas apontadas desde então pelo ponteiro ANTECIPADO: "bit jovem" setado e bit de acesso zerado. Se qualquer página apontada por ANTECIPADO não seguir as exigências citadas, o algoritmo volta imediatamente a operar em tendência seqüencial ou tendência LRU, de acordo com o bit de acesso associado à página apontada por CLOCK. A Tabela 2 retrata de forma mais objetiva os três tipos de execução previstos pelo 3P.

Tabela 2. Estados de execução definidos pelo algoritmo 3P.

ALGORITMO 3P				
Estado de execução	**Condições (A = ANTECIPADO, C = CLOCK, PC = Período de Carência)**	**Período de Carência**	**Critério Teórico de Substituição**	**Ponto de Substituição (Ponteiro)**
Operação Seqüencial	A→BIT = 0 e A→BIT_JOVEM = 1 e PC = 0	Zerado	MRU-n	ANTECIPADO
Tendência Seqüencial	Fora de Operação Seqüencial e C→BIT = 0	Diminui	LRU	CLOCK
Tendência Original (LRU)	Fora de Operação Seqüencial e C→BIT = 1	Aumenta	LRU	CLOCK (após movimento)

Estando em tendência seqüencial, a estabilidade de comportamento é fator essencial para que o algoritmo passe a operar em modo seqüencial, estabilidade esta que só é constatada após o período de carência vigente ter sido respeitado. Tal inércia comportamental pressupõe que páginas estão sendo carregadas continuamente na memória e rapidamente se tornando inativas, sugerindo a provável presença de acessos seqüenciais. A cada falta de página registrada em tendência original, o período de carência é redefinido em função do tamanho da "área jovem" e do número de deslocamentos sofridos pelos ponteiros até que uma página seja substituída por CLOCK.

3.3. Descrição Prática

O algoritmo 3P pode ser escrito na forma do pseudo-código exposto na Figura 3, o qual se refere especificamente aos procedimentos de substituição de página. Por conseguinte, nesta situação, a memória disponível já estará totalmente ocupada e os três ponteiros se encontrarão nas posições corretas. Os ponteiros são inseridos na medida em que as páginas que ocupam suas posições iniciais são carregadas por demanda na memória, em posições crescentes da fila. Sendo M o tamanho da memória principal disponível, a posição inicial do ponteiro CLOCK é sempre 1 (primeira posição da fila circular) e os ponteiros APAGADOR e ANTECIPADO ficam dele recuados a uma distância de, respectivamente, MIN(M/10; 10) e MIN(M/2; 50) posições. Tais distâncias foram definidas a partir de estudos previamente realizados sobre parâmetros do algoritmo LRU-WAR que, na prática, possuem a mesma finalidade: isolar uma área de observação para identificar acessos seqüenciais. Assim, a distância entre os ponteiros é pequena o suficiente para implementar um ponto de substituição MRU-n viável, mas não tão pequena a ponto de tornar imprecisa a detecção de padrões seqüenciais.

É importante lembrar que, quando uma página residente é referenciada (*hit*), o único procedimento necessário consiste na simples ativação do bit de acesso associado à mesma. Em geral, esta operação é executada automaticamente pelo hardware, sem a interferência do sistema operacional. O tamanho mínimo de memória aceito pelo 3P é de 10 páginas residentes. Abaixo deste tamanho, o algoritmo se comporta exatamente como o Clock original, situação representada pelas linhas 18, 19 e 20 do pseudo-código. O trecho compreendido entre as linhas 2 e 16 reflete as operações de substituição realizadas pelo 3P em situações normais, com uma memória de tamanho adequado. A rotina de movimentação dos três ponteiros em conjunto está localizada entre as linhas 24 e 30.

```
    Substituição 3P:
1.  Se M ≥ MEMÓRIA_MÍNIMA:    /* MEMÓRIA_MÍNIMA é originalmente igual a 10 páginas */
2.          Se ANTECIPADO→BIT = 0 e ANTECIPADO→BIT_JOVEM = 1 e CARÊNCIA = 0:
3.                  Remove da memória ANTECIPADO→PÁGINA;    /* Operação Seqüencial */
4.                  ANTECIPADO→PÁGINA = CLOCK→PÁGINA;
5.                  ANTECIPADO→BIT = CLOCK→BIT;
6.          Senão (ANTECIPADO→BIT = 1 ou ANTECIPADO→BIT_JOVEM = 0 ou CARÊNCIA > 0):
7.                  Se CLOCK→BIT = 0:    /* Tendência Seqüencial */
8.                          Se CARÊNCIA > 0:
9.                                  CARÊNCIA = CARÊNCIA – 1;
10.                 Senão (CLOCK→BIT = 1):    /* Tendência Original (LRU) */
11.                         CARÊNCIA = TAMANHO_ÁREA_JOVEM - 1;
12.                         Enquanto CLOCK→BIT = 1:
13.                                 Se CARÊNCIA < M:
14.                                         CARÊNCIA = CARÊNCIA + 1;
15.                                 MOVIMENTA();
16.                 Remove da memória CLOCK→PÁGINA;
17. Senão (M < MEMÓRIA_MÍNIMA):    /* Algoritmo Clock Original */
18.         Enquanto CLOCK→BIT = 1:
19.                 MOVIMENTA();
20.         Remove da memória CLOCK→PÁGINA;
21. Insere a página carregada em CLOCK→PÁGINA;    /* Em qualquer estado de operação */
22. CLOCK→BIT_JOVEM = 1;
23. MOVIMENTA().

    Rotina MOVIMENTA():    /* Responsável pela movimentação dos três ponteiros em conjunto */
24. ANTECIPADO→BIT_JOVEM = 0;
25. Avança ANTECIPADO em uma posição;
26. Se APAGADOR→BIT_JOVEM = 1:
27.         APAGADOR→BIT = 0;
28. Avança APAGADOR em uma posição;
29. CLOCK→BIT = 0;
30. Avança CLOCK em uma posição.
```

Figura 3. Pseudo-código referente ao algoritmo de substituição 3P.

4. Avaliação de Desempenho

Diversas simulações foram efetuadas para avaliar o desempenho do algoritmo 3P e compará-lo com outros algoritmos. Cada simulação considerou uma carga de trabalho – um arquivo de *trace* (UHLIG; MUDGE, 1997) – e um tamanho específico de memória disponível. Além do caso ótimo (BELADY, 1966), dados relativos a cinco algoritmos de substituição foram coletados: Clock, CAR, CART, 3P e LRU-WAR, este último de natureza *offline*.

Tabela 3. *Traces* utilizados como cargas de simulação nos experimentos.

Arquivo	Descrição Resumida	Páginas Distintas Acessadas	Total de Acessos à Memória
Espresso	Simulador de circuito	77	326938361
GCC	Compilador C/C++ do projeto GNU, versão 2.7.2	458	37524334
Gnuplot	Gerador de gráficos do projeto GNU	7718	68458509
Grobner	Programa matemático que trabalha com Bases de Gröbner	67	7787835
GS	GhostScript 3.33, interpretador PostScript	558	134371942
Lindsay	Simulador de hipercubo	521	123690749
P2C	Tradutor de programas em Pascal para C	132	30722431
2_Pools	Simulador sintético	9939	100000
CPP	Pré-processador do projeto GNU para compilação de programas C	1223	9047
CS	Cscope, ferramenta que analisa códigos fonte de programas C	1409	6781
GLI	Glimpse, utilitário para busca de texto	2529	6015
Multi1	Execução simultânea de CS e CPP	2606	15858
Multi2	Execução simultânea de CS, CPP e PS	5684	26311
Multi3	Execução simultânea de CPP, Gnuplot, GLI e PS	7454	30241
PS	Postgres, banco de dados relacional (operações de consulta)	3083	10448
Sprite	Servidor de arquivos em rede (requisições efetuadas por clientes)	7075	133996

4.1. Simulações Realizadas

A Tabela 3 apresenta 16 programas, cujos *traces* foram utilizados como carga de simulação nos experimentos realizados. A segunda coluna da tabela descreve resumidamente cada programa. A terceira coluna indica o número de páginas distintas referenciadas pelos mesmos durante o seu processamento, sendo informado, na quarta coluna, o número total de acessos realizados à memória.

Espresso, GCC, Gnuplot, Grobner, GS, Lindsay e P2C compõem um pacote de *traces* originalmente aplicado por Smaragdakis; Kaplan; Wilson (1999) nas simulações do algoritmo EELRU, sendo também por eles disponibilizado. Por sua vez, 2_Pools, CPP, CS, GLI, Multi1, Multi2, Multi3, PS e Sprite representam o conjunto de programas empregados por Jiang; Zhang (2002) nas simulações do algoritmo LIRS, também separadamente usados em estudos anteriores (CHOI et al., 1999); (CHOI et al., 2000); (KIM et al., 2000); (LEE et al., 2001).

Tabela 4. Simulações consideradas para a avaliação dos algoritmos.

Arquivo	Tamanhos de Memória Simulados	Intervalo entre Tamanhos	Número de Simulações	Aumento percentual médio no número de faltas de página em relação ao caso ótimo				
				Clock	CAR	CART	3P	LRU-WAR
Espresso	10, 11, 12, ..., 73, 74, 75	1	66	114,98%	114,14%	102,99%	80,60%	66,88%
GCC	10, 15, 20, ..., 445, 450, 455	5	90	86,06%	86,00%	84,74%	82,13%	79,26%
Gnuplot	100, 200, 300, ..., 7500, 7600, 7700	100	77	65,72%	65,75%	60,87%	0,71%	0,47%
Grobner	10, 11, 12, ..., 63, 64, 65	1	56	139,20%	139,34%	118,21%	108,06%	125,40%
GS	10, 15, 20, ..., 545, 550, 555	5	110	66,15%	72,45%	66,21%	59,33%	55,75%
Lindsay	10, 15, 20, ..., 510, 515, 520	5	103	44,05%	45,02%	51,07%	32,80%	32,80%
P2C	10, 15, 20, ..., 120, 125, 130	5	25	148,63%	148,18%	136,32%	130,71%	136,50%
2_Pools	100, 200, 300, ..., 9700, 9800, 9900	100	99	58,79%	57,95%	58,04%	58,81%	59,00%
CPP	50, 100, 150, ..., 1100, 1150, 1200	50	24	15,40%	7,40%	7,05%	13,89%	12,27%
CS	50, 100, 150, ..., 1300, 1350, 1400	50	28	100,06%	98,64%	99,26%	10,44%	3,92%
GLI	50, 100, 150, ..., 2400, 2450, 2500	50	50	35,12%	28,08%	26,94%	8,00%	6,72%
Multi1	50, 100, 150, ..., 2500, 2550, 2600	50	52	53,46%	38,12%	38,40%	17,91%	42,10%
Multi2	100, 200, 300, ..., 5400, 5500, 5600	100	56	37,55%	24,66%	19,94%	18,39%	34,79%
Multi3	100, 200, 300, ..., 7200, 7300, 7400	100	74	35,65%	27,53%	22,22%	22,62%	34,01%
PS	50, 100, 150, ..., 2950, 3000, 3050	50	61	33,20%	28,42%	22,96%	9,42%	1,54%
Sprite	100, 200, 300, ..., 6800, 6900, 7000	100	70	18,67%	23,68%	26,27%	25,13%	21,36%
			1041	**65,79%**	**62,84%**	**58,84%**	**42,43%**	**44,55%**

A Tabela 4 detalha as situações analisadas em nossos experimentos, citando as baterias de simulação às quais os algoritmos foram submetidos. Mostra também os resultados médios obtidos pelos mesmos, levando-se em conta todas as simulações realizadas com um determinado programa. A segunda coluna da tabela relata os tamanhos de memória considerados, enquanto a terceira coluna indica o intervalo entre estes tamanhos e a quarta contabiliza o número total de simulações com cada *trace*. Os percentuais visualizados nas cinco últimas colunas informam o acréscimo médio no número de faltas de página gerado pelos algoritmos, em suas respectivas simulações, quando comparados com o caso ótimo. Quanto maior um percentual, pior o desempenho médio do algoritmo, e vice-versa.

4.2. Análise dos Resultados Obtidos

Os dados da Tabela 4 demonstram que o algoritmo 3P alcança bons resultados na grande maioria das situações. Seu desempenho médio é próximo do ótimo com programas que apresentam predominância de acessos seqüenciais à memória, como Gnuplot, CS, GLI e PS. Não obstante, tende também a superar – às vezes em muito – ou se igualar ao algoritmo Clock, em termos de desempenho, na execução dos demais programas. Portanto, o objetivo da proposta é atingido: grande eficiência na substituição em meio a uma sobrecarga operacional extremamente baixa. Sua complexidade computacional é praticamente equivalente à do algoritmo Clock e inferior à dos demais algoritmos.

A Figura 4 é composta por gráficos construídos a partir de resultados coletados nas simulações. Os gráficos exibem a variação percentual das faltas de página geradas pelo algoritmo 3P em relação às faltas geradas pelo algoritmo Clock, permitindo assim uma comparação mais precisa entre a nova proposta e a política tradicional da qual se originou. Seis casos representativos foram selecionados, três deles caracterizados por uma forte presença de acessos seqüenciais à memória (CASSETTARI, 2004), posicionados à direita na figura.

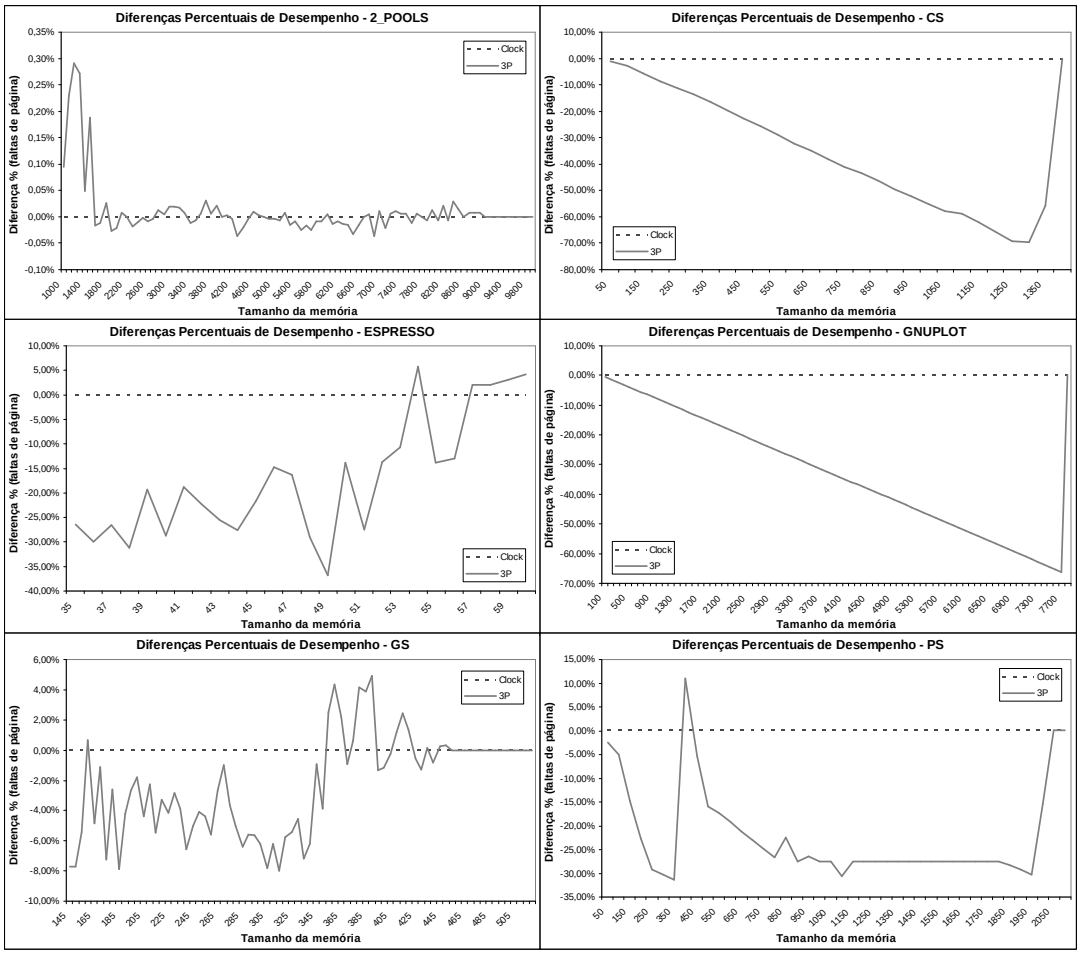

Figura 4. Variação de desempenho do algoritmo 3P em relação ao Clock.

Percebe-se que o desempenho do algoritmo 3P pode ser melhor ou pior que o do algoritmo Clock, porém tende a ser melhor, com picos muito positivos em alguns casos críticos. O fato mais importante é que, nas situações em que o 3P é superado pelo Clock, seu desempenho não se distancia significativamente do caso ótimo. Em outras palavras, o 3P não demonstrou nenhum resultado inaceitável ou mesmo preocupante nas simulações realizadas. Por outro lado, seus melhores resultados foram muito satisfatórios, o que não se repetiu na atuação dos demais algoritmos *online*. Embora sejam mais eficientes do que a política Clock em média, o desempenho dos algoritmos CAR e CART é geralmente inferior ao do algoritmo 3P. Este mantém a eficácia média do algoritmo LRU-WAR, inclusive o superando em diversos momentos.

Alguns dados adicionais complementam a avaliação: o maior ganho de desempenho do 3P em relação ao Clock foi de 69,61% (programa CS, com memória disponível de 1300 páginas), enquanto o seu pior resultado comparativo gerou um número de faltas de página 19,74% superior àquele gerado pelo algoritmo Clock (programa Sprite, com memória disponível de 4300 páginas). Considerando as 1041 simulações descritas, em média, o 3P apresentou um desempenho 10,98% melhor que o do Clock, 9,41% melhor que o do CAR e 7,91% melhor que o do CART. Seu desempenho foi igual ou superior ao do algoritmo Clock em 844 casos. Por fim, o desempenho do 3P foi significativamente melhor que o do Clock – acima de 10% – em 356 situações simuladas e significativamente pior – idem – em apenas 16 casos.

5. Conclusão

Descrevemos e avaliamos, neste artigo, o algoritmo adaptativo para substituição de páginas de memória 3P, plenamente *online* e inspirado no algoritmo teórico recém-desenvolvido LRU-WAR. A

carga de simulação utilizada nos experimentos relatados foi composta por programas contendo diferentes características de acesso à memória, previamente estudados e disponibilizados por outros cientistas na forma de arquivos de *trace*, corroborando para a credibilidade dos resultados obtidos.

Tais experimentos confirmaram o bom desempenho esperado para o algoritmo 3P, motivando o aprofundamento dos estudos e a futura implementação da proposta em um sistema operacional com código aberto, como o Linux (GORMAN, 2004) ou o FreeBSD (McKUSIC; NEVILLE-NEIL, 2005). Também é interessante pesquisar a possibilidade de utilização do 3P em outros níveis de *memória cache* na hierarquia de memória, ou mesmo em *web caches*. Futuras adaptações podem ser idealizadas para adequar o funcionamento do algoritmo às necessidades de cada enfoque.

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.

BELADY, L.A. A study of replacement algorithms for a virtual storage computer. IBM Systems Journal, v.5, n.2, p.78-101, 1966.

CARR, R.W.; HENNESSY, J.L. WSClock – A simple and effective algorithm for virtual memory management. In: SYMPOSIUM ON OPERATING SYSTEMS PRINCIPLES, 8., Pacific Grove, 1981. *SOSP'81:* Proceedings. Pacific Grove: ACM, 1981. p.87-95.

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. Algoritmo adaptativo de substituição de páginas LRU-WAR: exploração do modelo LRU com detecção de acessos seqüenciais. In: WORKSHOP DE SISTEMAS OPERACIONAIS, 1., Salvador, 2004. *I WSO:* Anais. Salvador: SBC, 2004. CD-ROM.

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.

CORBATÓ, F.J. A paging experiments with the Multics System. In: FESHBACH, H.; INGARD, K. (Ed.) *Festschrift in Honor of P.M. Morse*. Cambridge: MIT Press, 1969. p.217-228.

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.

GORMAN, M. *Understanding the Linux virtual memory manager*. Upper Saddle River: Prentice Hall, 2004. 727p.

JIANG, S.; CHEN, F.; ZHANG, X. CLOCK-Pro: An effective improvement of the CLOCK replacement. In: ANNUAL TECHNICAL CONFERENCE, 10., Anaheim, 2005. *USENIX' 05:* Proceedings. Anaheim: USENIX, 2005. p.323-336.

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.

LEE, D. et al. LRFU: a spectrum of policies that subsumes the Least Recently Used and Least Frequently Used policies. IEEE Transactions on Computers, v.50, n.12, p.1352-1361, 2001.

LEVY, H.; LIPMAN, P.H. Virtual memory management in the VAX/VMS operating system. IEEE Computer, v.15, n.3, p.35-41, 1982.

McKUSICK, M.K.; NEVILLE-NEIL, G.V. *The design and implementation of the FreeBSD operating system.* Boston: Addison-Wesley, 2005. 683p.

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.

SMITH, A.J. Sequentiality and prefetching in database systems. ACM Transactions on Database Systems, v.3, n.3, p.223-247, 1978.

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