RICARDO LEANDRO PIANTOLA

Uso de técnicas e informações em algoritmos adaptativos para substituição de páginas

São Paulo
2010

RICARDO LEANDRO PIANTOLA

Uso de técnicas e informações em algoritmos adaptativos para substituição de páginas

Dissertação apresentada à Escola Politécnica da Universidade de São Paulo para obtenção do Título de Mestre em Engenharia Elétrica.

São Paulo
2010

RICARDO LEANDRO PIANTOLA

Uso de técnicas e informações em algoritmos adaptativos para substituição de páginas

Dissertação apresentada à Escola Politécnica da Universidade de São Paulo para obtenção do Título de Mestre em Engenharia Elétrica.

Área de Concentração:
Sistemas Digitais

Orientador:
Prof. Dr. Edson Toshimi Midorikawa

São Paulo
2010

À Dayane, minha esposa, com amor, admiração e gratidão, pelo apoio e incentivo que me fizeram trilhar o caminho que trilhei, e que me mantiveram nele até o fim, mesmo nos momentos mais difíceis.

AGRADECIMENTOS

Aos meus pais, Edna e Roberto, por simplesmente tudo que já realizei nesta vida.

A Deus, pelas oportunidades que me foram concedidas.

Ao meu orientador, o professor Edson Toshimi Midorikawa, pela paciência, confiança e apoio, desde o início da minha pesquisa.

Aos amigos Artur Baruchi e Denis Ogura, pela amizade, companheirismo, paciência e incentivo, sempre.

À professora Líria Matsumoto Sato e a todos os colegas do LAHPC(*Laboratory of Architecture and High Performance Computing*) pela solidariedade e pelo ótimo ambiente de trabalho.

Ao professor Jorge Kinoshita, pela leitura atenta do relatório de qualificação e apoio.

À Escola Politécnica da Universidade de São Paulo, pela oportunidade da realização da Pós-Graduação.

RESUMO

O desempenho do sistema de memória virtual depende diretamente da qualidade da política de gerência de memória. Estratégias podem ser desenvolvidas para melhorar tal desempenho: uma delas é criar novas políticas de gerência de memória que tenham, ao mesmo tempo, bom desempenho e simplicidade; outra maneira é desenvolver técnicas e incluir informações para auxiliar as políticas já existentes. Este trabalho procura mostrar uma estratégia para auxiliar políticas de substituição com a finalidade de obter bom desempenho em um sistema de gerência de memória, sem a necessidade de alterar o comportamento da política de substituição. Para isso, foi utilizada a técnica de busca antecipada de páginas em conjunto com a informação de frequência de acessos, obtida por meio de um método usado em processamento estatístico de linguagem natural. Os resultados mostram, além do bom desempenho, que a mesma estratégia pode ser adotada em qualquer algoritmo.

ABSTRACT

The virtual memory system performance depends directly on the quality of the memory management policy. Strategies can be developed to improve such performance: one of them is creating new memory management policies that present, at the same time, simplicity and good performance; another one is developing techniques and include information that will aid the policies that already exist. This paper aims to show a strategy that will aid replacement policies in order to obtain a good performance in a memory management system without changing the replacement policy behavior. To do so, a page prefetching technique along with information about access frequency, obtained through a method used in a statistical natural language processing, was used. The results show, besides the good performance, that the same strategy can be adopted in any algorithm.

SUMÁRIO

LISTA DE FIGURAS
LISTA DE TABELAS
LISTA DE ABREVIATURAS E SIGLAS

1. INTRODUÇÃO 12
1.1. Motivação 13
1.2. Objetivo 14
1.3. Contribuições Esperadas 14
1.4. Metodologia 15
1.5. Organização do Texto 16
2. CONCEITOS GERAIS DE GERÊNCIA DE MEMÓRIA 18
2.1. Gerência de Memória 18
2.1.1. Memória Virtual 19
2.1.2. Paginação 21
2.2. Algoritmos de Substituição de Páginas 24
2.2.1. Algoritmos Adaptativos 31
3. ADAPTABILIDADE, TÉCNICAS E INFORMAÇÕES 34
3.1. Algoritmo Adaptativo LRU-WAR 34
3.2. Estudo de Frequência de Acessos 37
3.3. Estudos dos n-gramas Aplicados à Gerência de Memória 38
3.4. Técnicas de *Profiling* e *Prefetching* 40
4. TÉCNICAS E INFORMAÇÕES NA GERÊNCIA DE MEMÓRIA 41
4.1. Ajuste nos Parâmetros de Controle do LRU-WAR 41
4.1.1. Configuração dos Testes 41
4.1.2. Resultados Experimentais e Análises 44
4.2. Frequência de Acessos e o Algoritmo LRU-WARlock 49
4.2.1. Descrição do Algoritmo 50
4.2.2. Configuração dos Testes 52
4.2.3. Resultados Experimentais e Análises 54
4.3. Padrão de Acessos e o Algoritmo LRU+ng 59

4.3.1. Descrição do Algoritmo 60
4.3.2. Configuração dos Testes 63
4.3.3. Resultados Experimentais e Análises 63
5. CONCLUSÕES 68
5.1. Contribuições 68
5.2. Trabalhos Futuros 69
LISTA DE REFERÊNCIAS 71

LISTA DE FIGURAS

Figura 1 – Metodologia utilizada para a realização de simulações 15
Figura 2 – Espaços de endereçamento virtual e real 19
Figura 3 – Memória virtual com o mecanismo de paginação 22
Figura 4 – Divisão lógica da fila LRU utilizada pelo algoritmo LRU-WAR .. 35
Figura 5 – Exemplo de operações do LRU-WAR 36
Figura 6 – Características do programa gnuplot. Padrão de acessos à memória à esquerda e desempenho do LRU-WAR à direita 43
Figura 7 – Características do programa grobner. Padrão de acessos à memória à esquerda e desempenho do LRU-WAR à direita..... 43
Figura 8 – Características do programa sprite. Padrão de acessos à memória à esquerda e desempenho do LRU-WAR à direita 44
Figura 9 – Variação dos parâmetros para o gnuplot. Carência mínima C à esquerda e Tamanho da região sequencial L à direita 45
Figura 10 – Variação dos parâmetros para o grobner. Carência mínima C à esquerda e Tamanho da região sequencial L à direita 47
Figura 11 – Ganho de desempenho do LRU-WAR no grobner em relação ao LRU 47
Figura 12 – Variação dos parâmetros para o sprite. Carência mínima C à esquerda e Tamanho da região sequencial L à direita 49
Figura 13 – Divisão lógica do algoritmo LRU-WARlock 50
Figura 14 – Pseudocódigo do algoritmo LRU-WARlock 51
Figura 15 – Característica do *trace* multi1. Padrão de acesso à memória à esquerda e Desempenho de alguns algoritmos à direita 53
Figura 16 – Característica do *trace* multi2. Padrão de acesso à memória à esquerda e Desempenho de alguns algoritmos à direita 53
Figura 17 – Característica do *trace* multi3. Padrão de acesso à memória à esquerda e Desempenho de alguns algoritmos à direita 54
Figura 18 – Gráficos do *trace* multi1. Páginas versus Acessos à esquerda e Variação do parâmetro K à direita 55

Figura 19 – Gráficos do *trace* multi2. Páginas versus Acessos à esquerda e Variação do parâmetro K à direita 57
Figura 20 – Gráficos do *trace* multi3. Páginas versus Acessos à esquerda e Variação do parâmetro K à direita 59
Figura 21 – Estrutura do algoritmo LRU+ng 61
Figura 22 – Pseudocódigo do algoritmo LRU+ng 62
Figura 23 – Gráficos do *trace* multi1. Busca sobre total de faltas à esquerda e Comparação entre LRU, LRU+ng e Ótimo à direita 64
Figura 24 – Gráficos do *trace* multi2. Busca sobre total de faltas à esquerda e Comparação entre LRU, LRU+ng e Ótimo à direita 66
Figura 25 – Gráficos do *trace* multi3. Busca sobre total de faltas à esquerda e Comparação entre LRU, LRU+ng e Ótimo à direita..... 67

LISTA DE TABELAS

Tabela 1 – Bigramas mais comuns no *New York Times* 39
Tabela 2 – Descrição dos *traces* gnuplot, grobner e sprite..... 42
Tabela 3 – Descrição dos *traces* multi1, multi2 e multi3 52

LISTA DE ABREVIATURAS E SIGLAS

ARC	– Adaptive Replacement Cache (algoritmo)
CPU	– Central Processing Unit
EELRU	– Early Eviction LRU (algoritmo)
FBR	– Frequency-Based Replacement (algoritmo)
FIFO	– First In, First Out (algoritmo)
IRR	– Inter-Reference Recency
LFU	– Least Frequently Used (algoritmo)
LIRS	– Low Inter-reference Recency Set (algoritmo)
LRFU	– Least Recently/Frequently Used (algoritmo)
LRU	– Least Recently Used (algoritmo)
LRU+ng	– Least Recently Used with n-grams prefetching (algoritmo)
LRU-WAR	– LRU with Working Area Restriction (algoritmo)
MAX	– Máximo valor
MFU	– Most Frequently Used (algoritmo)
MRU	– Most Recently Used (algoritmo)
NPV	– Número da Página Virtual
NRU	– Not Recently Used (algoritmo)
OPT	– Optimum (algoritmo)
PFF	– Page Fault Frequency (algoritmo)
RAM	– Random Access Memory
UBM	– Unified Buffer Management (algoritmo)
USP	– Universidade de São Paulo
WS	– Working Set (algoritmo)

1. INTRODUÇÃO

Na década de 40, Arthur W. Burks, Herman H. Goldstine e John von Neumann já discutiam o uso da hierarquia de memória no documento “Preliminary Discussion of the Logical Design of an Electronic Computing Instrument”, no qual diziam que foram “forçados a reconhecer a possibilidade de construir uma hierarquia de memórias, cada qual com maior capacidade que o seu precedente, mas que fosse acessível com maior latência.” Desde então, o conceito de hierarquia de memória fez com que gerações de projetistas se deparassem com o problema de buscar a melhor alternativa de gerência da memória. Hoje, 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 essa diferença é imprescindível criar novas estratégias de gerência de memória, que sejam mais robustas em desempenho.

No contexto da gerência de memória em sistemas operacionais, vários grupos de pesquisa têm desenvolvido trabalhos na área de algoritmos para gerência de memória virtual. Vários algoritmos para substituição de páginas têm sido propostos na literatura recente, em particular, Cassettari (2004) que, em sua dissertação de mestrado, avalia políticas de substituição e propõe uma nova estratégia adaptativa de substituição de páginas: o algoritmo LRU-WAR – *LRU with Working Area Restriction*. Esses algoritmos impulsionaram novos trabalhos na área, com a busca de uma solução dinâmica e adaptativa para a melhora contínua do uso da memória pelos programas.

A criação e o aperfeiçoamento de políticas de substituição, não são o único enfoque que desperta interesse em grupos de pesquisas atuais. Este trabalho visa estudar e avaliar especificamente algoritmos de substituição de páginas de memória. Em sistemas de alto desempenho, dentro do escopo de sistemas operacionais, o tema pertence ao gerenciamento da memória principal. O presente trabalho é composto pela análise do algoritmo LRU-WAR, modificação de seus parâmetros para melhoria de desempenho, a

criação de um novo algoritmo adaptativo baseado no LRU-WAR e a análise de técnicas e informações complementares no auxílio de algoritmos de substituição.

1.1. Motivação

O aumento da carga computacional tem motivado a evolução dos computadores. No contexto da computação moderna, a centralização de recursos tornou-se evidente com o atual ressurgimento das tecnologias de virtualização, como o VMware ESX Server e o sistema Microsoft Hyper-V (codinome Viridian), 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. Esses trabalhos comprovam que a aplicação do conceito de adaptatividade em algoritmos de substituição de páginas pode trazer benefícios bastante satisfatórios. Por consequência, algoritmos adaptativos podem diminuir muito o tempo global de processamento requerido por um sistema de alto desempenho, uma vez que tornam o gerenciamento da memória virtual paginada potencialmente muito mais eficiente.

Por terem processamento ou estrutura de dados mais complexos, algoritmos adaptativos podem onerar o sistema de memória. Várias propostas apresentadas constituem excelentes alternativas teóricas, mas impraticáveis em um sistema real devido à possível sobrecarga. Em algumas propostas, a idéia de implementação é muito bem elaborada e, com isso, são passíveis de obter uma versão *online* (viável) por meio de alguns ajustes e aproximações, porém perde-se muito da idéia teórica original.

O trabalho se concentra essencialmente em projetos adaptativos de substituição de páginas; Algoritmos com boa base teórica e que sejam eficientes em situações reais.

1.2. Objetivo

O objetivo deste trabalho é contribuir para o aprimoramento dos sistemas de memórias atuais. Isso será realizado através de estudos dos fatores que afetam diretamente o desempenho das políticas de substituição e alguns meios em potencial de como tratá-los.

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

O estudo sobre estratégias para auxiliar políticas de substituição páginas em um sistema de gerenciamento de memória global, será conduzido sobre o algoritmo LRU modificado, chamado LRU+ng, que aplicará a técnica de busca antecipada (prefetching) de páginas junto com a informação de frequência de acessos, obtida por meio de um método usado em processamento estatístico de linguagem natural. A proposta é passível de se concretizar em um sistema operacional de código livre.

1.3. Contribuições Esperadas

Por meio deste trabalho, pretende-se contribuir com as seguintes realizações:

- Estudo dos benefícios do controle dinâmico dos parâmetros de algoritmos adaptativos, na execução de certas aplicações;
- Relação entre localidade de programa e desempenho obtido em execução, considerando não só as políticas de substituição, mas também os métodos de gerenciamento de memória;
- Estudo analítico de técnicas e informação que possam auxiliar algoritmos de substituição de páginas, no reconhecimento de

padrões de acesso em um sistema de gerenciamento de memória virtual paginada. O algoritmos LRU+ng foi criado para esse experimento;

- Proposta de adequação do algoritmo LRU-WAR, para que obtenha melhor desempenho em um sistema de gerencia de memória global: a criação do novo algoritmo LRU-WARlock.

1.4. Metodologia

A análise dos algoritmos de substituição é feita por meio de simulação e desenvolvida no – LAHPC (*Laboratory of Architecture and High Performance Computing)* – Departamento de Sistemas Digitais da Escola Politécnica da USP. O estudo consiste basicamente de quatro passos:

- Confecção dos arquivos de *traces* (rastros);
- Desenvolvimentos dos simuladores do sistema de substituição de páginas;
- Execução dos experimentos sob os *traces* dos programas;
- Análise dos resultados obtidos.

O arquivo de *traces* é formado pela sequência dos acessos a memória realizados por uma ou várias aplicações. O *trace* define o comportamento do programa ou programas em relação a utilização da memória. O gerador de *traces* é o responsável pela coleta dos acessos à memória ao longo do processamento da aplicação alvo (UHLIG; MUDGE, 1997).

Após a obtenção dos arquivos de *traces*, várias simulações podem ser executadas com configurações de tamanho de memória diferentes, no intuito de avaliar o desempenho dos algoritmos de substituição. A figura 1 apresenta o processo tomado nesses estudos.

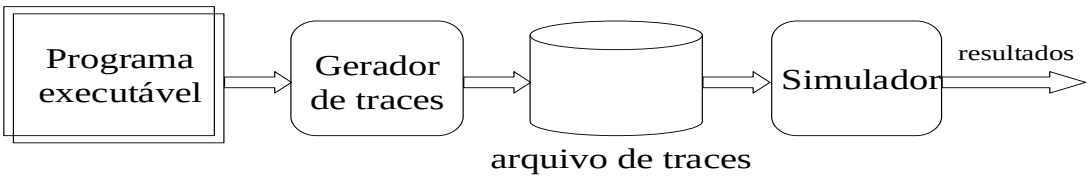


Fig. 1: Metodologia utilizada para a realização de simulações

Os dados de desempenho obtidos são então analisados sob o ponto de vista das características de localidade de referências presentes em cada aplicação. Ou seja, a eficiência ou não das políticas de substituição de páginas procura ser explicada de acordo com os padrões de acesso à memória observados nos programas.

As simulações são executadas em equipamentos disponíveis no laboratório, independente de detalhes de hardware. As máquinas possuem arquitetura x86, o sistema operacional é Linux ou Windows.

Os algoritmos são avaliados através de simuladores confeccionados em linguagem C e Perl. O simulador do algoritmo adaptativo EELRU, que complementa nossos estudos, foi cedido gentilmente por seus autores (SMARAGDAKIS; KAPLAN; WILSON, 1999). Esse simulador foi usado para fins de comparação de desempenho com os algoritmos propostos neste trabalho.

Procura-se determinar o melhor algoritmo de substituição de páginas no contexto específico de cada aplicação, com os resultados obtidos por meio das simulações, assim como a justificativa de um algoritmo ter mais desempenho que outro na situação. O método escolhido foi a relação tamanho da memória pelo total de faltas de páginas em uma execução, utilizando um *trace* considerado bastante relevante, conforme o objetivo a ser atingido. A ideia é que quanto mais faltas de páginas, maior o tempo gasto para concluir o processamento, portanto menor é o desempenho obtido.

1.5. Organização do Texto

O trabalho está organizado da seguinte forma: o Capítulo 2 revisa aspectos básicos sobre gerência de memória e o conceito de memória virtual. O foco principal é a alocação paginada. Nesse capítulo também são abordados os principais algoritmos de substituição de páginas existentes, tradicionais e adaptativos.

O Capítulo 3 detalha o funcionamento do algoritmo LRU-WAR, que é à base dos estudos deste trabalho. Também aborda a importância do tratamento da frequência de acessos, a qual é explorada na proposta do novo algoritmo LRU-WARlock. Discute o uso do modelo n-grama, descrevendo sua importância em processamento estatístico de linguagem natural e sua adequação em sistemas de gerência de memória. Outras técnicas são apresentadas nesse capítulo, tais como busca antecipada de páginas e *profiling*.

O Capítulo 4 apresenta o estudo do comportamento algoritmo LRU-WAR, expondo experiências sobre a modificação de seus principais parâmetros C e L. A nova proposta, LRU-WARlock, que utiliza técnica de *profiling* para coletar a informação de frequência, sem modificar o comportamento de seu algoritmo base, também é discutida. Uma análise minuciosa é feita utilizando uma política de memória global. Por último e não menos importante, há os experimentos sobre o algoritmo LRU+ng que utiliza a técnica de busca antecipada e as informações de frequência capturadas pelos n-gramas.

As conclusões do trabalho são expostas no Capítulo 5. A bibliografia utilizada está registrada na Lista de Referências logo após as conclusões.

2. CONCEITOS GERAIS DE GERÊNCIA DE MEMÓRIA

Neste capítulo aspectos básicos sobre gerência de memória e memória virtual são abordados. Também são apresentados os principais algoritmos de substituição de páginas, os tradicionais e adaptativos.

2.1. Gerência de Memória

Um dos componentes mais importantes e críticos do sistema operacional é o sistema de gerenciamento da memória. Na execução de programas que necessitam de grande quantidade de processamento e, por isso, demandam alto desempenho, a boa gerência de memória é fundamental. (CASSETTARI, 2004)

A memória principal é o local onde ficam armazenadas as informações necessárias para execução dos programas, o que inclui o seu código e dados. É sempre nela que o processador busca os dados e instruções que são necessárias para a realização de qualquer tarefa. Com o advento da multiprogramação, o gerenciamento de memória tornou-se uma atividade complexa e crucial, sob o ponto de vista da eficiência. Uma característica indispensável para um sistema operacional executar satisfatoriamente tarefas que demandam alto desempenho é um bom gerenciamento de memória.

Essencialmente, o sistema de gerenciamento da memória deve controlar e decidir quais dados, quando e por quanto tempo permanecerão residentes na memória principal do computador. Essa decisão deve levar em conta uma série de fatores, como o tamanho e a quantidade de processos aptos a executar, o ambiente operacional vigente, a área de memória disponível, o tipo de alocação implementada etc. São atribuições importantes do sistema de gerenciamento de memória: a definição e a manipulação de mecanismos de blocagem (segmentação e paginação), o carregamento e alocação física na memória principal dos dados provenientes da memória secundária e o gerenciamento da memória virtual.

O gerenciamento da memória virtual é um tópico importante, pois os algoritmos de substituição de páginas, que são os objetos de estudo deste trabalho, fazem parte da gerência de memória virtual.

2.1.1. Memória Virtual

A técnica que se utiliza da memória secundária para produzir o efeito de aumentar significativamente o espaço de endereçamento disponível aos programas é a memória virtual. Essa técnica não depende da memória principal e tão pouco de seu tamanho, para ser implementada. A memória secundária (geralmente um disco rígido) passa a servir como uma extensão da memória principal, armazenando programas e dados carregados para execução, como se fosse a própria memória principal. São transferidas para a memória principal apenas algumas partes destes programas e dados, sempre ao momento da execução. Uma instrução só pode ser executada e um dado só pode ser lido, se eles estiverem na memória principal.

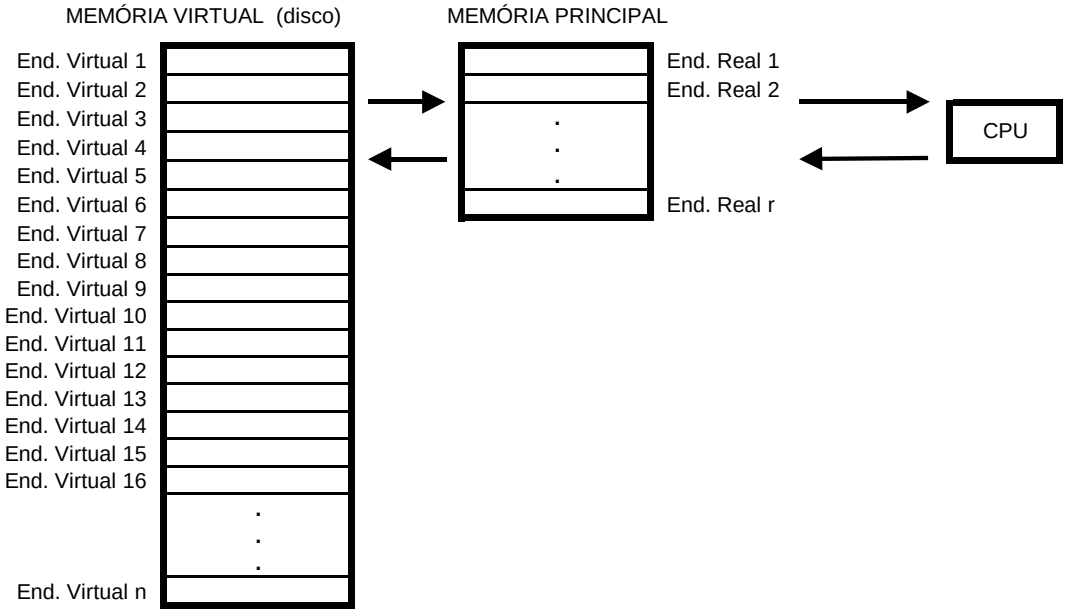


Fig. 2: Espaços de endereçamento virtual e real

Dessa maneira, são criados dois espaços de endereçamento de memória: o espaço de endereçamento real e o espaço de endereçamento virtual. Os endereços virtuais são essencialmente gerados pelos

compiladores e *linkers* na implementação dos programas. Eles representam o espaço completo que os programas necessitam para serem carregados, isto é, o tamanho de memória que efetivamente ocupam. No entanto, a memória real – ou memória principal (RAM – *Random Access Memory*) – pode ser bastante limitada, possuindo um espaço de endereçamento real às vezes menor até mesmo que um único programa. Um endereço real corresponde a uma célula física de endereçamento presente na memória principal da máquina.

Consequentemente, apenas parte dos programas carregadas no espaço de endereçamento virtual poderá ocupar a memória real num certo instante. Como já explicitado, somente os dados que estão na memória principal podem ser processados pela CPU. Tendo em vista a multiprogramação, várias aplicações podem estar sob execução, concorrendo por uma fatia de tempo do processador, a solução encontrada foi dividir o espaço total de endereçamento dos programas em pequenos blocos de tamanho fixo, as chamadas páginas de memória. A implementação dessa estratégia constitui o mecanismo de *paginação*. Outra alternativa é dividir o código de uma aplicação com base em critérios lógicos, ou seja, dividindo o programa de acordo com a função dos diversos trechos de código identificados. *Segmentação* é o nome que se dá a essa segunda estratégia de particionamento de código, em que os segmentos gerados possuem tamanhos independentes e variáveis. Tanto a segmentação como a paginação são mecanismos importantes, sobretudo em ambientes multiusuários e multiprogramáveis. De acordo com o momento de execução, parte dos dados são transferidos temporariamente para a memória principal com a finalidade de serem processados, retornando à memória secundária quando solicitados pelo sistema operacional, o qual é responsável pelo gerenciamento da memória e pelo mapeamento entre endereços virtuais e endereços físicos reais.

2.1.2. Paginação

Quando a paginação é o mecanismo de particionamento de dados utilizado, o espaço de endereçamento virtual de uma aplicação consiste em um conjunto de páginas de memória. Cada uma dessas páginas armazena uma quantidade fixa de dados, que varia de acordo com o sistema operacional. O tamanho de uma página normalmente se situa entre 512 bytes e 64 kbytes, sendo 4 kbytes um tamanho médio consideravelmente comum.

Um tamanho de página muito pequeno dificulta o gerenciamento por parte do sistema operacional, haja vista que cada página corresponde a uma entrada na *tabela de páginas*, responsável pelo mapeamento do espaço de endereçamento virtual. Logo, quanto menor uma página, maior a quantidade total de páginas endereçáveis e, consequentemente, maior a dificuldade de mapeamento. Além disso, uma página muito pequena não permite uma boa exploração da propriedade de localidade espacial, ou seja, pode haver necessidade de acesso ao disco mesmo quando endereços próximos entre si são referenciados num curto espaço de tempo.

Por outro lado, quando um tamanho de página muito grande é utilizado, a fragmentação na última página alocada à aplicação pode ser maior, além de diminuir o número de páginas que podem ser carregadas simultaneamente no espaço de endereçamento real. Ou seja, com uma página de tamanho maior, a necessidade de localidade espacial passa a ser ainda mais importante na aplicação, pois se menos páginas podem ser carregadas simultaneamente, maior será o número de acessos ao disco quando muitas páginas são referenciadas em um curto espaço de tempo.

As entradas na tabela de páginas representadas na figura 6 são compostas por alguns campos que permitem localizar o endereço efetivo de cada página na memória principal – quando carregada – ou na memória secundária (área de *swap*). Os campos principais são os seguintes:

- *Número da página virtual (NPV):* identifica uma única página virtual correspondente à entrada; de acordo com o tamanho de página utilizado, o endereço na memória secundária pode ser calculado.

- *Bit de presença:* indica se a página está ou não carregada na memória principal (a página está carregada se o *bit* possui valor 1).
- *Endereço do bloco:* se a página estiver carregada na memória principal, este campo informa o endereço físico em que está alocada; um bloco, ou janela *(frame)*, é um conjunto de células da memória principal que comporta dados de uma página.

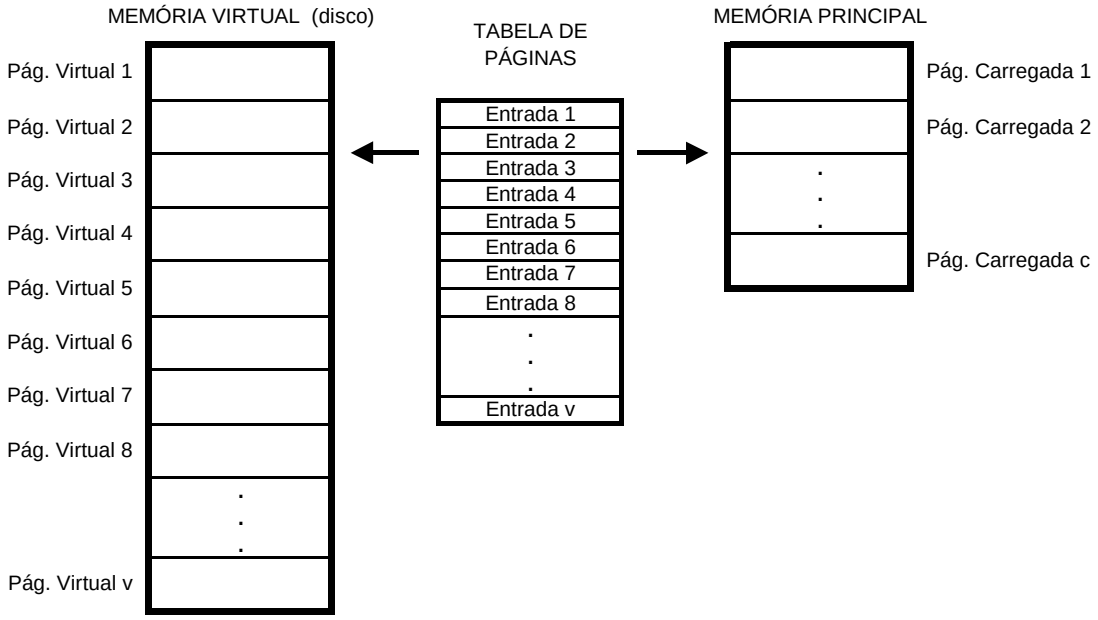


Fig. 3: Memória virtual com o mecanismo de paginação

- *Bits de proteção:* definem as permissões de acesso à página, normalmente através de 2 ou 3 bits (leitura, gravação e eventualmente execução).
- *Bit de modificação:* indica se a página carregada na memória teve ou não seu conteúdo alterado; se teve, o conteúdo deverá ser atualizado na memória secundária antes de a página deixar a memória principal.

Outros campos também podem fazer parte da tabela de páginas. Um *bit de referência*, por exemplo, pode ser utilizado para identificar as páginas menos ativas dentre as que estão carregadas na memória principal, o que pode ser essencial para algumas políticas de substituição de páginas.

Um endereço virtual é obtido através do número da página virtual e de um deslocamento interno específico. Esse deslocamento se refere à distância relativa do endereço procurado a partir do início da página. Da mesma forma, o endereço físico da página na memória principal é obtido somando-se o deslocamento interno ao endereço inicial do bloco onde a página está carregada.

Genericamente, em sistemas de memória virtual paginada, aplicações computacionais tendem a explorar, com maior ou menor grau de intensidade, a propriedade conhecida como localidade de referências. Essa propriedade afirma que apenas parte das páginas que compõem o espaço de endereçamento virtual de um processo são efetivamente necessárias à sua execução num certo intervalo de tempo. Em outras palavras, os dados processados em um determinado momento de execução normalmente se concentram em algumas poucas páginas. Essa propriedade é o que fundamenta e valoriza a importância da chamada hierarquia de memória, ou seja, a forma como os componentes físicos de memória são organizados na arquitetura de um computador. Dispositivos rápidos, porém dimensionalmente limitados, devem armazenar temporariamente os dados mais requisitados por um processo. Por outro lado, dispositivos de memória secundária – lentos, mas com grande capacidade de armazenamento – alocam a totalidade do espaço de endereçamento utilizado pelo processo.

O conceito de *working set* – conjunto de trabalho –, proposto por Denning (1968), também foi formalizado com base na propriedade de localidade de referências. O *working set* de um processo em execução é o conjunto das páginas requeridas para o seu processamento em um determinado intervalo de tempo. Ele representa, de certa forma, a concretização da propriedade de localidade: as páginas que compõem o conjunto retratam, por si só, uma localidade inerente ao momento de execução do processo.

Dizemos que ocorre uma falta de página quando um dado a ser processado se encontra em uma página que não está carregada na memória principal e, portanto, inacessível diretamente ao processador. Nesse caso, a

página precisa ser imediatamente alocada na memória para que a execução do processo continue. Contudo, se a memória principal já está totalmente preenchida, uma das páginas então carregadas deve ser sobreposta pela página que precisa ser trazida da memória secundária. Ocorre, assim, nesse momento, a necessidade de uma substituição de página. A dificuldade está em decidir qual página deve deixar a memória, isto é, qual a página menos importante para a execução do processo no momento da substituição, aquela que provavelmente não faz mais parte do seu *working set* corrente. O sistema operacional é o responsável por essa decisão e, para isso, implementa um algoritmo de substituição de páginas. O critério utilizado pelo algoritmo elege a página que deixa a memória quando uma substituição precisa ser efetuada.

A política de substituição de páginas é fator fundamental em qualquer sistema de memória com paginação. Esse aspecto é posteriormente detalhado na Seção 2.2 deste capítulo.

2.2. Algoritmos de Substituição de Páginas

Um dos aspectos mais importantes em um sistema de memória virtual paginada é a política de substituição de páginas utilizada. Tal política define os critérios de seleção empregados para a retirada de uma ou mais páginas da memória principal quando outra página precisa ser alocada e não há espaço disponível.

A implementação da política de substituição se dá por meio de um algoritmo de substituição de páginas, que pode ser basicamente de dois tipos: algoritmos com espaço fixo e algoritmos com espaço variável. Algoritmos com espaço fixo trabalham sobre uma área de memória disponível sempre constante. Ao contrário, algoritmos com espaço variável modificam o tamanho da memória alocada dinamicamente, conforme o contexto de execução do sistema.

As principais propostas de algoritmos com espaço fixo encontradas na literatura são citadas e comentadas a seguir:

- *FIFO:* O critério de substituição do algoritmo FIFO – *First In, First Out* – é retirar da memória a página carregada há mais tempo, independentemente dos acessos computados enquanto esteve residente na memória. Esse algoritmo apresenta uma deficiência denominada “anomalia de Belady” (BELADY; NELSON; SHEDLER, 1969): o número de faltas de página pode eventualmente aumentar quando o tamanho da memória disponível aumenta. Todavia, sua grande vantagem é a simplicidade e a extrema facilidade de implementação. Uma fila pode ser naturalmente utilizada para ordenar as páginas de acordo com a idade: uma página sempre entra no início da fila quando é carregada e a página substituída é sempre aquela que ocupa a última posição da fila.
- *LRU:* O algoritmo LRU – *Least Recently Used* – é simples e normalmente bastante eficiente. Seu critério é substituir a página acessada há mais tempo, ou seja, a página residente menos ativa no momento da substituição. Apesar de apresentar bom desempenho na maioria das situações, o LRU também sofre algumas deficiências, comentadas no Capítulo 3. Com o objetivo de minimizar essas deficiências, variações do LRU foram recentemente propostas. Entre os novos algoritmos publicados, destacam-se o LRU-K e o 2Q. O algoritmo LRU-K (O’NEIL; O’NEIL; WEIKUM, 1993); (O’NEIL; O’NEIL; WEIKUM, 1999) não leva em consideração há quanto tempo uma página foi acessada pela última vez e sim quando ocorreu o seu K-último acesso. Por exemplo, o LRU-2 verifica qual a página que realizou seu penúltimo acesso há mais tempo, enquanto o LRU-3 trabalha com o antepenúltimo acesso e assim sucessivamente. Para isso, informações sobre as últimas K referências realizadas em cada página, residentes ou não, são mantidas pelo LRU-K. Por sua vez, o algoritmo 2Q (JOHNSON; SHASHA, 1994) trabalha com duas áreas de gerenciamento de memória: em uma delas ficam as páginas recém-carregadas e na outra permanecem as páginas acessadas enquanto estavam na primeira área. As páginas removidas da memória pelo 2Q estão normalmente na primeira área e o critério utilizado é o FIFO. Mas se a primeira área se torna muito pequena –

quase todas as páginas carregadas já foram acessadas pelo menos por duas vezes –, a substituição é feita sobre uma página da segunda área e o critério utilizado neste caso é o LRU. Ou seja, a memória é dividida em dois níveis e as páginas são alocadas em cada um de acordo com sua frequência de utilização. Em cima dessa estrutura, o algoritmo 2Q utiliza uma combinação bastante interessante dos algoritmos LRU e FIFO para substituição das páginas.

- *Ótimo:* Proposto por Belady (BELADY, 1966), o algoritmo Ótimo – ou OPT – representa a melhor solução possível a fim de se minimizar o número de faltas de página. A sua estratégia é retirar da memória, em caso de falta, a página que levar mais tempo para ser novamente referenciada. O algoritmo "conhece" o futuro, uma vez que pode ler previamente todos os acessos à memória realizados pela aplicação, o que não ocorreria em uma situação real de execução. Portanto, sua implementação prática (*online*) é impossível. Todavia, como solução teórica, é muito útil em simulações e pode ser utilizado como paradigma na análise de algoritmos, representando o caso ótimo.
- *MRU*-n*:* Este algoritmo escolhe a última página acessada, ou a *n*-última página, para ser substituída. O MRU original – *Most Recently Used* – substitui sempre a última página acessada; pode ser chamado também de MRU-1. As variações MRU-n selecionam, por sua vez, a *n*-última página mais recentemente acessada para ser substituída. Por exemplo, o algoritmo MRU-2 substitui sempre a penúltima página acessada antes de cada falta, enquanto o MRU-3 remove sempre a antepenúltima e assim sucessivamente. Obviamente, as versões MRU-*n* utilizam um critério de substituição pseudo-MRU, normalmente mais eficiente que a versão original pelo fato de explorarem, com maior ou menor grau de intensidade, as características de localidade temporal que algumas páginas apresentam.
- *Clock:* Uma fila como a do FIFO é utilizada pelo algoritmo do relógio, ou algoritmo *Clock*. As páginas são dispostas nesta fila de acordo com a ordem em que foram carregadas na memória. A página substituída

normalmente é aquela carregada há mais tempo; porém, um bit de uso é adicionado ao mecanismo de seleção: a página residente mais antiga só é substituída se o seu bit de uso estiver zerado, caso contrário a próxima página com bit zerado na sequência da fila é selecionada. Um ponteiro circular, que se desloca sequencialmente – daí o nome do algoritmo –, indica a página a ser substituída. Em cada falta, este ponteiro se desloca uma posição para frente. No entanto, se o bit da página apontada estiver acionado, este é zerado e o ponteiro continua a se deslocar até encontrar uma página inativa (bit zerado). A finalidade do bit de uso é informar se uma página foi ou não referenciada desde que foi carregada ou desde que o ponteiro de substituição a apontou pela última vez. Com esse artifício, o algoritmo não corre o risco de remover páginas acessadas recentemente, comportando-se de maneira similar ao LRU.

- *NRU:* O algoritmo NRU – *Not Recently Used* – tem como critério retirar da memória uma página não acessada recentemente, isto é, entre as duas últimas faltas. Além disso, o algoritmo verifica se cada página residente foi ou não modificada, ou seja, se teve seus dados alterados desde que foi carregada na memória. Dois *bits* são utilizados: um para representar a existência ou não de acessos recentes e outro para sinalizar se a página foi modificada desde o seu carregamento. A prioridade de substituição segue a seguinte ordem: páginas não referenciadas e não modificadas; páginas não referenciadas; páginas não modificadas; páginas referenciadas e modificadas.
- *LFU:* A página substituída pelo critério LFU – *Least Frequently Used* – é aquela menos acessada dentre as que se encontram carregadas na memória. Essa verificação é feita através de um contador de acessos (*hits*) associado a cada página residente. A contagem dos acessos é zerada sempre que uma página deixa a memória principal. Portanto, a frequência de utilização se refere apenas aos acessos recentes e não a todo o histórico de execução da aplicação. Um problema que acontece com o LFU é o seguinte: páginas recém-carregadas têm uma probabilidade de serem substituídas muito maior que as páginas mais

antigas, porque quanto antes uma página foi carregada, mais tempo ela teve para receber outros acessos. Visando solucionar o problema, foi proposto o algoritmo FBR (ROBINSON; DEVARAKONDA, 1990) – *Frequency-Based Replacement*. Tal algoritmo acrescenta um tempo de carência que toda página recebe após ser referenciada. Somente páginas fora do período de carência podem ser substituídas e novas referências ocorridas no início dessa carência não incrementam o contador de acessos da página, evitando outro vício do algoritmo LFU: a valorização de páginas muito acessadas em um único período de execução.

- *MFU:* O algoritmo MFU – *Most Frequently Used* –, por outro lado, substitui sempre a página residente mais acessada. Da mesma forma que o LFU, utiliza contadores de acesso para determinar a frequência de utilização de cada página. A deficiência desse algoritmo é que ele remove normalmente páginas com alta localidade temporal, exatamente aquelas com maior tendência a receber novos acessos.

Dentre os algoritmos com espaço variável, três propostas se destacam:

- *WS:* O critério da política WS (DENNING, 1968); (DENNING; SCHWARTZ, 1972) – *Working Set* – é exatamente o mesmo que o da política LRU quando a memória já está cheia e uma falta de página ocorre, a página acessada há mais tempo é substituída. No entanto, a atuação deste algoritmo não se restringe a substituir páginas; ele também estabelece um tempo máximo que cada página pode permanecer na memória sem ser referenciada. As páginas que atingem esse tempo máximo de inatividade são sumariamente retiradas da memória principal. O número de páginas carregadas é, portanto, variável. Esse algoritmo explora o conceito homônimo, também concebido por Denning, o qual afirma que somente um conjunto limitado de páginas é referenciado pela aplicação em um certo intervalo de tempo qualquer. Definido esse tempo – como parâmetro –, o algoritmo WS assegura que apenas as páginas acessadas no respectivo período permanecem

residentes, isto é, apenas aquelas pertencentes ao *working set* do processo no instante considerado.

- *PFF:* Como o algoritmo WS, o PFF (CHU; OPDERBECK, 1976a); (CHU; OPDERBECK, 1976b) – *Page Fault Frequency* – utiliza o critério LRU em caso de substituição quando o espaço de memória disponível está totalmente ocupado. Da mesma forma que o WS, ele retira sumariamente páginas da memória quando detecta inatividade aparente. Sua metodologia é executar esse procedimento quando a taxa de faltas está baixa, representando um indício de que há mais páginas residentes do que páginas sendo efetivamente acessadas. As páginas selecionadas para deixar a memória são todas aquelas não referenciadas no intervalo entre duas faltas de página consecutivas.
- *VMIN:* O algoritmo VMIN (PRIEVE; FABRY, 1976) representa o caso ótimo de desempenho – gera o número mínimo de faltas de página – no contexto dos algoritmos de substituição com espaço variável. Assim como o algoritmo Ótimo, ele “conhece” todos os acessos futuros que uma aplicação realiza. Sua filosofia é remover da memória as páginas cujo próximo acesso leve mais tempo para acontecer do que um parâmetro fixo pré-estabelecido. Assim, a memória gerenciada pelo VMIN dificilmente fica totalmente ocupada, pois só permanecem residentes aquelas páginas que certamente serão referenciadas em um futuro breves. Obviamente, trata-se também de um algoritmo *offline*.

Estudos demonstram que o desempenho médio dos algoritmos com espaço variável tende a ser superior ao desempenho daqueles com espaço fixo (DENNING, 1980), mas, devido à sua maior complexidade de implementação, são pouco utilizados na prática. Algumas métricas são comumente utilizadas para aferir, sob o ponto de vista de algum aspecto de desempenho, a eficiência dos vários algoritmos de substituição de páginas existentes (de ambos os tipos). Podemos citar como métricas potencialmente relevantes em estudos de desempenho:

- *Número de faltas de página:* quantidade de faltas de página computadas durante a execução de um processo;

- *Taxa de faltas de página:* taxa percentual de faltas em relação ao total de acessos realizados pelo processo num certo intervalo de tempo;
- *Produto espaço X tempo:* representa uma forma de quantificar a ocupação real da memória pelo processo, levando em conta seu tempo de execução.

Métricas de desempenho refletem o comportamento das aplicações sob diferentes visões. Em um nível mais profundo, estão diretamente associadas ao grau de ocorrência de uma série de fatores, muitas vezes determinantes para o desempenho da aplicação. Dentre esses fatores, pode-se destacar:

- *Localidade de referências:* Ocorre quando há uma concentração dos acessos à memória em determinadas páginas. A localidade pode ser temporal e/ou espacial. Um caso de localidade temporal é identificado quando um conjunto de páginas, de tamanho qualquer não nulo, é continuamente referenciado por um período de tempo considerável. Um caso de localidade espacial é identificado quando dados próximos entre si, em termos de endereçamento virtual, são acessados continuamente. Ou seja, muitos dados de uma ou mais páginas são acessados em um mesmo período.
- *Padrão de acessos à memória:* Designa uma regularidade na forma como os acessos à memória acontecem durante um período de execução do sistema. Essa regularidade também pode se dar em termos espaciais e/ou temporais. A sequência em que determinadas páginas são acessadas, por exemplo, pode constituir uma regularidade espacial. Por outro lado, o tempo que decorre entre dois acessos consecutivos a uma mesma página, se relativamente estável, representa uma regularidade temporal. A verificação sequencial de todos os elementos de uma matriz é um exemplo prático de padrão de acessos à memória cuja regularidade é espacial. Se a média dos elementos da matriz está

sendo calculada, a página que armazena esta média deve ser constantemente atualizada, representando um caso de regularidade temporal. Por sua vez, *loops* constituem padrões de acesso em que tanto a regularidade espacial como a temporal estão sempre presentes.

- *Características especiais da aplicação:* O tamanho de uma aplicação, por exemplo, pode influenciar o desempenho do sistema de memória. Aplicações muito pequenas atingem ótimos desempenhos se o seu espaço de endereçamento virtual puder ser totalmente carregado na memória principal. Aplicações muito grandes, por outro lado, geram normalmente um número de faltas de página muito alto em sua execução, exceto se houver uma exploração eficiente do conceito de localidade, fato que depende da maneira como a aplicação foi construída. Aplicações paralelas ou distribuídas também podem alterar o desempenho do sistema de memória. O grau de localidade apresentado por essas aplicações pode ser diferente daquele apresentado pelas aplicações tradicionais, fazendo com que certas premissas de comportamento – em termos de utilização da memória – às vezes não se confirmem.

2.2.1. Algoritmos Adaptativos

Os algoritmos adaptativos de substituição atuam de forma dinâmica, adaptando seu comportamento de acordo com o padrão de acesso à memória em tempo de execução. Para tanto, podem ser utilizadas várias técnicas adaptativas, como, por exemplo, modificar o critério de substituição, o tamanho da memória utilizada ou os parâmetros de controle do algoritmo.

O princípio da adaptação é encontrar um padrão a partir de uma informação coletada sobre as páginas usadas recentemente e então adequar o algoritmo ao comportamento destes acessos à memória. A ideia dos algoritmos adaptativos de substituição não é nova: o Atlas loop detector

(BAYLIS; FLETCHER; HOWARTH, 1968), predecessor dos mais recentes detectores de *looping* usados nos algoritmos SEQ (GLASS; CAO, 1997) e EELRU (SMARAGDAKIS; KAPLAN; WILSON, 1999), já poderia ser considerado uma política que se adapta ao comportamento do sistema.

Existem alguns padrões bastante explorados por esses algoritmos, tais como a frequência de acesso às páginas (localidade temporal), a recência (localidade espacial), a probabilidade de acesso individual, a previsão de acesso futuro baseado no reuso, os acessos sequenciais e os loopings. As estratégias utilizadas pelos algoritmos variam. Nos últimos doze anos, porém, surgiram várias propostas baseadas apenas na mistura de frequência e recência, tais como LIRS (JIANG; ZHANG, 2002), LRFU (LEE et al, 2001) e ARC (MEGIDDO; MODHA, 2003).

Pesquisas recentes nesta área utilizam ferramentas da inteligência artificial para auxiliar a adaptação. Exemplos são os sistemas baseados em regras e lógica *fuzzy*. Quando uma política de substituição de página utiliza mais de um parâmetro para a decisão, achar a relação entre os parâmetros e combiná-los torna-se um problema. Assim, uma forma de combinar esses parâmetros é modelá-los para a utilização de lógica *fuzzy* (SABEGHIL; YAGHMAEE, 2006).

O LIRS – *Low Inter-reference Recency Set* – tem como objetivo 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. Em outras palavras, o algoritmo LIRS utiliza uma fonte de informação adicional: além do grau de recência atual das páginas – isto é, que páginas foram acessadas mais recentemente que outras –, ele leva em conta também a recência observada entre os dois últimos acessos realizados em cada página. Isso significa que o LIRS não substitui necessariamente a página acessada há mais tempo, mas sim que ele utiliza o histórico recente desta

informação para prever quais páginas têm maiores probabilidades de acesso em um futuro breve.

A política adaptativa de substituição de páginas ARC (MEGIDDO; MODHA, 2003) – *Adaptive Replacement Cache* – divide a memória em duas áreas de gerenciamento, uma englobando as páginas recém-carregadas e outra armazenando aquelas que receberam mais de um acesso. Entretanto, a substituição é feita sempre utilizando o critério LRU em uma das duas áreas. A seleção da área se dá de forma balanceada, de acordo com o estado em que cada uma delas se encontra nos momentos de substituição.

3. ADAPTABILIDADE, TÉCNICAS E INFORMAÇÕES

O Capítulo que ora apresentamos detalha o funcionamento do algoritmo LRU-WAR. Descreve também a importância do tratamento da frequência de acessos, a qual é explorada no algoritmo LRU-WARlock, como informação importante de escolha para a substituição de páginas. Discute o uso do modelo n-grama, sua importância em processamento estatístico de linguagem natural e sua adequação em sistemas de gerência de memória. Por último, e não menos importante, as técnicas para auxílio à política de substituição de páginas, como busca antecipada de páginas e *profiling*, são detalhadas.

3.1. Algoritmo adaptativo LRU-WAR

O princípio de funcionamento do algoritmo LRU-WAR é baseado na estimativa e na análise da variação do tamanho máximo do *working set* do programa. Tal estimativa é validada entre faltas de página consecutivas. Ou seja, a dimensão máxima do *working set* temporário é usado como fator decisivo na adaptabilidade do LRU-WAR.

O LRU-WAR monitora os acessos à memória e verifica, entre duas faltas de página consecutivas, a página referenciada com menor recência em relação ao seu último acesso, ou seja, identifica a posição mais alta da fila LRU que recebeu acessos neste período. Tal posição W limita uma porção da fila LRU, conhecida como área de trabalho (*working area*).

Há três estados de execução definidos: tendência LRU, tendência sequencial e operação sequencial. Quando a área de trabalho é maior que o parâmetro L (tamanho da região sequencial), considera-se que o programa está reutilizando as suas páginas em memória e então fica no estado de tendência LRU. Neste estado, o critério LRU é adotado, e permanece até que seja detectada uma tendência de acessos sequencial.

Quando, entre duas faltas de página, o tamanho da área de trabalho se reduz, ficando menor que L, o algoritmo entra em tendência sequencial,

pois subentende-se que está ocorrendo um número adicional de faltas de página sem que haja uma boa reutilização das páginas em memória. Nesse estado ainda se adota o algoritmo LRU para a substituição de páginas.

Se a tendência sequencial se mantiver por um número de faltas de página proporcional ao tamanho da área de trabalho, desde o instante em que esta tendência começou, o LRU-WAR muda para o estado de operação sequencial, onde se adota o algoritmo MRU-n para substituição. Para definir qual página substituir neste estado, o algoritmo define outro parâmetro: a carência mínima C. Ela define uma região inicial da fila LRU chamada região protegida. Esta região contém páginas que não podem ser removidas da memória, assim, a menor área de trabalho aceita pelo LRU-WAR é C+1. A figura 4 ilustra as divisões lógicas da fila LRU que orientam o algoritmo. O parâmetro C também é usado na determinação do número de faltas de página para o algoritmo passar do estado de tendência sequencial para operação sequencial, com o objetivo de evitar que o algoritmo detecte erroneamente um padrão sequencial de acessos.

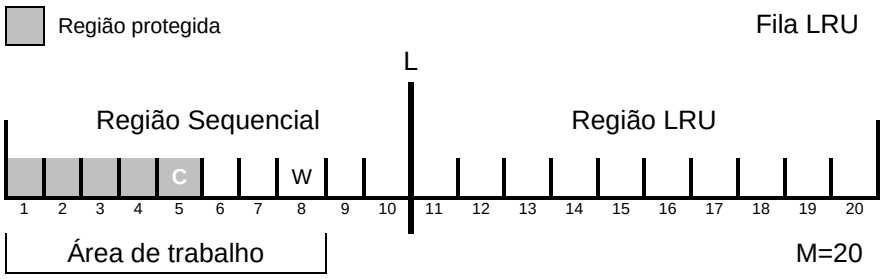


Fig. 4: Divisão lógica da fila LRU utilizada pelo algoritmo LRU-WAR

O valor padrão para C é igual a 5 e o parâmetro L é especificado como o MIN(50, M/2), onde M é o tamanho total da memória em número de páginas.

A Figura 5 mostra dois exemplos de modo operação do algoritmo LRU-WAR. O modo de funcionamento operação sequencial, é ilustrado na parte superior da figura. Esse modo substitui a página que está na posição descrita como +1, por ser a W+1. Note que a definição da área de trabalho a cada falta de página é feita sobre a posição da fila LRU. Em outras palavras,

o W não é a última página acessada e sim a maior posição na fila LRU acessada entre as duas últimas faltas de páginas. A posição é marcada e a página daquela posição vai para a primeira posição da fila. O modo de funcionamento tendência original está na parte inferior da figura. Esse modo substitui a página na última posição da fila. Isso ocorre sempre quando entre duas faltas de páginas existiu uma página referenciada que estava nas posições depois do parâmetro L, neste caso entre as posições 11 e 20.

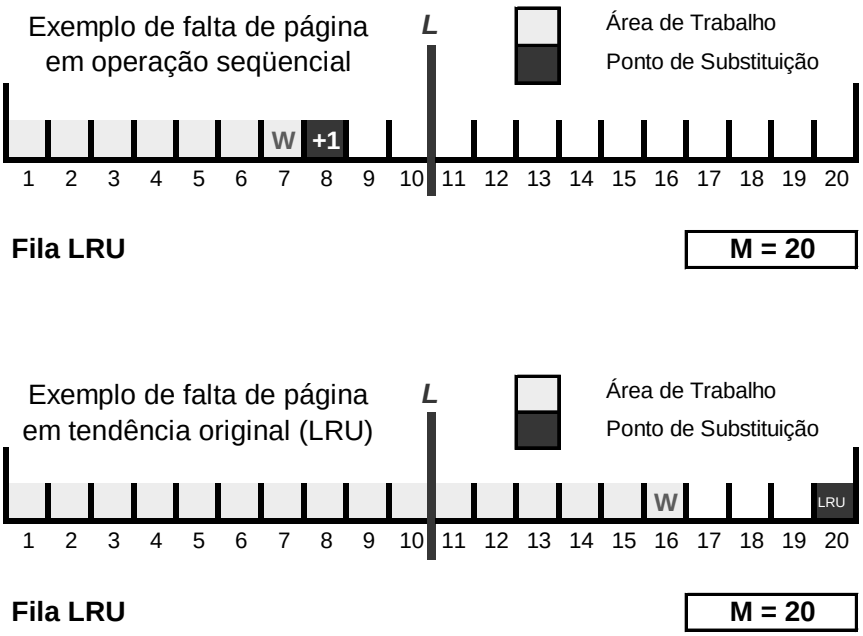


Fig. 5: Exemplo de operações do LRU-WAR

O LRU-WAR não trata a frequência com que as páginas são acessadas. Isso pode onerar seu desempenho, já que páginas frequentemente acessadas podem deixar a memória ocasionando faltas de páginas. O estudo sobre essa deficiência do algoritmo LRU-WAR é um tópico fundamental, e será discutido na próxima seção.

3.2. Estudo de Frequência de Acessos

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.

O principal objetivo da proposta apresentada neste trabalho é fazer com que o LRU-WAR 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 sequências.

O princípio do funcionamento de LRU-WARlock surgiu da idéia de separar os acessos diferentes do padrão de acesso sequencial e tratá-los de outra maneira. A questão crucial foi descobrir o que mais atrapalhava a detecção de referências sequenciais. 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. Nesse cenário, com vários *working sets* 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 em tal cenário seria remover as páginas dos *working sets* da detecção do padrão sequencial do LRU-WAR, isto é, separar as páginas acessadas mais frequentemente.

O mecanismo utilizado pelo LRU-WARlock é a reserva de parte da memória para as páginas com a maior frequência 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 frequência de acesso, e pode ser controlado diretamente pelo sistema operacional.

Com relação à determinação das páginas de maior frequência, 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), que localiza quais partes da aplicação são mais frequentemente executadas, direcionando assim o foco da otimização.

Outro aspecto importante, além da frequência, são as relações entre as páginas. É possível, por meio dessas relações, prever qual será a próxima página a ser acessada e, assim, utilizando uma técnica adequada, evitar a futura falta de páginas. Na sessão que segue, será apresentada uma técnica no auxilio da descoberta dessas relações entre as páginas, o n-grama, e como ele pode ser aplicado em gerência de memória.

3.3. Estudo dos n-gramas Aplicados à Gerência de Memória

Os n-gramas são tipos de modelos probabilísticos para prever o próximo elemento em uma sequência. Eles são usados em várias aplicações de processamento estatístico de linguagem natural e em Biologia, especificamente, em Genética (MANNING; SCHUTZE, 1999). Dada uma sequência, um n-grama é uma sub-sequência de n elementos. Os elementos podem ser letras, palavras, fonemas, nucleotídeos ou, como ocorre neste trabalho, páginas do sistema de memória virtual. Um n-grama de tamanho 1 é chamado de unigrama, de tamanho 2 é conhecido como bigrama, tamanho 3 é trigrama e tamanho 4 ou mais é nomeado n-grama.

A base matemática dos n-gramas foi proposta por Markov em 1913. O problema de como prever o próximo elemento foi muito estudado e é conhecido como Shannon Game (SHANNON, 1955). O jogo é simples: dada uma sequência de letras, qual seria provavelmente a próxima letra?

Exemplo: “prova de geo...”. De uma coleção de dados para treinamento de um programa que dará a resposta, o próprio programa pode retirar a distribuição probabilística para a próxima letra onde as probabilidades de todas as letras somadas seja 1,0. (exemplo: a=0,0001, b=0,0002, ..., g=0,8, ...).

A Tabela 1 mostra os bigramas mais comuns encontrados no jornal The New York Times de agosto a novembro de 1990 (MANNING; SCHUTZE, 1999). É possível, por meio deste corpus obtido com o jornal, aplicar os bigramas a um programa de reconhecimento de escrita para melhorar sua precisão. Um modelo de trigrama foi usado no sistema de reconhecimento de voz chamado IBM TANGORA já na década de 70 (JURAFSKY; MARTIN, 2008).

Tab. 1: Bigramas mais comuns no *New York Times*

Frequência	**Palavra 1**	**Palavra 2**
80871	of	the
58841	in	the
26430	to	the
21842	on	the
21839	for	the
18568	and	the
16121	that	the
15630	at	the

Pretendemos utilizar o modelo de bigrama para trazer à memória principal, além da página responsável pela falta, a possível próxima página a ser referenciada.

A técnica de substituição antecipada de páginas (*prefetching*) é uma excelente candidata a ser usada em conjunto com o modelo de bigrama, e será comentada na seção seguinte juntamente com a técnica de *profiling*.

3.4. Técnicas de *Profiling* e *Prefetching*

Grande parte dos sistemas de gerência de memória utiliza a técnica de paginação por demanda, onde somente a página responsável pela falta é trazida para a memória. No caso da busca antecipada de páginas ou pré-paginação (*prefetching*), não somente a página que sofreu a falta pode ser movida para a memória, mas outras quaisquer de acordo com a estratégia escolhida, tentando prever quais poderão ser referenciadas em um futuro próximo.

No contexto dos algoritmos adaptativos de substituição de páginas, toda vez que um programa é executado, é necessário que o algoritmo se adapte as referências à memória na execução atual do programa. Isso tem um custo relativo alto, até que a adaptação ocorra. No meio da execução do programa, podem ocorrer outros padrões de acessos, e o algoritmo levará um tempo até se adaptar ao novo padrão encontrado. Uma alternativa para evitar o tempo de adaptação e faltas de páginas desnecessárias, seria utilizar a técnica de *profiling*. Essa técnica possibilita capturar informações em tempo de execução do programa, e usar esse conhecimento em suas próximas execuções.

No sistema operacional Windows Vista, da Microsoft, foi criada uma tecnologia denominada por eles *SuperFetch*. O *SuperFetch* utiliza, ao mesmo tempo, *prefetching* e *profiling*; ele reconhece o programa de uma execução anterior e, através da técnica de *profiling*, grava o padrão de referencias as páginas. Nesse momento, o sistema operacional tenta pré-paginar a maior parte das páginas necessárias.

4. TÉCNICAS E INFORMAÇÕES NA GERÊNCIA DE MEMÓRIA

Um estudo mais aprofundado sobre o algoritmo LRU-WAR é apresentado inicialmente neste capítulo. Após tal estudo, o algoritmo LRU-WARlock é proposto para eliminar algumas deficiências do LRU-WAR. Ele utiliza a técnica de *profiling* para coletar a informação de frequência de acesso às páginas, sem modificar o comportamento de seu algoritmo base. Também é analisada aqui outra técnica que pode ser usada para aumentar o desempenho da gerência da memória, e outro algoritmo é criado para demonstrá-la, o LRU+ng.

4.1. Ajuste nos Parâmetros de Controle do LRU-WAR

O algoritmo LRU-WAR se caracteriza na adaptação de seu funcionamento com base no padrão de acessos à memória dos programas e em alguns parâmetros de controle. Os valores desses parâmetros de controle foram determinados heuristicamente e são fixos durante a operação do algoritmo. Esta seção procura estudar o comportamento e o respectivo desempenho do algoritmo em relação à variação destes parâmetros de controle (MIDORIKAWA; PIANTOLA; CASSETTARI, 2007). Os resultados obtidos mostram que um controle dinâmico desses parâmetros pode trazer benefícios interessantes na execução de certas aplicações.

Iniciamos com a descrição dos programas usados para a avaliação e, em seguida, com a apresentação e análise dos resultados preliminares obtidos.

4.1.1. Configuração dos Testes

A avaliação dos algoritmos foi executada usando-se três diferentes *traces* de programas, a saber, gnuplot, grobner e sprite (Tabela 2). Estes foram escolhidos com base no seu padrão de acessos e desempenho dos diversos

algoritmos de substituição de páginas estudados. As simulações foram realizadas com as ferramentas disponibilizadas pelo ambiente Elephantools.

Tab. 2: Descrição dos *traces* gnuplot, grobner e sprite

Trace	**Descrição**	**Origem**	**Total de páginas**
gnuplot	Trace relativo à geração de um gráfico em *postscript*.	VMTrace	7718
grobner	Programa para reorganização de fórmulas baseado em funções base de Grobner.	VMTrace	67
sprite	Proveniente do sistema de arquivos de rede Sprite. Contém requisições a um servidor de arquivos a partir de várias estações de trabalho cliente em um período de dois dias.	LIRS	7075

O programa gnuplot se caracteriza por apresentar dois padrões de acessos bem definidos: um conjunto de páginas com forte localidade temporal ao longo da execução (com endereços virtuais baixos) e um padrão de acessos sequencial devido à execução de laços que percorrem três vezes quase todo o espaço de endereçamento virtual (figura 6). Essa característica faz com que o algoritmo LRU apresente um desempenho “em forma de degrau” porque, devido ao fato deste não ser adequado a este padrão de acessos sequencial, o algoritmo sempre faz uma escolha ruim, até que o espaço de memória disponível seja igual ou maior ao tamanho total do programa (figura 6). O gnuplot apresenta o padrão de acessos mais adequado ao LRU-WAR, que apresenta um desempenho muito próximo do algoritmo Ótimo.

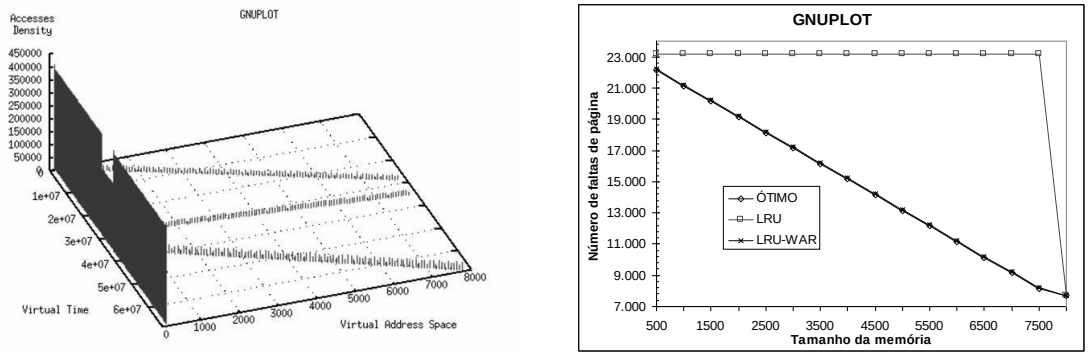


Fig. 6: Características do programa gnuplot. Padrão de acessos à memória à esquerda e desempenho do LRU-WAR à direita

O programa grobner também apresenta um padrão de acessos sequencial bem definido, como mostrado na figura 7. Contudo, ao contrário do gnuplot, esse padrão sequencial está intercalado com outros padrões de acesso à memória. Entre os outros padrões, observa-se um com acessos a poucas páginas com forte localidade temporal. Essa característica faz com que o LRU-WAR apresente um ganho modesto de desempenho em relação ao LRU, não chegando a mais de 12% para tamanho de memória igual a 28 páginas (figura 7). Esse programa foi escolhido, pois se trata de um caso em que o LRU-WAR poderia explorar melhor o padrão sequencial presente.

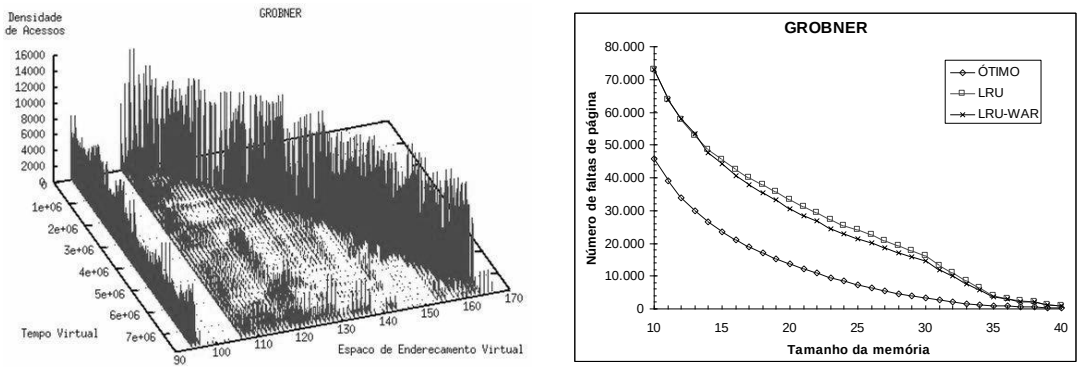
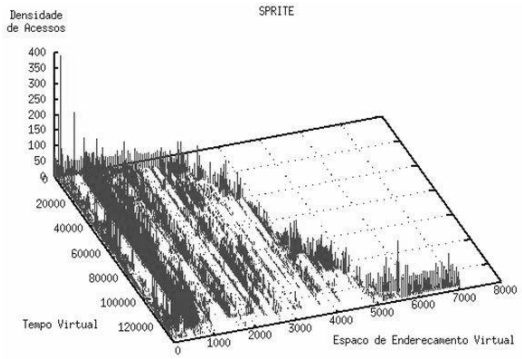


Fig. 7: Características do programa grobner. Padrão de acessos à memória à esquerda e desempenho do LRU-WAR à direita

O terceiro programa escolhido foi o sprite. O padrão de acessos observado para este programa mostra um grande conjunto de páginas que são acessadas com uma certa frequência, mas sem apresentar um padrão destacado e com intervalos irregulares (figura 8). O LRU-WAR comete decisões equivocadas, cujas consequências são amenizadas pelo

mecanismo de detecção de erros. Contudo, o desempenho final do algoritmo é pior que o LRU, chegando a cerca de 7% para tamanho de memória igual a 2200 páginas (figura 8).



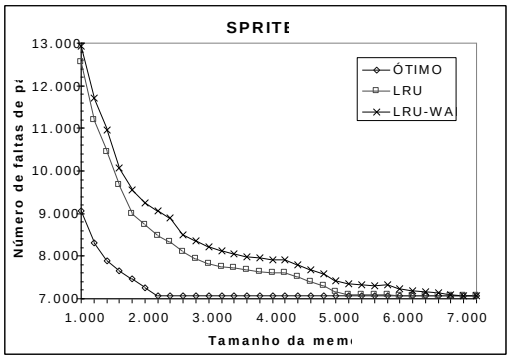


Fig. 8: Características do programa sprite. Padrão de acessos à memória à esquerda e desempenho do LRU-WAR à direita

A seguir apresentamos os resultados das análises da variação de desempenho do LRU-WAR com a modificação dos valores dos seus parâmetros de controle C e L.

4.1.2. Resultados Experimentais e Análises

Para cumprir os objetivos definidos neste estudo, apresentamos aqui os resultados e análises de desempenho do algoritmo LRU-WAR com a variação dos seus parâmetros de controle.

- Gnuplot:

O LRU-WAR é um algoritmo que alcança excelente desempenho quando um programa apresenta padrões de acessos sequenciais predominantes. Análises anteriores já demonstraram este comportamento (CASSETTARI; MIDORIKAWA, 2004). Como o programa gnuplot apresenta acessos sequenciais do início ao fim, o LRU-WAR apresentou desempenho muito próximo do algoritmo Ótimo: apenas uma diferença constante em relação ao número de faltas de página.

Apesar do excelente desempenho alcançado pelo LRU-WAR, foi conduzido um experimento com a alteração dos valores dos parâmetros C e L do algoritmo (figura 9).

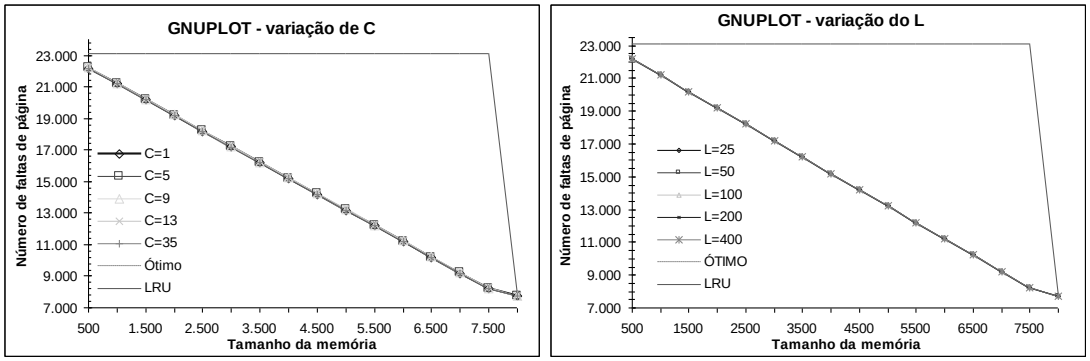


Fig. 9: Variação dos parâmetros para o gnuplot. Carência mínima C à esquerda e Tamanho da região sequencial L à direita

Como pode ser observado, o desempenho do LRU-WAR permaneceu bem próximo do algoritmo Ótimo, mesmo com uma variação ampla dos parâmetros. Apesar disso, foi possível obter ganhos modestos em relação aos valores padrão: para C=35 e C=1, obtivemos um número menor de faltas de página de 23 e 21, respectivamente, o que fez com que ficássemos, respectivamente, a uma distância de apenas 42 e 44 faltas de páginas do algoritmo Ótimo de um total de 68.458.509 acessos à memória.

O estudo realizado do efeito da variação da carência mínima C mostrou que, para valores maiores que 50, o algoritmo LRU-WAR se comportou como o algoritmo LRU para o gnuplot. A razão desse efeito é que quanto maior a carência, maior é o tamanho da região protegida. Assim, o algoritmo LRU-WAR, mesmo em operação sequencial, irá substituir uma página MRU em uma posição mais alta da fila (W+1, com W»C), o que é próximo da página que seria substituída caso estivesse usando a algoritmo LRU.

As variações do L não surtiram efeito para esse programa e o LRU-WAR apresentou o mesmo desempenho para os valores analisados (figura 9). Uma razão para esse resultado que, para o gnuplot, o LRU-WAR normalmente está no estado de operação sequencial. Uma variação de L

somente muda o momento em que há a mudança do estado de tendência LRU para a operação sequencial, resultando em um comportamento bastante similar em todos os casos.

- Grobner:

O programa grobner se caracteriza por possuir trechos com padrão sequencial intercalado com referências a muitas outras páginas, utilizando quase todas as páginas do seu espaço de endereçamento virtual. O desempenho do LRU-WAR com parâmetros padrão é ligeiramente melhor que o do LRU (figura 7). Dessa forma, algoritmos que procuram detectar padrões de acessos sequenciais podem obter bons resultados.

Como o Grobner possui apenas 67 páginas referenciadas, neste estudo, vamos desconsiderar tamanhos de memória muito pequenos (menores que 10) e muito grandes (maiores que 67) pelo fato de não agregar informações relevantes para as comparações.

Ao analisarmos os efeitos da variação da carência mínima C, verificamos que quanto menor o valor de C, o LRU-WAR detecta mais rapidamente os trechos sequenciais do grobner e, por consequência, melhor o seu desempenho (figura 10). A razão para este comportamento é que, depois de estar em tendência sequencial, quanto menor a carência mínima, mais rapidamente o algoritmo entra em operação sequencial.

Quando C=1, o LRU-WAR supera o LRU em todos os tamanhos de memória até 44 páginas, quando então as faltas de página se convergem para todos os valores de C estudados (figura 10). O maior ganho de desempenho foi obtido para uma memória de 35 páginas, com um ganho de quase 21% em relação ao LRU. Em comparação com os valores padrão do LRU-WAR, uma carência C=1 também apresenta desempenho melhor, com um ganho de até 13% para uma memória de 38 páginas.

Para C=9 e C=13, o LRU-WAR apresenta uma tendência de comportamento para o LRU, porém ainda obtendo resultados satisfatórios em alguns tamanhos de memória.

Para as simulações para estudo do parâmetro L, foram definidos valores fixos do tamanho da região sequencial até o valor 10, devido ao

tamanho total de páginas do programa. Os resultados são apresentados na figura 10.

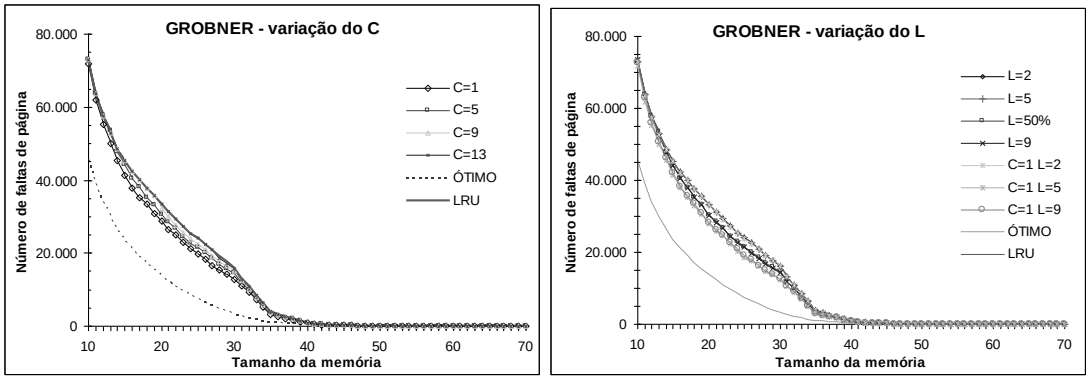


Fig. 10: Variação dos parâmetros para o grobner. Carência mínima
C à esquerda e Tamanho da região sequencial L à direita

Quando L=9, o LRU-WAR superou o LRU em todos os tamanhos de memória até 43 páginas, exceto quando M=10. Obteve o maior ganho de desempenho para M=35, superando o LRU em 13,2%. De uma maneira geral, valores baixos de L não apresentaram melhor desempenho que o LRU. Esse comportamento pode ser explicado pelo fato do padrão de acessos predominante do grobner estar intercalado com outros padrões. Assim, um valor muito pequeno de L levaria o LRU-WAR a entrar em operação sequencial de forma equivocada.

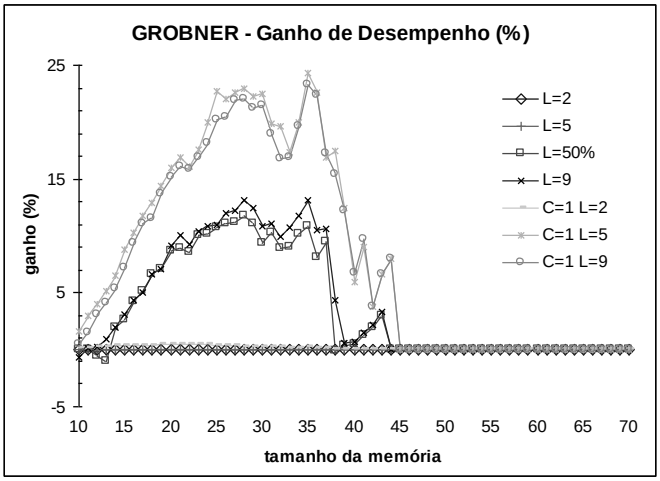


Fig. 11: Ganho de desempenho do LRU-WAR no grobner em relação ao LRU

Acima, dados os ganhos de desempenho obtidos com a variação individual de C e L, resolvemos testar a variação conjunta de ambos os parâmetros. O melhor resultado obtido foi com C=1 e L=5, com um ganho máximo de quase 24% em relação ao LRU para M=35 páginas. Esse valor representou um ganho de mais de 15% em relação aos valores padrão do LRU-WAR (figura 11). Tais resultados mostram que é possível otimizar o desempenho do LRU-WAR, de acordo com o padrão de acessos do programa.

- Sprite:

O programa sprite tem baixa localidade temporal e acessa um grande conjunto de páginas com alta frequência. Nesse caso, o algoritmo LRU-WAR toma decisões equivocadas entrando muitas vezes em operação sequencial, e assim apresenta desempenho pior que o LRU. A frequência de acessos a vários conjuntos de páginas faz com que a área de trabalho do LRU-WAR diminua, o que faz o algoritmo entrar em tendência sequencial e, posteriormente, em operação sequencial. Como a localidade temporal não é uniforme, páginas que serão acessadas em um futuro próximo são logo descartadas. O LRU-WAR com valores padrão para os parâmetros de controle teve um desempenho médio pior de cerca de 4% em relação ao LRU (figura 7.b).

A análise da variação de C mostrou que quanto maior o valor da carência mínima C, o desempenho apresentado pelo LRU-WAR fica mais próximo do LRU (figura 10.a). Nos testes realizados, o melhor desempenho foi obtido para C=44, onde o LRU-WAR igualou os resultados do LRU.

Uma explicação para esse resultado é que um valor alto de C faz com que o LRU-WAR diminua a frequência com que entra em operação sequencial, assim permanecendo mais tempo em tendência LRU. Além disso, um valor alto de C faz com que a região protegida contenha páginas que saíram recentemente do working set e que podem ser reutilizadas em breve.

A análise da variação do comportamento do LRU-WAR com o tamanho da região sequencial L mostra que à medida que aumentamos este

tamanho, melhor é o desempenho observado (figura 12). Contudo, mesmo com L=800, o LRU-WAR não igualou o desempenho do LRU.

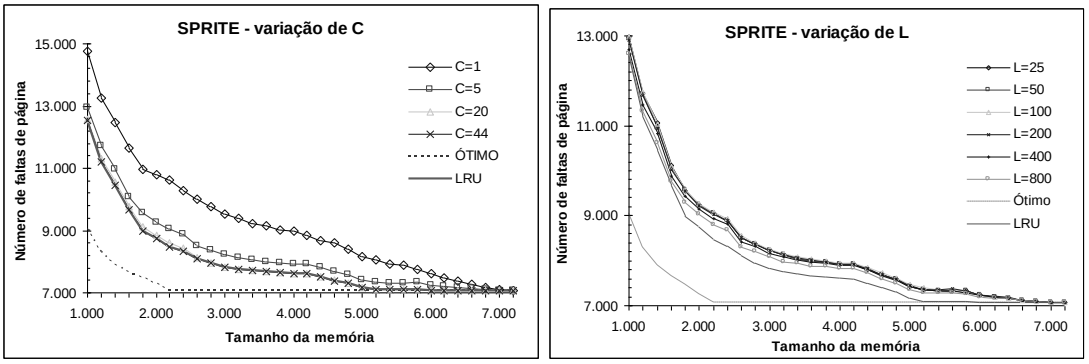


Fig. 12: Variação dos parâmetros para o sprite. Carência mínima C à esquerda e Tamanho da região sequencial L à direita

Esse comportamento pode ser explicado pelo fato de que um valor muito alto de L faz com que o algoritmo demore mais para sair do modo de operação sequencial. Dessa forma, a área de trabalho irá conter muitas páginas, ou seja, apresentando um valor relativamente alto para W. Então, embora o algoritmo adote a política de substituição MRU-n, como o ponto de substituição é definido pelo valor de W, a página selecionada para substituição será uma página próxima ao extremo LRU da fila LRU.

4.2. Frequência de Acessos e o Algoritmo LRU-WARlock

Mesmo que algum mecanismo que utilize uma técnica apropriada de Inteligência Artificial seja incluída, para ajustar os parâmetros do LRU-WAR, seu desempenho ainda será limitado. Isso acontece por dois principais fatores: primeiro que a decisão está relacionada com apenas duas páginas a cada instante, ou seja, se a memória tem 100 páginas, o algoritmo deverá escolher entre somente duas, a página que está na posição MRU-n ou na LRU. Apenas 2% de autonomia na decisão. O segundo fator é que mesmo se a escolha seja sempre perfeita, o LRU-WAR não trata de nenhuma maneira a frequência com que as páginas são acessadas. O algoritmo LRU-

WARlock foi criado para ter as mesmas características do LRU-WAR e tratar a freqüência em que as páginas são acessadas.

4.2.1. Descrição do Algoritmo

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 (PIANTOLA; MIDORIKAWA, 2008).

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 sequencial e operação sequencial. Na operação padrão, tendência LRU, o critério adotado para a substituição de páginas é o LRU, o mesmo se aplica à tendência sequencial. Quando são detectados acessos sequenciais, o algoritmo entra em operação sequencial, 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 13).

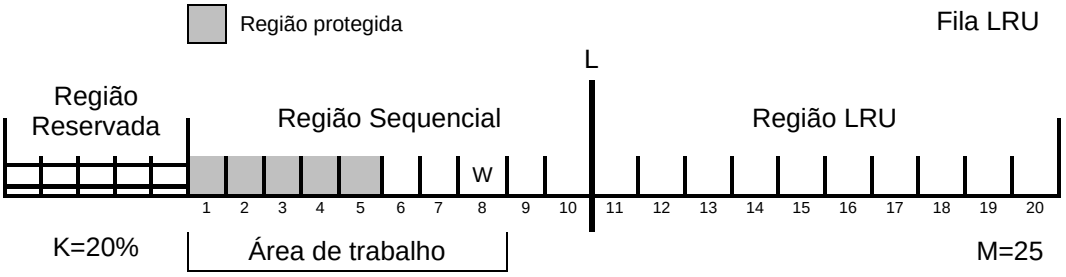


Fig. 13: Divisão lógica do algoritmo LRU-WARlock

A nova região da divisão lógica do algoritmo, a Região Reservada, é composta pelas páginas referenciadas com maior frequência, 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 14 apresenta o pseudocódigo do algoritmo, onde são destacados os acréscimos ao código original do LRU-WAR.

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. Nesse 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.

```
1. Se a página referenciada está na memória:
2.      Se o bit “page locked” for igual a zero:
3.              Se está em Operação Sequencial:
4.                      Termina a Operação Sequencial
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 Sequencial
11.                     Remove a página na posição (Área de Trabalho + 1) da
fila;
12.             Senão, se estiver em Tendência Sequencial:
13.                     Se a Área de Trabalho for menor que L e excedeu a
carência
14.                             Entrar em Operação Sequencial
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 sequencial
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
```

Fig. 14: Pseudocódigo do algoritmo LRU-WARlock

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.

4.2.2. Configuração dos Testes

Apresentamos a composição dos traces utilizados nos testes, além de uma breve descrição dos padrões de acesso e seu comportamento com alguns algoritmos de substituição apresentados na literatura como o LRU e o LRU-WAR, entre outros.

Tab. 3: Descrição dos *traces* multi1, multi2 e multi3

Trace	Descrição	Origem	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 14). O cpp é o pré-processador do GNU C, durante cuja execução podem ser observados blocos de referências sequenciais 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 sequencial. Esse padrão de acesso prejudica muito o desempenho do algoritmo LRU e isso acontece até que o espaço de memória disponível seja maior ou igual ao tamanho total das duas aplicações (Figura 15). Os programas individuais apresentam um padrão de acessos adequado ao LRU-WAR, porém como estão intercalados pela multiprogramação, diminuem consideravelmente o desempenho do algoritmo.

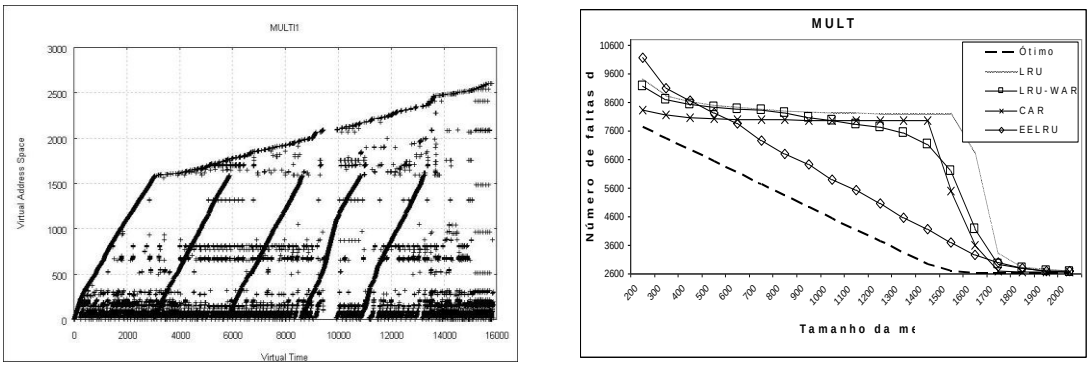


Fig. 15: Característica do *trace* multi1. Padrão de acesso à memória à esquerda e Desempenho de alguns algoritmos à direita

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, que apresenta um padrão de acessos sequencial e *looping* com períodos não-constantes. A Figura 16 mostra o padrão de acessos do *trace* multi2 e o desempenho de vários algoritmos, de modo geral, a maioria dos algoritmos apresenta um desempenho muito próximo ao LRU.

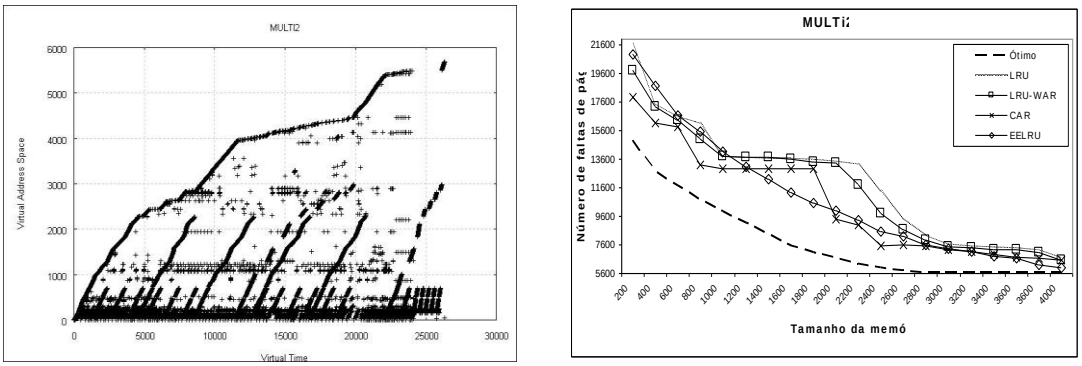
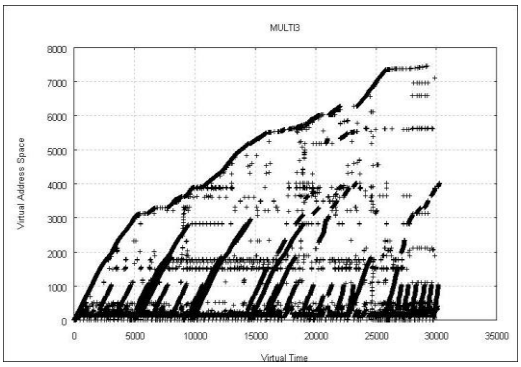


Fig. 16: Característica do *trace* multi2. Padrão de acesso à memória à esquerda e Desempenho de alguns algoritmos à direita

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 e seu padrão de referência à memória é bem diverso. Já o gnuplot tem um padrão de acessos sequenciais bem definido. O gnuplot é um programa interativo de plotagem gráfica. É possível observar (Figura 17) que com o

aumento da quantidade de programas contidos no multi3, a diferença de desempenho entre o algoritmo LRU e LRU-WAR diminui.



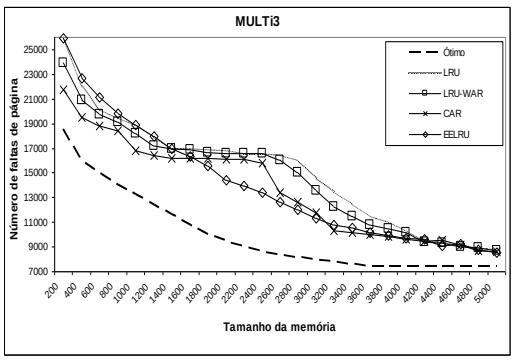


Fig. 17: Característica do *trace* multi3. Padrão de acesso à memória à esquerda e Desempenho de alguns algoritmos à direita

4.2.3. Resultados Experimentais e Análises

Apresentamos nesta seção, 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 sequenciais, o algoritmo LRU-WAR atinge um excelente desempenho. 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 sequenciais (Figura 18).

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 frequência de acessos às páginas mostra que somente 7,5% das páginas são responsáveis por 50% dos acessos à memória (Figura 18). 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.

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 18).

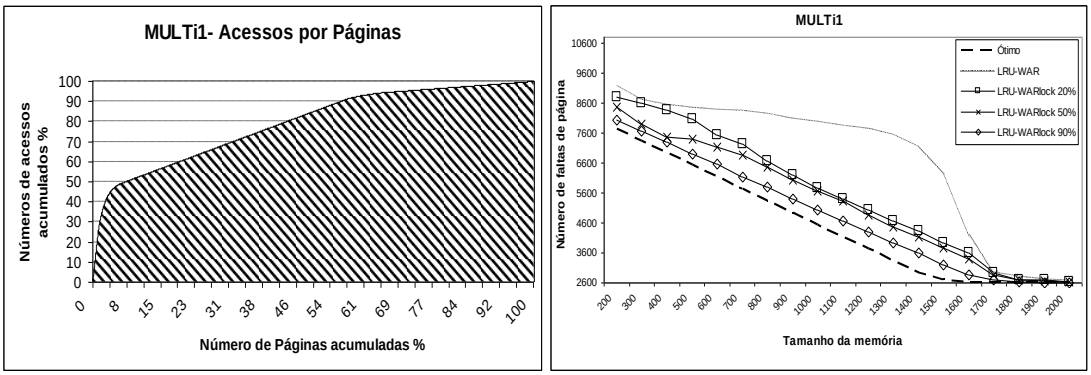


Fig. 18: Gráficos do *trace* multi1. Páginas versus Acessos à esquerda e Variação do parâmetro K à direita

Um fato importante que foi observado na análise dos traces é o tempo em que o LRU-WARlock fica no modo de operação sequencial. Em todos os casos estudados, o período em que o algoritmo trabalha com deteção de acessos sequenciais aumentou. Para o multi1, o algoritmo LRU-WAR ficou

em torno de 11% do tempo em operação sequencial. 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 sequencial. Este resultado mostra que com essa estratégia fica mais fácil detectar acessos sequenciais 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. A estratégia do LRU-WARlock também garante bom desempenho para este trace (Figura 19). Desse modo, algoritmos que utilizam técnicas para separar padrões podem obter bons resultados.

A análise do perfil de frequência 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 19).

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 19). 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 frequentes, 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.

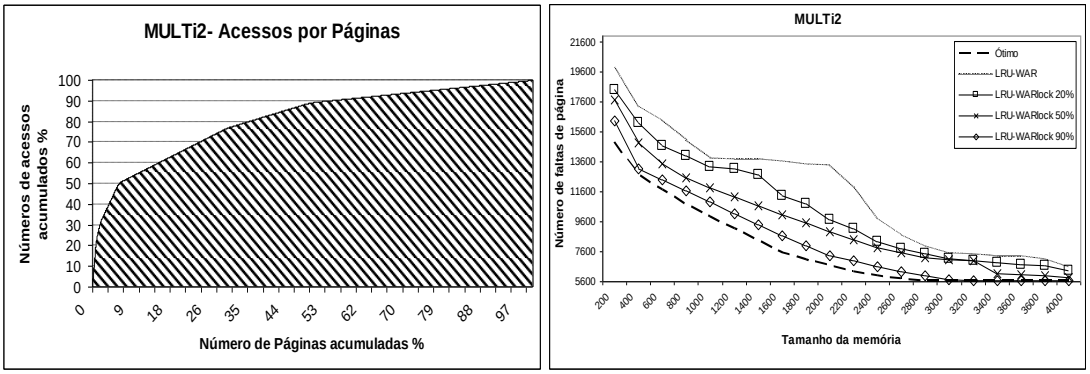


Fig. 19: Gráficos do *trace* multi2. Páginas versus Acessos à esquerda e Variação do parâmetro K à direita

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 18). Tais 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. Nesse caso o algoritmo LRU-WAR toma decisões equivocadas entrando muitas vezes em operação sequencial, 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 sequência de referências o LRU-WARlock também supera o LRU-WAR (Figura 20).

A variação do parâmetro K mostrou que quanto maior a área destinada às páginas mais frequentes, melhor é a detecção das páginas sequências 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%).

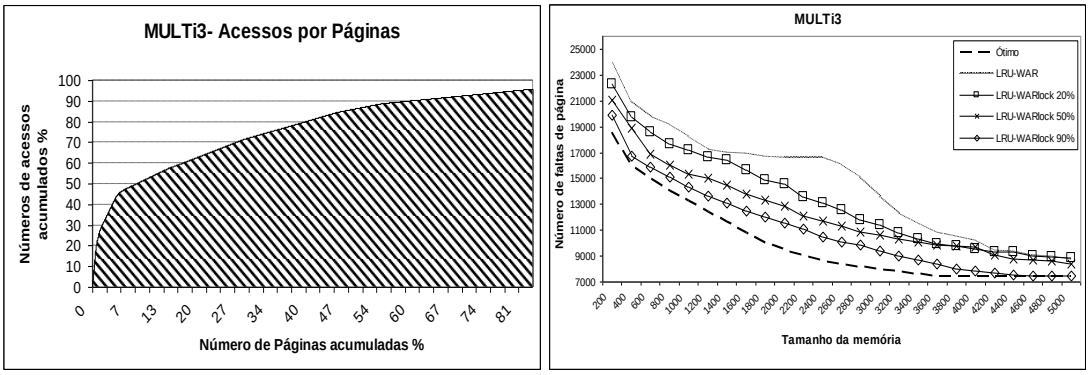


Fig. 20: Gráficos do *trace* multi3. Páginas versus Acessos à esquerda e Variação do parâmetro K à direita

Esse 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 frequência 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. Dessa forma, a área não reservada da memória, que irá trabalhar com o algoritmo LRU-WAR, deve conter muitas páginas que são acessadas com pouca frequência, facilitando assim, a detecção de padrões sequenciais.

4.3. Padrão de Acessos e o Algoritmo LRU+ng

Apesar de ter resultados muito positivos e a técnica de *profiling* parecer ser uma ótima alternativa, já que elimina o tempo de adaptação da política ao padrão de acesso as páginas, o desempenho máximo possível de se obter é limitado pelo algoritmo ótimo sob demanda. Outra questão é que cada página tem relação com outras páginas, e é possível prever alguns acessos antes que eles aconteçam. Para isso, uma proposta de auxílio à política de substituição de páginas foi criada.

O objetivo principal da proposta aqui apresentada é criar uma estratégia para auxiliar políticas de substituição de páginas com a finalidade de obter bom desempenho em um sistema de gerência de memória virtual, sem a necessidade de alterar o seu comportamento (PIANTOLA;

MIDORIKAWA, 2009). Para conquistar este objetivo, foi desenvolvido o algoritmo LRU+ng, que auxilia a política de substituição LRU (Least Recently Used), sem modificá-la, somente utilizando informações de frequência de acessos e adotando a técnica de busca antecipada de páginas (*prefetching*).

O princípio do funcionamento do LRU+ng surgiu da ideia da arquitetura funcional do algoritmo LRU-WARlock que, sem mudar a política LRU-WAR e utilizando a tecnologia de *profiling*, explora a informação de frequência de acessos as páginas. Assim, o algoritmo base LRU-WAR consegue detectar padrões mais facilmente, melhorando o desempenho da gerência da memória. A política LRU foi escolhida por ter um desempenho médio para a maioria de padrões de acesso exibidos pelas aplicações.

O mecanismo utilizado pelo LRU+ng é a busca antecipada de páginas. A busca antecipada trabalha de maneira diferente da paginação por demanda, onde somente a página responsável pela falta é trazida para a memória. No caso da busca antecipada de páginas, não somente a página que sofreu a falta pode ser movida para a memória, mas outras quaisquer de acordo com a estratégia escolhida, tentando prever quais poderão ser referenciadas em um futuro próximo. No LRU+ng, as páginas são selecionadas de acordo com o modelo de n-gramas que é utilizado para simultaneamente separar frequência e tentar prever qual página será acessada imediatamente após a atual.

Com relação à determinação das páginas selecionadas para a busca antecipada, foi escolhido o modelo de bigramas. Esse modelo possibilita capturar informações de frequência de acesso adicionado à probabilidade de previsão da próxima página a ser acessada. A Google, por exemplo, utiliza modelos de n-gramas em vários de seus projetos, tais como reconhecimento de voz, mineração de dados, tradução de texto, e verificação de pronúncia.

4.3.1. Descrição do Algoritmo

O algoritmo LRU+ng adiciona uma técnica e uma informação à política LRU: a técnica de busca antecipada de páginas e a informação de

frequência de acesso. Mesmo com esses mecanismos de auxílio, a política de substituição de páginas não foi modificada, ou seja, a vítima continua sendo a página menos recentemente acessada na memória.

A estrutura de dados do algoritmo mantém uma fila em ordem de acesso. No fim da fila, encontra-se a última página acessada, isto é, a mais recente. No outro extremo, está a menos recentemente acessada, em outras palavras, a vítima da substituição.

A operação do LRU+ng é a mesma do algoritmo LRU, pois quando ocorrer uma falta de página, a página referenciada é movida para a memória e entra na fila na posição MRU. Já a página na posição LRU é descartada da memória. Essa é a descrição de um sistema de paginação por demanda. No caso em que se adota a busca antecipada no momento da falta de página, tem-se, então, que as páginas representadas pelas letras “y” e “z” serão as vítimas da substituição. A página “a” será aquela da busca antecipada e a página “b”, a responsável pela falta (Figura 21).

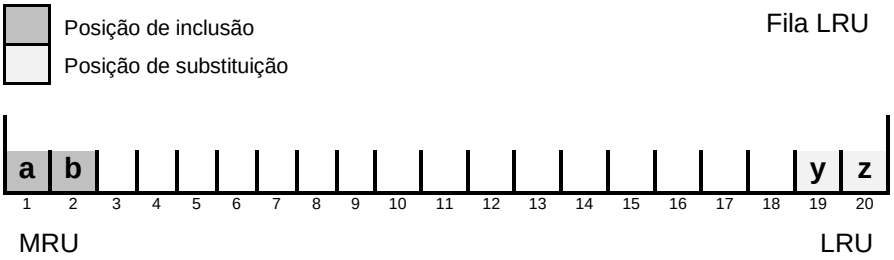


Fig. 21: Estrutura do algoritmo LRU+ng

A busca antecipada de páginas pode acontecer ou não quando ocorre uma falta de página, dependendo da configuração atual do sistema de memória. Se a página escolhida para a busca antecipada já estiver na memória, não haverá busca antecipada e o algoritmo funcionará como o LRU. A escolha da página da busca antecipada é feita através do método de bigramas. Para cada duas referências consecutivas é guardada a frequência com que essas duas páginas são referenciadas novamente em sequência. A página que aparece com maior frequência após a referenciada é a escolhida para a busca antecipada. Por exemplo, se a sequência de acesso para a

página 725 é a 730 e isso já ocorreu 20 vezes e nenhuma outra página a supera, então a página 730 é a candidata da busca antecipada quando ocorre uma falta para a página 725. A estrutura escolhida para guardar esta relação foi um *hash*, pela velocidade da busca nesta estrutura, onde a chave é a página responsável pela falta e o valor é a página da busca antecipada.

A Figura 22 apresenta o pseudocódigo do algoritmo, onde são destacados os acréscimos ao código original do LRU. As novas condições adicionadas ao código são referenciadas nas linhas 6 até 10 e 12. A linha 6 verifica se a página para a busca antecipada encontra-se na memória. Em caso negativo, as linhas seguintes tratam de trazê-la para a memória e remover a página na fila LRU (caso não tenha mais memória disponível); no pseudocódigo, a linha 12 cria, exclui e atualiza a estrutura a cada referência para a busca antecipada.

```
1. Se a página referenciada está na memória:
2.      Reordena a página na primeira posição da fila.
3. Senão, se não está na memória:
4.      Se a memória está cheia:
5.              Remove a página na posição LRU da fila.
6.      Se página do hash para busca antecipada existe e não está na memória:
7.              Se a memória está cheia:
8.                      Remove a página na posição LRU da fila.
9.              Carrega a página referenciada e a página da busca antecipada.
10.     Senão, se a página da busca antecipada está na memória:
11.             Carrega a página referenciada no início da fila.
12. Manutenção do hash para busca antecipada.
```

Fig. 22: Pseudocódigo do algoritmo LRU+ng

É possível remover a função de manutenção do *hash* na linha 12 (Figura 22). Para isso é necessário incluir a técnica de *profiling*, deixando assim o código mais rápido e com maior precisão desde o início de sua execução.

4.3.2. Configuração dos Testes

A avaliação do LRU+ng foi efetuada usando três diferentes traces de aplicações, são eles: multi1, multi2 e multi3. Os três *traces* foram selecionados pelo fato de conterem acessos simultâneos de aplicações, simulando um ambiente multiprogramado. Os *traces* utilizados foram os mesmos da seção 4.2.2 para a análise do algoritmo LRU-WARlock.

4.3.3. Resultados Experimentais e Análises

Nesta seção apresentamos, segundo os objetivos definidos neste trabalho, as análises e os resultados do estudo do desempenho do algoritmo LRU+ng, que utiliza o modelo de bigramas e com busca antecipada de páginas.

- Multi1:

O algoritmo LRU tornou-se o mais amplamente utilizado como política de substituição de páginas porque ele consegue prever razoavelmente o comportamento geral dos programas, atingindo, assim, bom desempenho na maioria dos padrões de acesso à memória. Entretanto, em um programa com padrões de acessos sequenciais e que contém grandes loops, o algoritmo LRU não atinge um bom desempenho. Já o *trace* multi1 apresenta acessos de duas aplicações intercaladas em um sistema de gerência de memória global que têm padrões de acessos sequenciais, fato que causa o mau desempenho. (Figura 23).

No multi1, a análise do perfil de frequência de acessos às páginas mostra que somente 7,5% das páginas são responsáveis por 50% dos acessos à memória. Podemos, então, prever que a informação de frequência de acessos, obtida pelo modelo dos n-gramas, irá proporcionar menos faltas de páginas. O LRU+ng supera o LRU em todos os tamanhos de memória, com ganho médio de quase 36% (Figura 23), o que confirma a nossa previsão.

O gráfico da Figura 23 representa duas grandezas de eficiência sobre a busca antecipada no algoritmo testado. A primeira, com o tracejado em “x”,

representa em porcentagem de quantas vezes LRU+ng fez a busca antecipada sobre todos as faltas de páginas ocorridas em um tamanho de memória especificado pela abscissa. A segunda, com o tracejado em forma de quadrado, representa o total de acertos sobre a página buscada. Um acerto só é mensurado nas páginas que foram buscadas e imediatamente acessadas, ou seja, acessadas na próxima referência.

Uma marca de extrema importância sobre o algoritmo LRU+ng é que ele supera o algoritmo Ótimo em todos os tamanhos de memória menores a 800 páginas. O algoritmo Ótimo é o algoritmo teórico que apresenta o melhor desempenho possível entre os algoritmos de paginação sob demanda. No caso do LRU+ng, a razão para seu melhor desempenho em relação ao algoritmo Ótimo é o fato de que ele não é um algoritmo de paginação sob demanda, pois usa uma técnica de busca antecipada de páginas. Portanto, concluímos que, como nenhum algoritmo de paginação sob demanda pode ser melhor que o Ótimo, nenhum pode alcançar o desempenho obtido pelo LRU+ng para memórias de tamanho pequeno com o *trace* Multi1. Toda referência ao algoritmo Ótimo neste trabalho se trata da substituição ótima sob demanda e não a busca antecipada ótima.

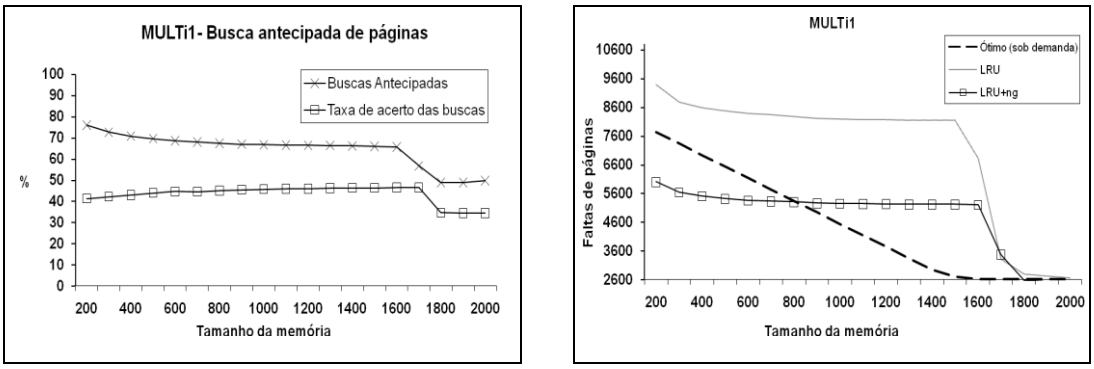


Fig. 23: Gráficos do *trace* multi1. Busca sobre total de faltas à esquerda e Comparação entre LRU, LRU+ng e Ótimo à direita

Podemos observar também que, como preservamos a política LRU no algoritmo LRU+ng, o comportamento do novo algoritmo permanece o mesmo. No gráfico da figura 23 as linhas representadas pelos algoritmos

permanecem paralelas. É possível, assim, melhorar os algoritmos sem a necessidade de modificar a política vigente.

O que marca a grande eficiência do LRU+ng em memórias de pequeno tamanho é que quando temos pouca memória e um número grande de páginas do programa, a probabilidade das páginas acessadas estarem fora da memória principal é alta. Por esse motivo, do total de faltas de páginas, em 76% dos casos é utilizada a busca antecipada de página, ou seja, a página mais acessada após aquela que ocasionou a falta também é colocada na memória. Já em tamanhos grandes de memórias isso ocorre em apenas 49% das faltas.

- Multi2:

A principal característica do *trace* multi2 para a análise surge do fato de ele ser semelhante ao multi1. O que difere os dois *traces* é a adição do programa postgres. O desempenho do LRU+ng também é muito bom, apesar de ser mais difícil distinguir padrões de acessos com a inserção de novas referências intercaladas (Figura 15).

A análise do perfil de frequência de acessos às páginas mostra que metade das referências é feita por apenas 6% do total de páginas. Conforme analisado no *trace* multi1, a informação de frequência, obtida pelo modelo dos n-gramas, proporciona menos faltas de páginas também neste caso.

O LRU é superado pelo LRU+ng em todos os tamanhos de memória. O mesmo acontece com o algoritmo Ótimo, que é superado pelo LRU+ng em tamanhos de memórias pequenas. A eficiência do LRU+ng em memórias de pequeno tamanho no trace multi2, é determinado pelo fato do algoritmo não encontrar a página da busca antecipada na memória e trazê-la para a memória em quase 75% das vezes, contra 22% das vezes em memórias de tamanho grande. Esse comportamento pode trazer muitos benefícios a sistemas que utilizam controle mais rígido de memória para aplicações, ou sistemas de gerenciamento de memória que compartilhem um conjunto mínimo de páginas por processo. Por exemplo, sistemas que adotam uma política de gerenciamento de memória local.

A curva representada no gráfico do LRU+ng se alinha paralelamente com a curva do algoritmo LRU, aumentando ainda mais sua identidade com a política LRU preservada no LRU+ng.

É possível notar que na Figura 24 quanto maior é o tamanho da memória, menos buscas antecipadas são feitas. Entretanto, a precisão aumenta. Isso ocorre porque a maioria das páginas já está na memória. Então, nas poucas vezes em que é executada a busca antecipada, ela ocorre em uma mudança de *working set*. Esses resultados mostram que o LRU+ng é bem adequado para sistemas com política de memória global.

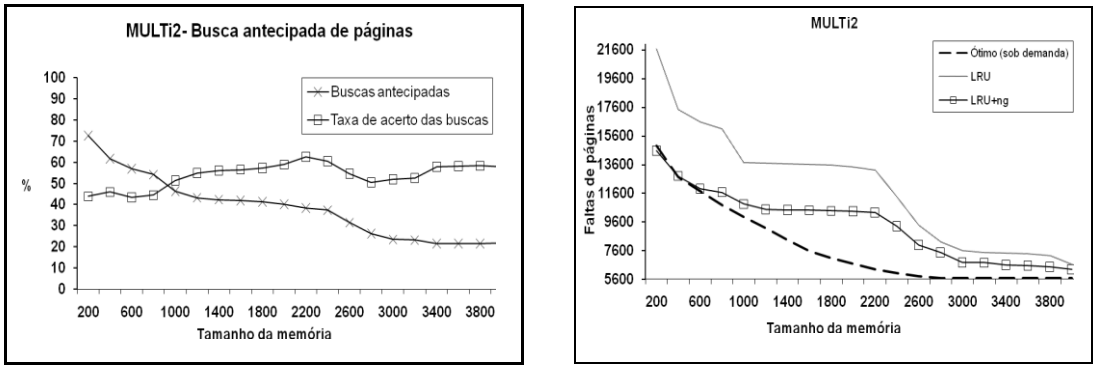


Fig. 24: Gráficos do *trace* multi2. Busca sobre total de faltas à esquerda e Comparação entre LRU, LRU+ng e Ótimo à direita

- Multi3:

Dos traces utilizados na análise, o que contém o maior nível de multiprogramação é o *trace* multi3. São diversos padrões de acessos intercalados entre os quatro programas. Como não existe uma uniformidade da localidade temporal, páginas que serão acessadas em um futuro próximo são descartadas prematuramente. Para essa sequência de referências, o LRU não é bom, porém o LRU+ng recupera essas páginas que foram descartadas prematuramente e supera o desempenho do LRU (Figura 25).

Assim como os *traces* multi1 e multi2, o LRU+ng também supera o algoritmo Ótimo nas memórias de pequeno tamanho no *trace* multi3. Outro fator em comum é o padrão da curva de desempenho do LRU que foi mantida no LRU+ng. Os ganhos obtidos chegam em média a 26% em comparação com o LRU.

Na Figura 25, quanto menor o tamanho da memória, mais buscas antecipadas são feitas, mas a precisão no acerto imediato da página buscada é baixo. No entanto, o desempenho é muito melhor, comparando com as memórias de tamanho médio e grande. O oposto acontece com as memórias de maior tamanho. Apesar da precisão no acerto das páginas das buscas antecipadas ser baixa, os pontos de maior desempenho são encontrados nas memórias de menor tamanho. Uma hipótese a considerar é que as páginas da busca antecipada, apesar de não serem acessadas imediatamente após a busca, são acessadas pouco tempo depois, e assim, não entrando na contagem de taxa de acerto no gráfico (Figura 25). Dois motivos enriquecem essa hipótese. O primeiro é que o *trace* intercala acessos de quatro aplicações, ou seja, uma página qualquer pode estar entre duas páginas da mesma aplicação que geralmente são acessadas consecutivamente. O segundo motivo é que a página da busca antecipada tem grande possibilidade de pertencer ao *working set* atual.

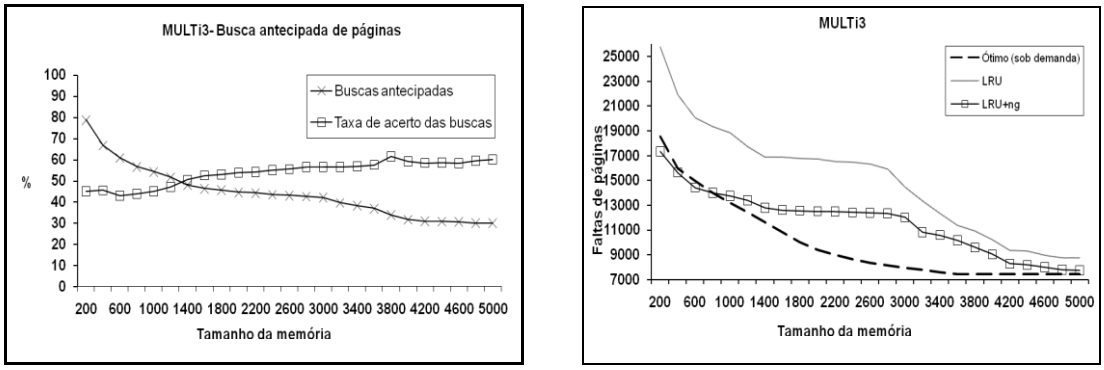


Fig. 25: Gráficos do *trace* multi3. Busca sobre total de faltas à esquerda e Comparação entre LRU, LRU+ng e Ótimo à direita

Como nos outros dois traces estudados, o multi3 também atinge excelente desempenho quando se tem disponíveis quantidades mínimas de memória. Desse modo, é uma estratégia que pode beneficiar sistemas dedicados com recursos restritos. Uma possibilidade de aumentar a taxa de busca antecipada e melhorar o desempenho é, além de utilizar os bigramas, adotar os trigramas.

5. CONCLUSÕES

O objetivo deste trabalho foi contribuir para o aprimoramento dos sistemas de memórias atuais. Isso foi realizado por meio de estudos dos fatores que afetam diretamente o desempenho das políticas de substituição e alguns meios em potencial de como tratá-los.

Sobre a adequação do algoritmo adaptativo LRU-WAR para políticas de gerência de memória global, o 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 frequência 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.

O estudo sobre estratégias para auxiliar políticas de substituição páginas em um sistema de gerenciamento de memória global foi conduzido também sobre o algoritmo LRU modificado, chamado LRU+ng, que aplicou a técnica de busca antecipada de páginas junto com a informação de frequência de acessos, obtida por meio de um método usado em processamento estatístico de linguagem natural.

5.1. Contribuições

Uma das contribuições deste trabalho 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 frequência nos acessos à memória, quando o sistema utilizar uma política de gerência de memória global. A prova dessa 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.

A contribuição mais importante foi mostrar que, apesar de vários estudos anteriores apontarem novas políticas sofisticadas para solucionar o problema da substituição de páginas ou a junção de várias políticas (SMARAGDAKIS, 2004), é possível, sem modificar a política vigente, melhorar seu desempenho com o auxílio de técnicas e informações. A política deixa de ser o objeto único dos esforços para a solução e novas técnicas complementares ganham valor. Outro resultado interessante é que, nos três traces estudados, o LRU+ng supera o algoritmo Ótimo em memórias de tamanho pequeno. Podemos tirar duas conclusões: a primeira é que todo algoritmo de substituição sob demanda é limitado ao Ótimo, ou seja, o Ótimo é o melhor algoritmo de substituição sob demanda, mas impossível de implementar. Algoritmos com busca antecipada podem ultrapassar esse limite imposto pelos sob demanda.

Mais um resultado interessante gira em torno da boa atuação do algoritmo testado em memórias de pequeno tamanho, pois isso faz com que o algoritmo proposto obtenha vantagens em sistemas de memória com gerência local, que distribuem poucas páginas para cada um dos processos.

5.2. Trabalhos Futuros

Alguns estudos complementares podem ser desenvolvidos para dar continuidade a este trabalho. Uma sugestão a considerar é incluir um mecanismo ao LRU-WARlock que trate a frequência de acessos de forma dinâmica sem *profiling* e que não inviabilize sua implementação.

Além da frequência de acessos, outras informações poderiam ser disponibilizadas ao algoritmo de substituição para auxiliar na gerência de memória, como por exemplo, o padrão de acessos à memória, composição do *working set*, última referência de cada página.

Outra sugestão a considerar é incluir, além dos bigramas, trigramas ou n-gramas, para que aumentem as buscas antecipadas em memórias de maior tamanho, diminuindo, assim, faltas de páginas posteriores. Uma segunda sugestão é repetir o estudo realizado com o LRU+ng para políticas

adaptativas, como o LRU-WAR, de forma que informações mais detalhadas e técnicas mais robustas sejam disponibilizadas para a política.

LISTA DE REFERÊNCIAS

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

BELADY, L.A.; NELSON, R.A.; SHEDLER, G.S. An anomaly in space-time characteristics of certain programs running in a paging machine. Communications of the ACM, v.12, n.6, p.349-353, 1969.

BAYLIS, M. H. J., FLETCHER, D. G., and HOWARTH, D. J. (1968). Paging studies made on the I.C.T. ATLAS Computer. In Information Processing, IFIP Congress Booklet D, pages 831-837.

CASSETTARI, H.H.; MIDORIKAWA, E.T. Análise do padrão de acessos à memória de programas paralelos. In: CONGRESSO BRASILEIRO DE COMPUTAÇÃO, 2., Itajaí, 2002. *Congresso Brasileiro de Computação*. Itajaí: Univali, 2002 (a). 1 CD-ROM.

____. StackPar: um simulador paralelo de sistemas de memória baseado em algoritmos de pilha. In: FÓRUM DE INFORMÁTICA E TECNOLOGIA DE MARINGÁ, 4., Maringá, 2002. *IV FITEM:* Anais. Maringá: UEM, 2002 (b). 1 CD-ROM.

____. 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.

____, 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.

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.

CHU, W.W.; OPDERBECK, H. Analysis of the PFF replacement algorithm via a semi-Markov model. Communications of the ACM, v.19, n.5, p.298-304, 1976 (a).

____. Program behavior and the page fault frequency replacement algorithm. Computer, v.9, n.11, p.29-38, 1976 (b).

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

____. Working sets: past and present. IEEE Transactions on Software Engineering, v.SE-6, n.1, p.64-84, 1980.

DENNING, P.J.; SCHWARTZ, S.C. Properties of the working-set model. Communications of the ACM, v.15, n.3, p.191-198, 1972.

FERNÁNDEZ, E.B.; LANG, T.; WOOD, C. Effect of replacement algorithms on a paged buffer database system. IBM Journal of Research and Development, v.22, n.2, p.185-196, 1978.

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, E.E. PDATS II: improved compression of address traces. In: INTERNATIONAL PERFORMANCE, COMPUTING, AND COMMUNICATIONS CONFERENCE, 18., Phoenix, 1999. *IPCCC' 99:* Proceedings. Phoenix: IEEE, 1999. p.72-78.

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.

JURAFSKY, D. and MARTIN, J. H. (2008) Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition. Prentice Hall, New Jersey.

KAPLAN, S.F.; SMARAGDAKIS, Y.; WILSON, P.R. Trace reduction for virtual memory simulations. In: INTERNATIONAL CONFERENCE ON MEASUREMENT AND MODELING OF COMPUTER SYSTEMS, 24., Atlanta, 1999. *SIGMETRICS' 99:* Proceedings. Atlanta: ACM, 1999. p.47-58.

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' 2000:* Proceedings. San Diego: USENIX, 2000. p.119-134.

LEE, D.C. et al. Execution characteristics of desktop applications on Windows NT. In: ANNUAL INTERNATIONAL SYMPOSIUM ON COMPUTER ARCHITECTURE, 25., Barcelona, 1998. *ISCA' 98:* Proceedings. Barcelona: IEEE/ACM, 1998. p.27-38.

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.

MANNING, C. D. and SHUTZE, H. (1999) Foundations of Statistical Natural Language Processing. The MIT Press, Cambridge, MA.

MATTSON, R.L. et al. Evaluation techniques and storage hierarchies. IBM Systems Journal, v.9, n.2, p.78-117, 1970.

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. 193p. Tese (Doutorado) – Escola Politécnica, Universidade de São Paulo. São Paulo, 1997.

____, E.T. et al. Algoritmos adaptativos de substituição de páginas para a próxima geração de sistemas de memória virtual. In: WORKSHOP DE COMPUTAÇÃO, 3., São José dos Campos, 2000. *WORKCOMP' 2000:* Anais. São José dos Campos: ITA, 2000. p.101-107.

____, E.T., PIANTOLA, R.L., CASSETTARI, H.H. 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, 2007.

____, E.T., PIANTOLA, R.L., CASSETTARI, H.H. On adaptive replacement based on LRU with working area restriction algorithm. ACM SIGOPS Operating Systems Review (OSR), vol.42, n.6, Oct 2008, New York, pp 81 – 92.

NGUYEN, A-T. et al. The Augmint multiprocessor simulation toolkit for Intel x86 architectures. In: INTERNATIONAL CONFERENCE ON COMPUTER DESIGN, 14., Austin, 1996. *ICCD' 96:* Proceedings. Austin: IEEE, 1996. p.486-490.

O'NEIL, E.T.; O'NEIL P.E.; WEIKUM, G. The LRU-K page replacement algorithm for database disk buffering. In: INTERNATIONAL CONFERENCE ON MANAGEMENT OF DATA, 19., Washington, 1993. *SIGMOD' 93:* Proceedings. Washington: ACM, 1993. p.297-306.

____. An optimality proof of the LRU-K page replacement algorithm. Journal of the ACM, v.46, n.1, p.92-112, 1999.

PIANTOLA, R.L. and MIDORIKAWA, E.T. Ajustando o LRU-WAR para uma Política de Gerência de Memória Global. In: Anais do V Workshop de Sistemas Operacionais (WSO 2008), Belém, PA, 2008.

___, R.L. and MIDORIKAWA, E.T. Uso de Técnicas e Informações para Potencializar Políticas de Substituição em Sistemas de Memória Virtual. In: Anais do VI Workshop de Sistemas Operacionais (WSO 2009), Bento Gonçalves, RS, 2009.

PINHEIRO, J.C. Avaliação de políticas de substituição de objetos em caches na web. 2001. 108p. Dissertação (Mestrado) - Instituto de Ciências Matemáticas e de Computação de São Carlos, Universidade de São Paulo. São Carlos, 2001.

PRIEVE, B.G.; FABRY, R.S. VMIN – an optimal variable-space page replacement algorithm. Communications of the ACM, v.19, n.5, p.295-297, 1976.

ROBINSON, J.T.; DEVARAKONDA, M.V. Data cache management using frequency-based replacement. In: INTERNATIONAL CONFERENCE ON MEASUREMENT AND MODELING OF COMPUTER SYSTEMS, 15., Boulder, 1990. *SIGMETRICS' 90:* Proceedings. Boulder: ACM, 1990. p.134-142.

ROSEN, Z. Access graph based heuristics for on-line paging algorithms. 1996. 56p. Thesis (Mestrado) - Raymond and Beverly Sackler Faculty of Exact Science, Tel-Aviv University. Tel-Aviv, 1996.

SAMPLES, A.D. Mache: no-loss trace compaction. In: INTERNATIONAL CONFERENCE ON MEASUREMENT AND MODELING OF COMPUTER SYSTEMS, 14., Oakland, 1989. *SIGMETRICS' 89:* Proceedings. Oakland: ACM, 1989. p.89-97.

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.

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.

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

WOOD, C.; FERNÁNDEZ, E.B.; LANG, T. Minimization of demand paging for the LRU stack model of program behavior. Information Processing Letters, v.16, n.2, p.99-104, 1983.