Avaliação de índices de carga de memória em sistemas computacionais distribuídos

William Voorsluys

Avaliação de índices de carga de memória em sistemas computacionais distribuídos

William Voorsluys

Orientador: Prof. Dr. Marcos José Santana

Dissertação apresentada ao Instituto de Ciências Matemáticas e de Computação - ICMC-USP, como parte dos requisitos para obtenção do título de Mestre em Ciências da Computação e Matemática Computacional.

"VERSÃO REVISADA APÓS A DEFESA"

Data da defesa:	06/04/2006
Visto do orientador:	

U S P – S ã o C a r l o s
S e t e m b r o / 2 0 0 6

À Bárbara, minha maior incentivadora.

Agradecimentos

A Deus, pela proteção em todos os momentos.

Aos professores Marcos Santana e Regina Santana pela orientação, pela oportunidade e pela confiança depositada.

Ao professor e amigo Paulo pela orientação constante desde a graduação e pela co-orientação no mestrado.

Agradecimentos especiais à Bárbara, minha esposa, amiga e companheira em todos os momentos. Bárbara, sem você eu não chegaria até aqui!

Aos colegas do grupo LASDPC, amigos nos momentos de estudo e também de descontração.

À Simone, pela amizade, bom humor e hospitalidade em São Carlos.

Às nossas famílias pelo incentivo, que foi essencial para concluirmos essa etapa.

À Universidade de São Paulo, pela ótima estrutura oferecida para a realização do mestrado.

Ao CNPq, pelo apoio financeiro.

Resumo

Este trabalho apresenta uma análise de comportamento de índices de carga relacionados ao uso e à atividade de memória. São descritos 11 índices que refletem direta ou indiretamente a carga de memória. Esses índices podem ser obtidos facilmente no sistema operacional GNU/Linux através do sistema de arquivos /proc. Uma ferramenta de monitoramento foi criada para facilitar a análise de comportamento, podendo também servir para fornecer informações de carga para políticas de escalonamento. Os valores de cada índice foram monitorados durante a execução de uma carga de trabalho composta por aplicações reais que utilizam altas quantidades de memória. A partir dos resultados é possível descobrir a utilidade de cada índice, indicando como eles podem ser usados para auxiliar políticas de escalonamento a avaliar a carga de memória de uma máquina. Métricas de avaliação de carga originárias da combinação de mais de um índice são descritas. Essas métricas foram criadas em casos em que a combinação de mais de um índice permitia representar a carga de memória com mais precisão do que índices usados isoladamente. As métricas e índices pesquisados proporcionam formas precisas de representar a carga de memória em vários níveis, desde níveis de baixa utilização até níveis de saturação da memória principal com sintomas de ocorrência de thrashing.

Abstract

This work presents an analysis of the behavior of load indices related to memory usage and activity. It describes 11 indices which reflect memory load, directly or indirectly. These indices may be easily obtained on the GNU/Linux operating system using the /proc filesystem. A monitoring tool has been developed to make the behavior analysis easier, but it can also be employed to provide load information to scheduling policies. The values of each index have been monitored during the execution of a workload composed by real memory-intensive applications. From the monitoring results, it is possible to find out the usefulness of each index, pointing out how it may be used to help scheduling policies evaluate the memory load of a certain computer. Load evaluation metrics have been created combining more than one index, with the objective of characterizing memory load more precisely than just one isolated index. The explored metrics and indices have shown to provide precise forms of representing memory load at several levels, from low utilization levels up to excessive main memory usage with symptoms of thrashing.

Sumário

Lista de Figuras

Lista de Gráficos

Lista de Tabelas

1. Introdução

1.1 Motivação e objetivos

Os avanços na área da computação paralela e distribuída trouxeram várias vantagens para os usuários, entre elas a possibilidade de se conectar diversos elementos de processamento independentes para formar um sistema de alto desempenho. Isso permite que problemas de grande porte e complexos possam ser resolvidos de forma eficiente, a um custo relativamente baixo.

Um tipo específico de plataforma que merece destaque são as NOWs (*Networks of Workstations*). Esses sistemas são formados por estações de trabalho ou computadores pessoais (nós) conectados por uma rede de comunicação e disponíveis para executar aplicações paralelas (Anderson et al., 1995). Esse tipo de plataforma apresenta características especiais que normalmente não são encontradas nas plataformas de processamento maciçamente paralelo, como heterogeneidade e compartilhamento dos recursos com usuários interativos. Assim surgem novas preocupações a serem consideradas, principalmente no que se refere ao gerenciamento dos recursos da plataforma (Ferstl, 1996).

Uma atividade que tem seu foco fortemente influenciado por essas novas características é o escalonamento de processos. Nesse contexto, vários fatores afetam o desempenho final do sistema, como: características do hardware, carga de trabalho externa e as exigências das aplicações (Souza, 2000). Em ambientes compartilhados por aplicações de vários usuários,

criam-se situações onde nem sempre um determinado elemento de processamento está apto a executar um processo, já que seus recursos podem estar sobrecarregados. Nessas situações é essencial que os algoritmos de escalonamento possuam informações sobre a carga de cada recurso. As particularidades de cada aplicação, no que se refere à demanda por recursos, também devem ser levadas em conta, já que cada tipo de aplicação pode apresentar exigências distintas, tendo seu desempenho afetado por diferentes recursos do sistema.

A eficiência do escalonamento depende da qualidade das informações disponíveis no momento em que as decisões são tomadas. O estudo dos índices de carga visa aumentar a qualidade das decisões de escalonamento através da utilização de informações que reflitam a capacidade dos elementos de processamento em servir os processos a serem escalonados.

Nas décadas de 1980 e 1990 vários estudos foram feitos para avaliar a eficiência de índices de carga destinados predominantemente ao escalonamento de aplicações que dependiam mais da velocidade do processador. A aplicação de índices de carga em algoritmos de escalonamento feitos para aplicações dependentes de outros recursos (por exemplo, a memória), não recebeu muita atenção dos pesquisadores da área. Isso se deve ao fato de que naquela época a velocidade dos processadores era bem menor do que atualmente, tornando-os o principal gargalo dos sistemas computacionais. Além disso, devido ao tamanho reduzido da memória, as aplicações não podiam depender de muitos dados para executar. Portanto, a velocidade desses recursos interferia menos no desempenho das aplicações do que a velocidade do processador.

Com o rápido desenvolvimento da tecnologia dos microprocessadores, o processador deixou de ser um recurso tão caro em relação à memória. Os avanços nas tecnologias RISC (*Reduced Instruction Set Computer*) e VLSI (*Very Large Scale of Integration*) foram muito significativos na última década, contribuindo para o aumento da velocidade dos processadores. A velocidade das memórias também vem aumentando, mas num ritmo bem menor que a dos processadores. Isso contribuiu para aumentar significativamente a diferença

de velocidade entre esses recursos. Por outro lado, as aplicações atuais estão cada vez mais dependentes de dados para executar. Devido a essa realidade, o desempenho das aplicações depende cada vez mais do uso eficaz dos recursos de memória (Xiao et al., 2002; Stallings, 1999). O mesmo ocorre com os dispositivos de armazenamento externo, cuja velocidade de acesso é várias vezes menor que a velocidade de acesso à memória principal (Xiao et al., 2002).

Os fatores de velocidade dos recursos aliados à concorrência entre aplicações e o tamanho insuficiente da memória podem levar à ocorrência excessiva de ausências de página, causando sérias perdas de desempenho. Mesmo os acessos à memória respondidos sem ausências de página são muito custosos (Lin et al., 2001; Zhang et al., 2001). Em situações de carga de memória extrema, surgem as situações de *thrashing,* nas quais o sistema operacional permanece grande parte do tempo tratando ausências de página e pouca computação útil é realizada. O mau gerenciamento da memória acarreta efeitos muito negativos nos sistemas computacionais, podendo ser responsável pela anulação dos benefícios do processamento distribuído. Considerando as exigências das aplicações atuais, a necessidade de bons sistemas de gerenciamento da memória é evidente.

Este trabalho motiva-se no fato de que a memória é um recurso que deve ser utilizado com cuidado especial. Os algoritmos de escalonamento voltados às aplicações que dependem de grandes quantidades de dados devem ser dotados de informações sobre o estado da memória. Isso pode garantir que políticas de escalonamento tomem decisões que mantenham o bom uso desse recurso e idealmente evitem a ocorrência de ausências de páginas. Há uma carência na literatura de estudos sobre índices de carga de memória para escalonamento em plataformas distribuídas. A maioria dos trabalhos considera apenas a carga da CPU (*Central Processing Unit*) como indicativo da carga das máquinas. Esses trabalhos assumem que qualquer máquina do sistema possui memória suficiente para satisfazer todos os programas submetidos ao sistema. O preenchimento dessa lacuna da literatura pode estimular o

desenvolvimento de outros trabalhos que estudem com mais detalhes os efeitos da utilização de cada índice de carga em diferentes situações.

Os objetivos deste trabalho são: (1) pesquisar novos índices de carga relacionados ao uso e à atividade de memória, (2) monitorar o comportamento de cada índice durante a execução de uma carga de trabalho, (3) encontrar relações entre os índices e (4) descobrir como esses índices e as métricas criadas a partir deles podem representar o estado de carga de uma máquina. Foram pesquisados 11 índices de carga relacionados à memória, que podem ser facilmente obtidos no sistema operacional Linux, em nível de usuário e sem a necessidade de adição de módulos ou modificação do código no núcleo (*kernel*).

O objetivo dos índices de carga pesquisados é auxiliar as políticas de escalonamento a tomarem decisões visando otimizar o uso da memória. Este trabalho fornece indicações sobre as situações nas quais cada índice tem o uso recomendado. A maneira exata como esses índices serão utilizados pelas políticas dependerá do objetivo de cada escalonador. A atividade de comprovar a eficiência dos índices no escalonamento está fora do escopo deste trabalho.

Este trabalho foi desenvolvido no contexto do sistema Linux, considerando-se: (1) o fato de seu código ser aberto, o que possibilita um melhor entendimento do funcionamento de processos internos que influenciam o gerenciamento da memória; (2) o desenvolvimento de software no Linux é facilitado pela disponibilidade de boas ferramentas gratuitas; (3) atualmente a abrangência desse SO na área de computação de alto desempenho de baixo custo é muito significativa.

1.2 Organização dos capítulos

O Capítulo 2 apresenta uma revisão bibliográfica da área de escalonamento de processos e índices de carga. São apresentados alguns conceitos de sistemas distribuídos e escalonamento de processos. Também são mostrados conceitos sobre o uso de informações no

escalonamento, classes de aplicações e alguns índices de carga pesquisados por outros autores.

O Capítulo 3 apresenta uma descrição dos 11 índices de carga pesquisados neste trabalho. O significado de cada uma das informações de carga é explicado. Explicações sobre o funcionamento do sistema de gerenciamento de memória são apresentadas quando necessárias para melhor entender um índice.

O Capítulo 4 aborda o problema do monitoramento de índices de carga no Linux, apresentando uma explicação sobre o pseudo-sistema de arquivos /proc. A ferramenta de monitoramento desenvolvida também é descrita nesse capítulo.

O Capítulo 5 é a principal parte desta dissertação; nele é apresenta a análise de comportamento dos índices pesquisados. Essa análise é baseada num monitoramento realizado utilizando três aplicações reais. As configurações do monitoramento e as características das aplicações são detalhadas nesse capítulo. Em seguida, são mostrados diversos dados nos quais se baseiam as conclusões sobre o comportamento de cada índice. No decorrer do capítulo são apresentadas as indicações de como os índices podem beneficiar as políticas de escalonamento.

No Capítulo 6 são apresentadas as conclusões do trabalho, onde uma síntese dos resultados obtidos é mostrada. Também são descritas algumas idéias de trabalhos futuros que podem complementar este trabalho de mestrado. Por fim, o Capítulo 7 lista as referências bibliográficas utilizadas para o embasamento desse estudo.

2. Escalonamento de processos e índices de carga

2.1 Considerações Iniciais

A computação não é a única área onde se emprega o conceito de escalonamento. O ato de escalonar é praticado em diversas áreas do conhecimento, como na engenharia ou até mesmo em nosso cotidiano. Escalonar significa atribuir uma tarefa a ser realizada a um recurso disponível e determinar quando o recurso irá executar a tarefa. No contexto deste trabalho as tarefas a serem escalonadas são os processos e o recurso é um computador.

A eficiência do escalonamento está fortemente ligada às informações disponíveis no momento em que as decisões devem ser tomadas para a atribuição dos processos aos elementos de processamento. Diversas informações podem influenciar na decisão de escalonamento, entre elas está a quantidade de carga de trabalho presente na plataforma computacional (Souza, 2000).

Este capítulo trata dos principais tópicos relacionados ao problema de se escalonar processos em plataformas computacionais distribuídas. Inicialmente são definidos os principais termos, através da revisão da bibliografia da área, com o objetivo de situar o leitor no contexto deste trabalho. Também serão discutidas questões sobre as classes de aplicações, diferenciando-as segundo suas exigências por recursos. Ainda serão discutidas as questões sobre o uso de informações no escalonamento e sobre índices de carga, incluindo os tipos de índices e os exemplos encontrados na literatura.

2.2 Plataformas computacionais distribuídas para execução de aplicações paralelas

Uma plataforma computacional distribuída é uma coleção de elementos de processamento, fisicamente distribuídos, conectados por uma rede de comunicação e utilizada com objetivos específicos (Souza, 2000). Uma das aplicações das plataformas computacionais distribuídas é a execução de aplicações paralelas, onde cada computador interligado é visto como um processador disponível.

Uma abordagem muito difundida atualmente é o uso de NOWs (*Networks of Workstations*) (Anderson et al., 1995), ou COWs (*Clusters of Workstations*, ou simplesmente "*clusters*[1]") (Buyya, 1999). Essa abordagem fornece uma ótima relação custo/benefício, em comparação às máquinas paralelas fortemente acopladas (alto custo), porque são sistemas construídos utilizando-se equipamentos de fácil aquisição, baixo custo, e ao mesmo tempo, alto desempenho. Uma outra vantagem é que computadores de uso geral, conectados por uma rede de comunicação, normalmente já se encontram instalados, reduzindo significativamente o custo para a execução das aplicações paralelas (Souza, 2000). Além de baratos, os *clusters* se tornaram confiáveis e escaláveis devido aos avanços recentes na tecnologia de desenvolvimento de microprocessadores e equipamentos de interconexão (Buyya, 1999). Outro motivo que contribuiu para tornar os *clusters* tão populares foi o surgimento de várias opções de software largamente aceitos, de fácil acesso e custo reduzido. Como exemplo pode-se citar o sistema operacional gratuito Linux e os ambientes de troca de mensagens como o MPI (*Message Passing Interface*) (Gropp et al., 2000).

Um dos recursos dos ambientes de passagem de mensagens é permitir que as aplicações paralelas sejam executadas em plataformas heterogêneas (Souza, 2000). A *heterogeneidade* é um fator importante a ser observado nas plataformas distribuídas, principalmente quando se

[1] É comum encontrar o termo *cluster* como sinônimo de NOW (Buyya, 1999). Graham (2001) afirma que uma NOW é um tipo de *cluster* onde as máquinas são compartilhadas por outros usuários (interativos ou não); o autor trata os *clusters* Beowulf (Becker et al., 1995) como um outro tipo de *cluster*, no qual não há a presença de usuários interativos e as máquinas geralmente são dedicadas à aplicação.

trata de NOWs. Existem basicamente dois tipos de heterogeneidade. O primeiro refere-se a computadores de diferentes arquiteturas (heterogeneidade arquitetural), enquanto o segundo refere-se a computadores de mesma arquitetura, mas que apresentam diferentes configurações de recursos, como velocidade da CPU, quantidade de memória disponível, etc., (heterogeneidade de configuração) (Zhou et al., 1993). É comum que sistemas distribuídos de uso geral apresentem os dois tipos de heterogeneidade. O problema de escalonamento de processos nesses sistemas depende da avaliação da heterogeneidade, já que essa característica tem impacto decisivo no desempenho da maioria das aplicações paralelas, sobretudo se elas dependem da sincronização entre os processos.

2.3 Escalonamento em plataformas distribuídas

O escalonamento tratado neste trabalho é aquele realizado sobre vários computadores, também chamado de escalonamento global, onde o objetivo é decidir qual computador, entre vários disponíveis, está mais capacitado para atender às necessidades de certo processo, segundo objetivos bem definidos. O escalonamento feito em um único computador (local) não será abordado. No escalonamento local o objetivo é gerenciar vários processos que podem executar concorrentemente sobre um único processador disponível, atribuindo fatias do tempo do processador para os processos (Tanenbaum, 2003). O escalonamento de processos em plataformas distribuídas geralmente preocupa-se com o escalonamento global, deixando o escalonamento local a cargo do sistema operacional do computador. O escalonamento global tem sido muito estudado, principalmente com o advento da computação paralela em plataformas distribuídas (Souza, 2000).

Na literatura sobre escalonamento é comum encontrar os termos: elemento de processamento, processador, nó, *host* ou máquina, como sinônimos de computador. Como sinônimos de processo encontram-se os termos tarefa, *job*, programa ou aplicação. Os termos aplicação e *job* também são usados para designar um conjunto de processos (Souza, 2000).

Neste trabalho utilizam-se os termos processo e tarefa como sinônimos. O termo aplicação é usado como sendo a entidade maior de execução composta de um ou mais processos.

Vários autores definem a atividade de escalonamento de processos. Tanenbaum (2003) refere-se ao escalonamento global como "alocação de processadores", definindo essa atividade como sendo o estudo dos "algoritmos usados para determinar quais processos serão atribuídos para quais processadores". Shirazi e Hurson (1992) definem a área de escalonamento global como o estudo da distribuição dos processos entre os elementos de processamento, para atingir alguma meta de desempenho. Essa definição é especialmente interessante por que cita como característica do escalonamento, os objetivos a que ele se propõe. Segundo Souza (2000), esses objetivos inseridos no algoritmo de escalonamento, determinam a sua "personalidade" e restringem a sua área de atuação. Exemplos desses objetivos são: maximização do *throughput,* otimização da utilização dos processadores, redução do tempo de execução e balanceamento de carga (Shirazi e Hurson, 1992; Feitelson et al, 1997; Xu e Lau, 1997).

Casavant e Kuhl (1988) definem o escalonamento global, como sendo um recurso para o gerenciamento de recursos. Esse gerenciamento é composto basicamente por uma política usada para permitir o acesso e o uso de um recurso por seus vários consumidores, de modo eficiente (Figura 2.1). Os recursos são processadores, memórias, rede de comunicação, entre outros; e os consumidores são os usuários representados por suas aplicações seqüenciais ou paralelas.

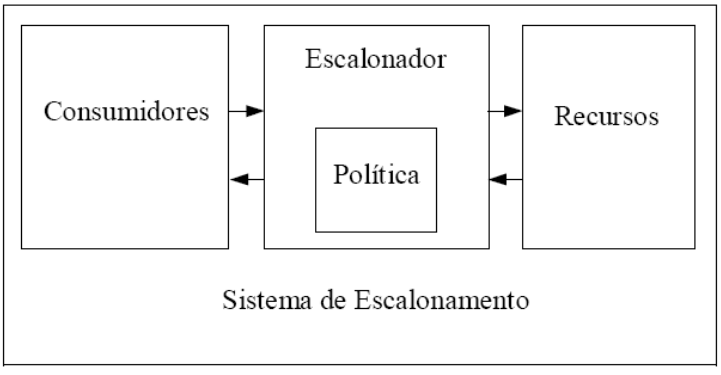

Figura 2.1 – O escalonamento segundo Casavant; Kuhl (1988). Extraída de Souza (2000)

A lógica necessária para se efetuar o escalonamento é implementada pelo *algoritmo de escalonamento*. No algoritmo podem ser inseridos diversos *mecanismos* responsáveis pelos procedimentos necessários para atingir o objetivo do escalonamento. Alguns exemplos são os mecanismos para coletar informações sobre a carga do sistema e trocar informações sobre os processadores (Souza, 2000).

2.3.1 Componentes de um algoritmo de escalonamento

Por questões práticas geralmente divide-se um algoritmo de escalonamento em módulos, ou "componentes". Shivaratri et al. (1992), afirmam que um algoritmo de escalonamento tipicamente é dividido em quatro componentes (chamados aqui de "políticas"): política de transferência, política de seleção, política de localização e política de informação.

A política de transferência determina se um nó deve ou não participar de uma transferência de processos, seja como transmissor ou como receptor. A utilização de limites (*thresholds*) é muito comum na implementação de políticas de transferência. Um nó se torna um transmissor quando a sua carga excede um limite T_1, ou pode se tornar um receptor quando a carga está abaixo de um limite T_2. Dependendo do algoritmo, os valores de T_1 e T_2 podem ser iguais. Uma outra opção seria uma política de transferência relativa, onde a carga de um nó A seria comparada à carga de outro nó B para determinar se A ou B deve se tornar transmissor ou receptor.

Assim que a política de transferência determina que deva haver uma transferência, deve-se escolher qual processo será transferido. Esse é o trabalho da política de seleção que deve ser capaz de encontrar um processo apto a ser transferido. Caso isso não seja possível o nó não pode mais ser considerado um transmissor. Efetuar uma transferência não significa necessariamente transferir um processo já iniciado. A maneira mais simples de implementar uma política de seleção é selecionar os processos mais novos, ou seja, aqueles que acabaram de chegar para serem escalonados e, portanto, não iniciaram sua execução. As transferências

desse tipo não são preemptivas, portanto são mais baratas. Transferências preemptivas requerem a migração de um processo que já iniciou sua execução e envolvem uma série de tarefas para possibilitar que o processo possa ser reiniciado em outra máquina a partir do ponto que parou. No caso da necessidade de transferências preemptivas, a política de seleção deve escolher os processos que proporcionarão uma transferência mais rápida. Nesse caso a seleção pode necessitar de uma análise mais criteriosa.

A política de localização é responsável por encontrar um parceiro para a transferência (transmissor ou receptor). Uma política de localização descentralizada muito usada procura um parceiro através de *pooling*. Nessa técnica um nó "pergunta" para os outros nós se algum deles aceita participar de uma transferência para receber ou transmitir processos. Numa abordagem centralizada, é feita uma consulta a um nó específico, chamado de coordenador, que localiza um outro nó apto a participar da transferência.

A política de informação é responsável por coletar informações sobre o sistema. Ela decide qual informação coletar, onde encontrar essa informação e quando utilizá-la. É com base nos valores fornecidos pela política de informação que a política de transferência determina se deve ou não haver uma transferência de processos.

2.4 Classes de aplicações

Sob o ponto de vista da utilização de recursos, as aplicações paralelas se comportam de diferentes maneiras. Uma forma de classificar as aplicações é agrupando-as em classes de acordo com o seu comportamento. Muitas aplicações paralelas, quando executadas, demonstram uma tendência em utilizar um determinado recurso (ou mais de um recurso) mais intensamente do que outros. Os recursos que predominantemente influenciam o desempenho das aplicações são: CPU, memória, dispositivos de I/O, e rede de comunicação.

As aplicações que permanecem a maior parte do seu tempo de execução utilizando a CPU são ditas "orientadas à CPU" (*CPU-bound*, ou *computation-intensive*). Diversos exemplos de aplicações *CPU-bound* podem ser facilmente identificados em áreas como

engenharia, física e biologia. Geralmente, as aplicações dessas áreas são baseadas em métodos matemáticos complexos, cálculos repetitivos ou buscas exaustivas, o que demanda muito processamento. Os processos gerados por esse tipo de aplicação geralmente mantêm a CPU ocupada for todo o seu tempo de execução.

Aplicações que necessitam de grandes quantidades de memória principal para executar são tratadas na literatura como *memory-intensive* (ou menos comumente *memory-bound*). Qualquer aplicação que manipula grandes conjuntos de dados e os armazena na memória principal pode se enquadrar nessa categoria, por exemplo, programas de *data mining*, buscas em bancos de dados, reconhecimento de voz, processamento de vídeo e modelos climáticos. É comum que as aplicações *memory-intensive* também sejam consideradas *CPU-bound*, já que, além de manipular grandes conjuntos de dados, elas também podem demandar por processamento pesado nesses dados.

Outra classe de aplicação importante é a classe das aplicações "orientadas a *I/O*" (*I/O-bound*, ou *I/O-intensive*). Na literatura o termo *I/O-bound* geralmente refere-se a aplicações que utilizam com freqüência os dispositivos de armazenamento (por exemplo, disco rígido local). Essas aplicações caracterizam-se por realizar muitas leituras e escritas no disco durante a execução. Isso acontece quando o processamento realizado gera muitos resultados que devem ser armazenados ou precisa ler um grande volume de dados.

Para as aplicações que efetuam muitas comunicações através da rede é reservada a denominação *communication-intensive*, ou menos freqüentemente, *communication-bound* e *network-bound*. Diversas aplicações, quando paralelizadas, geram processos que executam concorrentemente, mas não independentemente. Nesse caso, os processos necessitam trocar dados entre si freqüentemente para atingir o objetivo comum da aplicação.

O termo *data-intensive* é utilizado para se referir às aplicações que manipulam grandes conjuntos de dados (Amiri et al., 2000). Devido ao seu comportamento, geralmente essas aplicações são classificadas como *memory-intensive*, já que os dados necessários permanecem

na memória enquanto são processados e também *I/O-bound*, devido à necessidade de manter os dados em disco rígido e utilizá-los durante o processamento.

As classes de aplicações não são conjuntos disjuntos; há aplicações que não demonstram uma tendência bem definida para que estejam aptas a pertencer a apenas uma classe. Freqüentemente, durante a execução, as aplicações podem ter os comportamentos alterados, exibindo tendências de utilização de recursos opostas em diferentes fases de sua execução. Nesse caso, por exemplo, é possível que uma aplicação seja considerada *CPU-bound* durante uma ou mais fases e *I/O-bound* em outras fases da execução.

Na literatura encontram-se afirmações que confirmam a superioridade do escalonamento voltado a classes de aplicações específicas, frente ao realizado por escalonadores que não obtêm informações sobre as aplicações (Souza, 2000). O estudo deste trabalho visa viabilizar a exploração dessas informações quando são conhecidas. Em específico, os índices de carga pesquisados são voltados à classe de aplicações *memory-intensive*, que representa uma grande gama de aplicações das mais diversas áreas.

2.5 O efeito do mau gerenciamento de memória no desempenho dos sistemas

Um dos principais limitantes do desempenho das aplicações que utilizam bastante memória é a ocorrência excessiva de ausências de páginas (*page faults*). Isso se deve à concorrência entre os programas em sistemas com memória limitada e à lentidão dos dispositivos de armazenamento externo, que são usados como memória secundária. Segundo Hennesy e Patterson (1996), a latência de uma ausência de página é mais de 1000 vezes maior que um acerto (*page hit*).

Há estudos que avaliam a execução de programas reais e quantificam as porções de tempo gastas com operações de CPU e acessos à memória, como o descrito por Zhang et al. (2001). Usando os *benchmarks* do conjunto SPEC 2000 numa máquina simulada com processador de 1.6GHz, cache L1 de 64KB com associação de 4 vias, 1MB de cache L2 e

memória principal infinita, os autores constataram que, na média, o processador gasta 57% do tempo processando acessos à memória principal (perdas de L2), 12% servindo perdas de L1 e apenas 31% do tempo com operações de CPU.

A inserção das ausências de páginas num cenário como esse pode levar a utilização da CPU a níveis bem mais baixos. Nos momentos em que o tratamento das ausências de página foge do controle do sistema operacional e a utilização do sistema cai tanto a ponto de haver pouca computação útil, situações de *thrashing* aparecem. A ocorrência de *thrashing* está associada a várias condições: (1) ao tamanho limitado da memória principal, (2) ao número de processos competindo por memória, (3) às exigências de memória de cada programa e (4) ao algoritmo de substituição de páginas (Jiang e Zhang, 2001).

Os sistemas operacionais multitarefa gerenciam a memória virtual de modo que todos os processos tenham as mesmas chances de executar. Para garantir esse princípio eles adotam um mecanismo de substituição de páginas que entra em ação quando a quantidade de memória requisitada pelos programas excede o espaço de memória principal disponível. Nessa situação o mecanismo deve escolher, baseando numa política, uma página residente na memória principal, que deverá ser enviada para o espaço de *swap* e substituída por outra página. Uma política de substituição de páginas muito usada é a LRU (*Least Recently Used* ou Menos Recentemente Usada), que seleciona para substituição páginas que não foram acessadas recentemente (Tanenbaum, 2003).

Cada programa possui um conjunto mínimo de dados que devem ser mantidos na memória principal para que ele possa trabalhar. Esse conjunto é conhecido com o *"working set"* do programa. Quando o acúmulo da demanda de memória dos programas excede o espaço disponível, pode haver situações em que nenhum programa consegue estabelecer o seu conjunto mínimo de trabalho. Nessas situações o sistema pode enfrentar muitas ausências de páginas, baixa utilização da CPU e longas demoras no tempo de resposta dos programas. Nesse caso diz-se que o sistema está numa situação de "*thrashing*" (Jiang e Zhang, 2005).

O comportamento da política LRU também pode levar a situações desfavoráveis. Essa política permite que, em situações de *thrashing*, páginas que recentemente foram trazidas para a memória, com o objetivo de formar o conjunto mínimo de trabalho de um programa, sejam substituídas antes mesmo de o programa começar a operar sobre elas. Uma página de memória virtual torna-se candidata para substituição caso não seja acessada por certo período. Isso pode ocorrer em duas situações: (1) o programa não precisa da página e ela não é acessada; e (2) o programa não é capaz de acessar a página porque está tratando de ausências de páginas que estão ocorrendo. Na literatura as páginas LRU do primeiro caso são conhecidas como LRU verdadeiras, enquanto que as do segundo caso são as páginas LRU falsas. As páginas falsas aparecem porque, para a política LRU, todos os programas possuem o mesmo direito de manter seus dados na memória (Jiang e Zhang, 2005).

Segundo Jiang e Zhang (2001), o aspecto mais destrutivo das situações de *thrashing*, é que, apesar dessas situações serem iniciadas por picos repentinos de carga, o sistema pode continuar em *thrashing* por tempo indefinido, ou seja, é difícil recuperar-se dessas situações. Diversas soluções foram propostas para tentar proteger o sistema de situações de *thrashing*; algumas inclusive estão implementadas em sistema operacionais reais. As soluções estão divididas em duas categorias: (1) soluções para prevenir a ocorrência de *thrashing*, e (2) soluções para minimizar o efeito do *thrashing* depois que ele é detectado. Na primeira categoria estão incluídos os modelos de conjunto mínimo de trabalho (Denning, 1968a) e as soluções de substituição local de páginas (Alderson, 1972). Na segunda categoria, estão os mecanismos de controle de carga (Denning, 1968b).

Mesmo com a ajuda dos mecanismos de prevenção e recuperação, pode não ser possível manter o bom desempenho de um sistema sob grande quantidade de carga. A solução definitiva para o *thrashing* continua sendo atualizar o *hardware* do sistema caso as situações de falta de memória sejam freqüentes. É possível criar sistemas com grandes quantidades de memória, mas esse recurso possui um custo não desprezível e continua sendo limitado.

No caso de sistemas computacionais de processamento paralelo com memória distribuída, garantir a boa utilização da memória local em cada nó é essencial para manter o desempenho de todo o sistema. Como a maioria das aplicações que executam nesses sistemas depende da coordenação (sincronização) entre os processos, a ocorrência de *thrashing* em um único nó ou em um pequeno conjunto de nós, pode impedir que os processos que executam em outros nós façam alguma computação útil. Por isso, evitar o *thrashing* localmente em cada nó é altamente desejável para manter o bom desempenho do sistema como um todo (Jiang e Zhang, 2005). Outro problema que afeta negativamente o desempenho de sistemas paralelos é a falta de balanceamento da carga de memória. Acharia e Setia (1999) mostram que, na prática, as utilizações das memórias de diferentes nós tendem a ficar altamente desbalanceadas. Nós mais carregados podem estar sofrendo ausências de páginas, enquanto poucos acessos à memória são feitos em nós com pouca carga ou ociosos.

A partir dessas observações podem ser sintetizadas quatro motivações em que este trabalho se baseia: (1) as aplicações estão cada vez mais dependentes de dados, (2) a memória é um recurso lento, (3) o thrashing pode ser desastroso e (4) o mau gerenciamento de memória afeta o negativamente o desempenho dos sistemas paralelos.

2.6 Uso de informações no escalonamento

A eficiência do escalonamento depende da qualidade das informações disponíveis no momento em que as decisões devem ser tomadas para a atribuição dos processos aos elementos de processamento. Diversas informações podem influenciar na decisão de escalonamento, entre elas está a quantidade de carga de trabalho presente na plataforma computacional (Souza, 2000).

Freqüentemente também se desejam obter informações sobre as aplicações que serão executadas na plataforma. Isso inclui descobrir através de observações de comportamento quais são os recursos (memória, CPU, disco, etc.) mais utilizados pela aplicação durante sua execução. Por conseguinte, é possível saber quais recursos têm maior influência sobre o

desempenho dessa aplicação. Quando se conhece a classe das aplicações que serão executadas, podem-se utilizar políticas de escalonamento destinadas a essa classe. Essas políticas são mais específicas e geralmente dependem de menos tipos de informações adicionais. Por outro lado, quando se conhece pouco sobre as aplicações, as políticas utilizadas devem ser mais genéricas e, conseqüentemente, dotadas de informações mais abrangentes. Em resumo, quanto mais conhecimento se tem sobre o ambiente, mais precisa pode ser a decisão de escalonamento tomada. O nível de informação e a forma como essas informações são utilizadas dependem das características dos algoritmos de escalonamento.

Geralmente, algoritmos que realizam escalonamento dinâmico (os que tomam decisões no momento em que as aplicações são submetidas ao sistema) trabalham com menos informações porque têm menos tempo para calcular o escalonamento e as aplicações chegam em intervalos não previsíveis. Por outro lado, as técnicas de escalonamento estático usam um conjunto fixo de aplicações, um conjunto fixo de máquinas e um conjunto fixo de atributos das máquinas e das aplicações para gerar um escalonamento fixo. Como as decisões são feitas antes da execução (*off-line*) e são destinadas a ambientes de produção cujas características não mudam, o algoritmo pode levar mais tempo combinando várias informações e gerando um escalonamento quase ótimo (Ali et al., 2005). Existem heurísticas de escalonamento estático que utilizam informações sobre o tempo de execução de todas as aplicações em todas as máquinas disponíveis para definir a seqüência de execução de um conjunto de tarefas (Braun et al., 2001).

Os escalonadores dinâmicos podem “aprender” novos fatos que podem ser usados posteriormente para melhorar o escalonamento. Isso é obtido usando-se o *feedback* fornecido pelas próprias aplicações e através de observações da plataforma. A utilização do tempo de execução de tarefas terminadas pode ajudar na previsão do tempo de execução quando tarefas semelhantes (p.ex. pertencentes à mesma aplicação) forem submetidas. Na literatura podem ser encontrados alguns trabalhos que exploram as informações obtidas em execuções passadas

para prever o tempo de execução de aplicações submetidas posteriormente (Harchol-Balter e Downey, 1997; Senger, 2004). Os algoritmos desenvolvidos nesses trabalhos utilizam um conjunto de atributos das aplicações para calcular a similaridade entre aplicações paralelas. Esses atributos incluem: o identificador do arquivo executável da aplicação, argumentos da aplicação, usuário que fez a submissão, grupo do usuário e o número de elementos de processamento requisitados (Smith et al., 1999; Senger, 2004). Para prever o tempo de execução das tarefas os autores empregam mecanismos como médias, regressão linear, algoritmos genéticos e soluções de aprendizado baseado em instâncias (Senger, 2004).

2.7 Índices de carga

Um *índice de carga* é uma métrica destinada a quantificar a condição de carga de uma máquina num determinado instante, ou num passado recente, dependendo da freqüência de atualização da informação. O índice geralmente é um valor numérico iniciado em zero indicando que a máquina está ociosa, e que tenha seu valor aumentado à medida que a carga da máquina aumente, passando por níveis considerados "normais", até indicar a situação onde a máquina deve ser considerada sobrecarregada, não podendo mais aceitar processos para execução (Ferrari e Zhou, 1987). O principal objetivo do índice é poder prever a carga do processador num futuro próximo, fornecendo assim uma expectativa do desempenho de uma aplicação após o escalonamento. Geralmente, a máquina com o menor índice de carga tem o maior potencial para executar uma tarefa mais rapidamente.

Um índice de carga ainda deve possuir outras características importantes: (1) deve ter uma relação direta com a métrica utilizada para medir o desempenho das aplicações, para que seja possível sintonizar o índice a partir das observações do desempenho da aplicação; (2) deve gerar pouca sobrecarga na coleta das informações e no cálculo do índice, para permitir que medições de carga mais freqüentes sejam feitas, aumentando sua precisão; (3) não deve ser afetado com intensidade por flutuações nas medições instantâneas de carga; (4) deve servir para comparar a carga de diferentes máquinas, especialmente em sistemas heterogêneos

(Ferrari e Zhou, 1987; Mehra e Wah, 1997). A conformidade do índice com essas características depende de vários fatores, entre eles, a quantidade e o tipo das informações que o índice incorpora e a forma como o índice é calculado.

Basicamente, os índices de carga podem ser classificados em dois grupos: simples ou compostos. Os índices simples são mais específicos e são baseados em apenas um valor, que reflete a carga de um único recurso (por exemplo, CPU). Esses índices são mais indicados para casos onde o desempenho da aplicação depende apenas de um recurso e sabe-se qual é esse recurso. Por exemplo, o tamanho da fila da CPU é um índice simples que poderia ser usado no escalonamento de aplicações *CPU-bound*. Os índices compostos geralmente são mais genéricos e visam representar a carga de mais de um recurso combinando as medições de carga dos recursos em um único valor. Esse tipo de índice pode ser útil quando não há conhecimento suficiente sobre a aplicação para determinar qual recurso afeta mais o seu desempenho, ou quando se sabe que a aplicação depende de mais de um recurso (por exemplo, CPU e memória). Também pode haver índices compostos que combinam diferentes informações sobre o mesmo recurso; isso ocorre quando as informações são complementares e a combinação pode apresentar uma estimativa melhor do que apenas uma informação. Uma vantagem de usar índices simples é a relativa simplicidade de obtê-los e a pequena sobrecarga gerada. Em contrapartida, um índice composto pode oferecer informações mais completas que podem se transformar em mais desempenho para certos tipos de aplicações.

O problema da escolha de métricas de desempenho para a formação de índices compostos é um campo de estudo interessante e ainda não muito explorado. Tipicamente, num sistema operacional do tipo Unix, diversas variáveis, que podem indicar a situação da carga de vários recursos, são fornecidas pelo próprio sistema operacional. Partindo do princípio de que todas essas métricas já estão prontas para uso, pode-se querer idealizar a composição de um índice extremamente genérico composto de todas as métricas encontradas. No entanto, essa pode não ser a melhor solução. Alguns fatores fazem com que certas

variáveis se tornem inadequadas para a inclusão em um índice de carga, como: a sobrecarga associada à medição. Assim, deve-se escolher um pequeno conjunto de métricas que não estejam sujeitas aos fatores citados e que sejam representativas o suficiente para refletir a carga dos recursos para identificar as situações de sobrecarga.

A dificuldade em se calcular um índice de carga é determinada por vários fatores. Os principais pontos que devem ser considerados são: (1) a unidade utilizada para medir a carga de diferentes recursos, que determina se os valores podem ser comparados entre si, e (2) a existência de limite superior ou inferior para essa carga, que serve para determinar o nível no qual um recurso pode ser considerado sobrecarregado. Os valores da carga de muitos recursos não são medidos na mesma unidade (Mehra e Wah, 1997). A utilização da CPU é medida em ciclos; a fila da CPU é dada por um número de processos; a utilização da memória é medida em Megabytes; os recursos de comunicação podem ser medidos em disponibilidade de largura de banda, quando se deseja medir sua utilização, ou em Bytes transferidos ou número de pacotes quando se deseja medir sua atividade; o disco tem sua atividade medida em Bytes transferidos ou páginas transferidas. Essa disparidade de unidades faz com que as cargas desses recursos sejam incomparáveis entre si, tornando os cálculos necessários mais complicados do que "simples médias," principalmente quando os recursos são heterogêneos. No caso de alguns dos recursos citados a determinação de um limite superior é uma tarefa fácil. Por exemplo, a quantidade de memória disponível é finita e esse recurso pode ser considerado sobrecarregado assim que toda a memória esteja ocupada. Por outro lado, determinar quando a CPU está sobrecarregada, baseando-se em sua fila é uma tarefa difícil e pode requerer uma análise baseada em testes e observações sobre o comportamento do sistema.

Informações sobre a heterogeneidade configuracional da plataforma também podem ser incorporadas aos índices de carga. A abordagem mais comumente empregada para medir os níveis de heterogeneidade é a utilização de informações sobre a capacidade dos recursos das

máquinas (por exemplo: velocidade do processador, tamanho da memória). Essas informações são obtidas diretamente a partir de informações sobre o hardware da máquina, ou podem ser medidas por *benchmarks* (programas destinados a quantificar a capacidade dos recursos através de testes específicos). Uma forma de se utilizar esse tipo de avaliação de heterogeneidade é através da atribuição de um valor para cada máquina, que reflita os resultados dos testes e medições realizadas. Através desse valor é possível comparar quantitativamente a capacidade das máquinas. Para proporcionar decisões de escalonamento que reflitam a heterogeneidade, o valor de cada máquina pode ser usado para normalizar o índice de carga, isto é, as máquinas de potência computacional inferior têm seus índices rebaixados, enquanto que com as melhores máquinas ocorre o contrário. Um possível efeito dessa abordagem é que a política de escalonamento pode evitar a atribuição de processos às máquinas inferiores, mesmo que estas estejam ociosas, enquanto as máquinas superiores tenham condições de tratar os processos. Os índices que combinam informações de carga com informações de heterogeneidade são conhecidos como índices de desempenho (Branco, 2004).

2.7.1 Exemplos de Índices de Carga

A literatura estudada apresenta diversos exemplos de informações utilizadas em índices de carga, tanto simples como compostos. Alguns dos primeiros estudos sobre índices de carga foram feitos por Ferrari e Zhou (1987) e por Kunz (1991). Nesses dois estudos os autores investigaram, através de simulações ou execuções reais, o impacto da utilização de diferentes informações na distribuição de processos em plataformas homogêneas. Ambos os estudos apresentam conclusões semelhantes, afirmando que a melhor informação de carga investigada é a fila do processador (número de processos rodando) e que os ganhos obtidos com o uso de índices compostos eram insignificantes. Para formar os índices compostos os autores utilizaram informações como a quantidade de memória livre e atividade de *I/O*.

Essa ausência de ganhos expressivos nos estudos mais antigos deve-se ao fato de que no final dos anos 80 e no início dos anos 90 as aplicações eram predominantemente *CPU-bound*. Como já mencionado anteriormente, esse panorama não condiz com a realidade atual dos sistemas e das aplicações.

Alguns trabalhos relativamente recentes, que apresentam estudos sobre índices de carga voltados à memória, podem ser encontrados na literatura. Um exemplo relevante é o estudo feito por Xiao et al. (2002). Nesse estudo os autores utilizam apenas duas informações para compor o índice de carga utilizado para basear as decisões de escalonamento: a fila do processador e a quantidade de memória livre. O objetivo do algoritmo, com a utilização dessas duas informações, é minimizar os tempos de ociosidade do processador e o número de ausências de páginas ocorridas. Duas heurísticas foram criadas, uma destinada à computação de alto desempenho (*high-performance computing*) e outra destinada à computação de alta vazão (*high-throughput computing*). A primeira heurística impede que um nó receba processos quando a memória está sobrecarregada, sendo que nesse caso os processos são enviados a outros nós ou colocados em uma fila de espera. Nas situações onde isso acontece, o valor da fila do processador é agressivamente ajustado para um valor máximo, que indica sobrecarga total do sistema. Por definição, a computação de alta vazão enfatiza que os recursos devem ser bem gerenciados para que o maior número possível de usuários possa ser atendido. Portanto, a heurística destinada a esse tipo de computação não pode impedir que os processos sejam executados quando a memória está sobrecarregada. Nesse caso o ajuste do índice de carga é feito proporcionalmente à utilização da memória. Especificamente, quando a quantidade de memória utilizada é k vezes maior que a quantidade disponível, o valor da fila do processador aumenta k vezes. O aumento desse índice de carga diminui as chances do nó ser escolhido para receber processos. Em ambas as heurísticas todos os processos são executados localmente (na máquina onde são gerados), enquanto o valor do índice de carga não superar um máximo estipulado. Quando esse valor é ultrapassado a máquina remota

menos sobrecarregada é escolhida para executar o processo. Neste trabalho pouca atenção é dada às informações sobre os dispositivos de *I/O*. Algumas informações sobre as atividades de disco são utilizadas apenas na política de seleção do algoritmo desenvolvido. Os testes feitos nesse estudo demonstraram ganhos significativos nas execuções de diversas aplicações, inclusive em aplicações *I/O-bound*. Apesar dos autores não terem incorporado ao índice de carga nenhuma informação especificamente relacionada aos dispositivos de *I/O*, os ganhos obtidos com as aplicações *I/O-bound* são justificados pelo fato de seu desempenho também ser influenciado pelas ausências de página, já que os *buffers* necessários para as transferências de *I/O* competem pelo mesmo espaço de memória que as aplicações.

O trabalho de Amir et al. (2000) utiliza uma abordagem parecida com a apresentada por Xiao et al. (2002). Informações sobre a CPU e a memória são utilizadas para formar um único índice, chamado pelos autores de "custo". Nas simulações realizadas presume-se que a quantidade de memória que cada aplicação necessita é conhecida a priori. Para escolher onde escalonar um processo, o algoritmo estima qual seria o custo das máquinas após receberem aquele processo, e a máquina escolhida é a que apresenta o menor custo estimado.

No sistema de compartilhamento de carga MOSIX (Barak e Braverman, 1998), foi implementada uma política que monitora a quantidade de memória livre das máquinas e realiza migrações de processos quando o nível de memória livre fica abaixo de um mínimo predefinido. Neste trabalho o único índice de carga utilizado é a quantidade de memória livre. Não são considerados índices compostos que levem em conta, por exemplo, a fila do processador.

O trabalho de Mehra e Wah (1997) é caracterizado por apresentar diversas propostas de informações de carga, que podem fazer parte de um índice de carga composto. No entanto, nem todas as informações mencionadas foram efetivamente testadas em índice de carga reais, já que muitas não eram apropriadas para a utilização no escalonamento. O sistema de escalonamento utilizou quatro informações: (1) o número de páginas livres de memória, (2) o

tempo gasto nos diferentes estados da CPU: ocioso, em programas de usuário, em funções do SO, etc., (3) a quantidade de dados transferida em cada disco e (4) o número de pacotes transferidos em cada interface de rede. A partir dessa escolha, os autores empregaram um sistema de redes neurais para possibilitar um aprendizado automatizado das políticas de escalonamento. O objetivo do sistema é prever o tempo que um determinado processo leva para terminar, caso seja colocado em um determinado processador. O sistema leva um tempo considerável para aprender sobre o comportamento das aplicações, efetuando medições desses tempos em todas as máquinas do *cluster*. O sistema grava os padrões de utilização de recursos apresentados pela aplicação e no momento da execução ajusta os parâmetros do índice de carga da forma que melhor se ajuste àquela aplicação.

Outro trabalho que merece destaque neste contexto é o de Ishii (2005). Esse estudo tem como objetivo explorar as informações relacionadas à comunicação entre os processos das aplicações paralelas e utilizá-las na construção de uma política de escalonamento. A heurística desenvolvida faz uso de três informações para tomar as decisões: (1) a carga da CPU, (2) a carga da rede de comunicação e (3) a carga de comunicação gerada pela aplicação em questão. Quando as duas cargas de comunicação apresentam valores baixos, a heurística considera apenas a carga da CPU para escalonar os processos. Em outra situação extrema, ou seja, quando as duas cargas referentes à comunicação apresentam valores muito altos, a heurística procura escalonar os processos de forma que eles não gerem mais sobrecarga à rede de comunicação. Uma forma de fazer isto é colocando os processos que se comunicam muito no mesmo processador.

O trabalho de Branco (2004) propõe o desenvolvimento de novos modelos de índices de carga e índices de desempenho através de modelagem estocástica. Uma das contribuições desse estudo é um modelo para quantificar o grau de heterogeneidade de uma plataforma distribuída. Os cálculos são feitos baseando-se em medidas da potência computacional dos elementos de processamento e do desvio padrão existente entre essas medidas. O resultado

disto é um índice de desempenho que contempla a heterogeneidade da plataforma e é sensível a qualquer mudança ocorrida (adição ou remoção de elementos de processamento). A partir deste índice é possível prever o impacto causado no grau de heterogeneidade da plataforma devido a uma mudança. Nesse mesmo trabalho é apresentado um estudo baseado em simulação que testa a eficácia de um índice composto por informações de carga de quatro recursos (CPU, memória, disco e rede), no escalonamento de aplicações voltadas a um ou mais recursos.

2.8 Considerações finais

A área de escalonamento de processos continua sendo muito estudada. Isso pode ser constatado pela quantidade de propostas e novos algoritmos encontrados na literatura. Apesar da literatura dessa área estar sempre evoluindo, foi constatado que a maioria dos conceitos básicos, definidos há mais de uma década, ainda continua válida, sendo que muitas publicações clássicas ainda continuam sendo muito referenciadas por trabalhos atuais. A exposição de conceitos deste capítulo teve por objetivo fornecer uma visão geral da área de escalonamento para facilitar o entendimento do restante do conteúdo desta dissertação.

Os algoritmos de escalonamento se propõem a uma grande variedade de objetivos, como: compartilhamento de carga, aumento da utilização do processador, redução do tempo de execução e balanceamento de carga. Para atingir esses objetivos os escalonadores, freqüentemente, precisam utilizar informações sobre a carga de trabalho presente na plataforma. Essas informações são fornecidas pelos índices de carga, que representam o estado de carga de diferentes recursos das máquinas, como CPU, memória e disco.

O escalonamento auxiliado por índices de carga tem se mostrado superior ao escalonamento realizado sem informações, o que reforça a necessidade de se estudar índices de carga. Quando se tem conhecimento sobre as características das aplicações o escalonamento se torna ainda mais preciso. Apesar das aplicações *memory-intensive* serem muito comuns atualmente, pouca atenção foi dada ao estudo de índices de carga voltados a

essas aplicações. Este trabalho visa contribuir nesse sentido, fornecendo opções de índices de carga de memória. O próximo capítulo apresenta os índices de carga de memória identificados no desenvolvimento deste trabalho.

3. Índices de carga relacionados ao uso e à atividade de memória

3.1 Considerações iniciais

Para iniciar a busca por novos índices de carga, dois requisitos básicos foram definidos. Primeiro, os índices deveriam ser facilmente obtidos no sistema operacional Linux no nível de usuário e sem a necessidade de modificações no código do núcleo. Isso faria com que a ferramenta de monitoramento desenvolvida para coletar esses índices se tornasse mais flexível e fácil de instalar. O segundo requisito seria que as informações representadas pelos índices refletissem direta ou indiretamente a utilização ou a atividade da memória.

Foram identificadas 11 informações de carga que satisfazem ambos os requisitos básicos acima. Para fins de organização, foi atribuído um número (de 1 a 11) a cada índice. Por similaridade, as informações foram organizadas em cinco categorias: (1) informações sobre a atividade do processador, (2) informações sobre a utilização da memória, (3) informações sobre a utilização do espaço de *swap*, (4) informações sobre a atividade da memória virtual e (5) ausências de página.

3.2 Informação sobre a atividade do processador

Foi identificada uma informação relativa ao processador que pode refletir indiretamente o estado de utilização do subsistema de memória.

Índice 1. Processos bloqueados

No Linux há seis estados em que um processo pode estar em um determinado momento, identificados pelas letras R, S, D, N, T e Z. O estado R (*running*) significa que o processo está executando (consumindo a CPU) ou aguardando na fila do processador, pronto para executar. Processos no estado S (*sleeping*) estão "dormindo", o que significa que eles estão esperando algum comando ou instrução que reinicie a execução do processo. O estado D (*uninterruptible sleep*) é atribuído a processos que estão bloqueados, geralmente devido à espera de uma chamada de I/O. O estado N identifica processos que estão em baixa prioridade (*nice*). Processos que estão sendo depurados entram no estado T (*traced*). Por fim, o estado Z (*zombie*) indica um processo cujo pai morreu sem ter esperado o filho terminar e sem ter avisado que não pretendia esperar.

O número de processos bloqueados (estado D) é capaz de refletir o estado de utilização da memória indiretamente, já que em situações de uso intenso, muitas operações de I/O podem ser necessárias para viabilizar as operações de troca de páginas. A concorrência, pelas operações de leitura e escrita em disco, pode fazer com que alguns processos sejam bloqueados, fazendo-os entrar no estado D.

3.3 Informações sobre a utilização da memória

Os índices de carga descritos a seguir fornecem informações em termos de quantidade de memória medida em diferentes situações. O objetivo das informações que refletem "quantidade" é mensurar precisamente os níveis de utilização do *espaço* de memória disponível, incluindo a memória principal e o *swap*. Os valores dessas informações são dados em Megabytes.

Índice 2. Memória ocupada

Corresponde à quantidade total de memória principal em uso. Como o Linux fornece apenas a informação sobre a quantidade de memória livre, então para obter esse índice é

necessário utilizar também a informação sobre a quantidade total de memória principal disponível no sistema.

Índice 3. Memória usada em *caches* e *buffers*

Esse índice corresponde à quantidade de Bytes usada pelo sistema operacional para armazenar dados lidos do disco. O sistema operacional contabiliza, através dessas informações, as páginas usadas como *buffers* de escrita e leitura de I/O e o *cache* de páginas.

Os *buffers* de I/O, também chamados de *buffer cache,* são necessários porque, no caso de escritas de I/O, o SO não envia imediatamente para o disco os dados escritos pelas aplicações. Aqui o termo "I/O" refere-se apenas às operações de leitura e escrita no disco rígido. O acesso ao armazenamento externo é mais eficiente quando uma quantidade grande de dados é escrita de uma só vez. Para permitir o acúmulo de dados o SO mantém os dados na memória principal até que seja vantajoso escrevê-los no disco. Os *buffers* também são usados para otimizar a transferência de dados em operações de leitura. Os dados costumam ser lidos do disco em blocos de tamanho geralmente maior do que a porção que um programa precisa em determinado momento. Os dados excedentes ficam armazenados nos *buffers,* já que há uma grande chance de o programa requisitar um dado do mesmo bloco. Isso acontece devido ao princípio da *localidade espacial* (Stallings, 1999).

O *cache* de páginas é usado para armazenar programas executáveis e fragmentos de arquivos acessados recentemente do disco local ou de outro dispositivo de bloco. Depois que são utilizados, esses dados ainda têm uma grande probabilidade de serem usados novamente num futuro próximo. Isso se deve ao princípio da *localidade temporal*. Essa localidade define que os fragmentos de dados lidos tendem a se agrupar no tempo (Stallings, 1999; Love 2004).

No Linux, os *buffers* de I/O e o *cache* de páginas são gerenciados em conjunto, através das mesmas estruturas de dados (Love, 2004).

Índice 4. Memória ativa

Esse índice equivale à quantidade de *Bytes* ocupados por páginas que foram acessadas recentemente, ou seja, é a porção da memória usada que permanece ativa. No Linux, assim como em diversos sistemas operacionais, a política de substituição de páginas é baseada no algoritmo LRU (*Least Recently Used* ou Menos Recentemente Usada). Cada página no sistema possui um atributo que representa a sua idade (*age*). Periodicamente o SO percorre a lista de páginas em uso para atualizar a idade das páginas. O sistema soma um valor constante à idade caso a página tenha sido acessada. Por outro lado, caso a página não tenha sido acessada o valor da idade é dividido por dois. Valores maiores do atributo idade representam que a página foi acessada recentemente e é desejável mantê-la na memória principal. Quando o valor desse atributo chega a zero, significa que a página em questão não foi acessada por vários instantes em que o SO percorreu a lista de páginas. Nesse momento, a página é movida para uma lista de páginas inativas. Portanto, páginas ativas são aquelas cuja idade é superior a 0.

Índice 5. Memória inativa

A quantidade de memória inativa corresponde à porção da memória que está alocada, mas não foi usada recentemente. Esse valor equivale aos *Bytes* ocupados pelas páginas que foram enviadas para a lista de páginas inativas pelo sistema de substituição de páginas. Quando há a necessidade de enviar uma ou mais páginas para o espaço de *swap* devido à escassez de memória, o sistema utiliza a lista de páginas inativas para escolher a página que será retirada da memória. O Linux procura manter a lista de páginas inativas sempre abastecida. A meta é manter uma relação ideal de 2 para 1 entre o número de páginas ativas e inativas, isto é, para cada duas páginas ativas deve haver uma página inativa (Linux Kernel, 2005). Em situações de utilização intensa da memória o algoritmo de desativação de páginas pode ter mais dificuldade para manter esta relação ideal. A cada varredura da lista de páginas ativas o algoritmo é iniciado com um número de páginas que devem ser desativadas. Cada vez

que o algoritmo percorre toda a lista sem que esse número seja atingido, ele aumenta sua prioridade, que vai de 1 (alta) a 6 (baixa). Quanto mais alta a prioridade, maior é a freqüência que o algoritmo é chamado. Isso faz com que, em altas prioridades, as páginas ativas sejam desativadas com mais freqüência, já que sua idade chega a zero mais rapidamente. Assim, mesmo em níveis de utilização intensa da memória, a lista de páginas inativas não fica vazia (Jiang e Zhang, 2005).

3.4 Informações sobre o uso do espaço de swap

Índice 6. Espaço de swap usado

Esse índice mede a quantidade total de espaço de *swap* que está sendo usado por páginas que tiveram que ser retiradas da memória principal, em situações de falta de memória. Esse valor inclui a quantidade de *Bytes* ocupados por páginas contidas no *swap cache*.

Índice 7. Swap cache

A área de *swap cache* é mantida pelo sistema operacional para reduzir o número de acessos ao disco em situações de ocorrência de ausências de páginas. Quando uma ausência maior[2] ocorre, a página que é transferida de volta para a memória não é apagada do espaço de *swap* (no disco), a não ser quando o espaço de *swap* livre se torna escasso. Se a mesma página tiver que ser enviada novamente ao *swap*, o sistema verifica se a página foi modificada e não a envia para o *swap* caso ela não tenha sido alterada enquanto esteve na memória principal. Com isso há uma redução do número de acessos ao disco. O *swap cache* também é útil no caso de páginas que são compartilhadas por mais de um processo. As páginas compartilhadas, que são escritas no *swap,* são mantidas em disco enquanto houver pelo menos um processo usando a página (Gorman, 2004).

[2] Ver seção 3.6 para detalhes sobre os tipos de ausências de páginas.

3.5 Informações sobre a atividade da memória virtual

Essa categoria agrupa informações relacionadas às operações realizadas no conjunto de páginas que compõem a memória virtual do sistema. Essas métricas mensuram a atividade de utilização do sistema de gerenciamento de páginas. O monitoramento de informações de "atividade" visa capturar informações que reflitam o dinamismo das operações de memória realizadas que permitem observar comportamentos que não são possíveis de obter quando se consideram apenas métricas que medem "espaço".

Índice 8. Page stealing

A atividade de "*page stealing*" ocorre quando o *kernel* retira as páginas de *caches* e *buffers* a fim de usá-las em outro propósito. Esse índice é medido em número de páginas por segundo.

Índice 9. Páginas alocadas

Esse índice mede a atividade do sistema de alocação de páginas, isto é, a quantidade de novas alocações feitas por segundo para satisfazer a demanda dos programas.

3.6 Ausências de página

Quando um processo tenta acessar uma página alocada por ele, mas o endereço físico da página não pode ser obtido diretamente da tabela de páginas do processo, ocorre uma ausência de página (*page fault*). Há dois tipos de ausências de páginas: (1) ausências menores e (2) ausências maiores.

Índice 10. Ausências menores

As ausências menores (*minor faults*) são ausências de página que não requerem acesso ao disco. Diferentes motivos podem levar a essa situação. O primeiro motivo refere-se ao primeiro acesso a uma página alocada. Quando o Linux aloca uma área de memória para uma aplicação, os endereços das páginas alocadas não são imediatamente mapeados na tabela de páginas do processo. Quando o processo acessa uma página pela primeira vez, ainda não há

tradução para o endereço requisitado, por isso uma ausência de página é gerada. Contudo, essa ausência pode ser resolvida sem a necessidade de acesso ao disco, gerando, portanto, uma ausência menor. A ação que o sistema operacional toma nesse caso é simplesmente preencher o conteúdo da página requisitada com zeros e apresentar para a tabela de páginas do processo o endereço da página na memória. A partir desse momento, o processo pode ler e escrever na página normalmente (Red Hat, 2003b; Cockcroft, 2005).

A outra situação em que ausências menores ocorrem é devido ao sistema de substituição de páginas. O valor de idade da página, definido pela política LRU, reflete quão desejável é manter aquela página na memória principal. As páginas inativas são as melhores candidatas para serem substituídas caso haja falta de memória principal, já que têm menores chances de serem acessadas num futuro próximo. Quando uma página é marcada para substituição, o mapeamento (endereço virtual para endereço físico) na tabela de páginas do processo que referenciava a página é desfeito. Isso é feito porque o espaço ocupado pela página pode ser utilizado para outro propósito a qualquer momento. Quando um processo tenta acessar uma página que foi marcada para substituição, mas ainda não foi realmente enviada para o espaço de *swap*, o endereço volta a ser mapeado na tabela do processo, a página é movida para a lista de páginas ativas e uma ausência menor é contabilizada (Riel, 2001; Murray e Horman, 2004).

Índice 11. Ausências maiores

Ao contrário das ausências menores, uma ausência maior (*major fault*) ocorre sempre que uma ou mais páginas que residem no espaço de *swap* devem ser lidas do disco para a memória. Isso significa que a página havia sido reutilizada e não se encontrava mais na memória principal.

3.7 Considerações finais

A Tabela 3.1 apresenta um resumo dos índices pesquisados contendo a identificação do índice, a unidade utilizada e uma informação da orientação do valor desse índice. A

orientação do índice define se é um valor maior ou menor que indica que a carga aumenta ou diminui. A orientação pode assumir três valores: (0) indefinido, (1) quanto maior, mais carga ou (2) quanto menor, mais carga. Por exemplo, quanto maior a quantidade de processos bloqueados, mais carregada a máquina pode ser considerada. Por outro lado, quanto menor a quantidade de memória livre, mais carregada a máquina fica.

Tabela 3.1 - Resumo dos índices pesquisados.

Identificação	**Nome**	**Unidade**	**Orientação**
Índice 1	Processos bloqueados	Número de processos	1
Índice 2	Memória ocupada	MB	1
Índice 3	*Cache* e buffers	MB	2
Índice 4	Memória ativa	MB	1
Índice 5	Memória inativa	MB	2
Índice 6	*Swap* usado	MB	1
Índice 7	*Swap cache*	MB	0
Índice 8	*Page stealing*	Páginas por segundo	1
Índice 9	Páginas alocadas	Páginas por segundo	1
Índice 10	Ausências Menores	Ausências por segundo	1
Índice 11	Ausências Maiores	Ausências por segundo	1

O próximo capítulo explica como esses índices podem ser obtidos no sistema operacional Linux e descreve o desenvolvimento de uma ferramenta de monitoramento que coleta os índices e os disponibiliza para políticas de escalonamento.

4. Monitoramento de índices de carga

4.1 Considerações iniciais

A obtenção de índices de carga depende da forma como o sistema operacional disponibiliza as informações sobre o sistema para os usuários. É desejável que haja uma maneira fácil de obter as informações e com pouca sobrecarga. Isso pode ser obtido no Linux através do pseudo-sistema de arquivos /proc, que fornece uma interface amigável para obtenção de estatísticas diversas. A ferramenta de monitoramento desenvolvida neste trabalho utiliza o /proc para obtenção de todos os índices de carga pesquisados. O objetivo inicial dessa ferramenta foi auxiliar os estudos realizados neste trabalho, mas foram inseridas funcionalidades para torná-la capaz de operar como provedora de informações de carga para algoritmos de escalonamento de processos.

Este capítulo trata das questões relacionadas ao monitoramento de informações de carga no sistema operacional Linux. O diretório /proc é descrito, dando ênfase à obtenção dos índices pesquisados neste trabalho. Em seguida é apresentada uma descrição da ferramenta de monitoramento desenvolvida. Uma breve comparação com outros sistemas operacionais também é mostrada.

4.2 Coleta de informações do diretório /proc

O pseudo-sistema de arquivos /proc funciona como uma interface para as estruturas de dados internas do sistema operacional. Ele por ser usado tanto para coletar informações sobre

o sistema como para alterar parâmetros do SO em tempo de execução. Este trabalho faz uso da primeira funcionalidade para coletar estatísticas sobre o comportamento do sistema de gerenciamento de memória. O diretório /proc é considerado um pseudo-sistema de arquivos porque não está armazenado em mídia fixa. Todo o seu conteúdo fica armazenado na memória principal, mas pode ser acessado como arquivos comuns.

O diretório /proc contém um subdiretório para cada processo residente no sistema, nomeado segundo o PID (*process identification*) do processo. Além disso, podem ser encontrados diversos arquivos e diretórios que representam informações do sistema como um todo. A Tabela 4.1 apresenta o nome e a descrição desses arquivos e diretórios. Nem todas as entradas listadas nessa tabela estarão presentes em todos os sistemas. Isso depende da versão e da configuração do núcleo e da arquitetura da máquina. As informações dessa tabela correspondem à versão 2.6 do núcleo instalado em uma máquina com arquitetura x86.

Tabela 4.1 – Arquivos do diretório /proc que contêm informações do sistema como um todo.

Nome	**Conteúdo**
acpi	Configurações de gerenciamento de energia
asound	Configurações das placas de som
buddyinfo	Níveis de alocação de memória (normal, DMA, memória alta).
bus	Diretório com informações específicas sobre os barramentos
cmdline	A linha de comando usada para iniciar o núcleo
cpuinfo	Informações sobre o processador (arquitetura, freqüência, fabricante, etc)
crypto	Configurações de segurança criptográfica
devices	Dispositivos disponíveis (de bloco e de caracteres)
dma	Canais DMA (*Direct Memory Access*) usados
driver	Informações sobre os drivers de dispositivos instalados
execdomains	Relativo à segurança e suporte a formatos binários de outros sistemas

Nome	**Conteúdo**
filesystems	Sistemas de arquivos suportados
fs	Parâmetro do sistema de arquivos
ide	Diretório contendo informações sobre o subsistema IDE (*Integrated Drive Electronics*)
interrupts	Registra o número de interrupções por IRQ (*Interrupt Request*)
iomem	Mapeamento dos endereços de memória associados a dispositivos físicos
ioports	Listagem de endereços associados a portas de comunicações
irq	Diretório de máscaras de IRQ
ksyms	Símbolos usados por módulos para "linkedição" dinâmica de código do núcleo
kmsg	Registro de mensagem de log geradas pelo núcleo
loadavg	Média da carga do processador nos últimos 1, 5 e 10 minutos
locks	Mostra os arquivos travados pelo núcleo
meminfo	Mostra várias informações relativas ao uso da memória
modules	Lista de módulos carregados pelo núcleo
mounts	Lista de todas as montagens ativas
net	Diretório com informações sobre a rede
partitions	Detalhes das partições de disco (tamanho, número de blocos, etc)
pci	Lista de todos os dispositivos PCI (*Peripheral Component Interconnect*) em uso
slabinfo	Utilização de memória no nível *slab* (área reservada ao núcleo)
stat	Estatísticas gerais sobre o sistema (utilização da CPU, atividade de disco)
swaps	Medição do espaço de swap utilizado

Nome	Conteúdo
sys	Diretório onde podem ser feitas modificações em parâmetros do núcleo
tty	Drivers de terminal
uptime	Informações sobre há quanto tempo o sistema está ligado
version	Versão da distribuição, núcleo e gcc (GNU C Compiler)
vmstat	Informações sobre atividade da memória virtual (alocação, ausências, etc)

O diretório /proc é largamente utilizado por diversos utilitários de monitoramento disponíveis para o Linux, como o top, ps e free. O top é um utilitário que monitora em tempo real uma grande diversidade de estatísticas sobre o sistema como um todo e de cada processo separadamente. As informações fornecidas por ele são retiradas de arquivos como meminfo, stat, loadavg, uptime e dos diretórios individuais dos processos rodando.

Todos os índices de carga pesquisados neste trabalho foram obtidos a partir de arquivos que contêm informações do sistema como um todo. Os arquivos contidos nos diretórios de cada processo não foram utilizados. Mais especificamente, foram utilizados os seguintes arquivos: stat, meminfo e vmstat.

O arquivo /proc/stat contém informações gerais sobre a atividade do sistema, sobretudo sobre a atividade de processador. Por exemplo, a utilização do processador em diversos modos (sistema, usuário, ocioso, etc) pode ser obtida para cada processador do sistema. A Figura 4.1 mostra um exemplo do conteúdo desse arquivo. Algumas linhas foram suprimidas por serem muito longas e não representarem informações úteis para este trabalho.

O Índice 1, que contabiliza os processos bloqueados está disponível a partir do atributo `procs_blocked` do arquivo `/proc/stat`, que está destacado em negrito na Figura 4.1. O atributo `procs_running` fornece a quantidade de processos prontos; esse índice não foi incluído na lista dos índices pesquisados, mas é utilizado em comparações com outros índices.

```
william@escrivao:~$ cat /proc/stat
CPU  220 236 245 161347 1018 0 8
cpu0 220 236 245 161347 1018 0 8 0
intr 415799 407667 10 0 0 0 0 3 0 4 1 0 0 110 0 40 0 5033 2931
...
ctxt 140868
btime 1133036288
processes 4601
procs_running 1
procs_blocked 0
```

Figura 4.1 - Conteúdo do arquivo /proc/stat.

```
william@escrivao:~$ cat /proc/meminfo
MemTotal:       514232 kB
MemFree:        426016 kB
Buffers:          4660 kB
Cached:          42800 kB
SwapCached:          0 kB
Active:          50680 kB
Inactive:        25136 kB
HighTotal:           0 kB
HighFree:            0 kB
LowTotal:       514232 kB
LowFree:        426016 kB
SwapTotal:     1084348 kB
SwapFree:      1084348 kB
Dirty:             128 kB
Writeback:           0 kB
Mapped:          44780 kB
Slab:             8356 kB
CommitLimit:   1341464 kB
Committed_AS:    81100 kB
PageTables:        660 kB
VmallocTotal:   507896 kB
VmallocUsed:      2320 kB
VmallocChunk:   505512 KB
```

Figura 4.2 - Conteúdo do arquivo /proc/meminfo

O arquivo /proc/meminfo fornece todas as informações relativas ao uso de memória e do espaço de swap utilizadas na obtenção dos índices descritos nas seções 3.3 e 3.4. A Figura 4.2 mostra um exemplo do conteúdo do arquivo /proc/meminfo capturado de uma máquina com 512 MB (524288 KB) de memória principal total. O valor da quantidade de memória total (MemTotal) representa a quantidade disponível para as aplicações. O valor real apresentado pelo Linux (514232 KB) é um pouco menor do que a quantidade real de memória

física porque não contabiliza a porção reservada para armazenar o código binário do núcleo do sistema operacional (Red Hat, 2003a).

O Índice 2 é baseado em informações retiradas do arquivo `/proc/meminfo`, onde os atributos considerados são `MemFree` e `MemTotal`. O valor desse índice corresponde à diferença entre a quantidade total memória (`MemTotal`) e a quantidade de memória livre (`MemFree`). A soma dos atributos `Buffers` e `Cached` equivale ao Índice 3, que mede a quantidade de memória usada em *caches* e *buffers*. O Índice 4, que representa a memória ativa e o Índice 5, que mede a memória inativa são obtidos a partir dos atributos `Active` e `Inactive`, respectivamente. O Índice 6, que mede a quantidade de swap usado é obtido a partir da diferença entre os atributos `SwapTotal` e `SwapFree`. O atributo `SwapCached` é usado para obter o Índice 7.

O arquivo `/proc/vmstat` fornece diversas informações que contabilizam as atividades do sistema de gerenciamento da memória virtual. Um exemplo do conteúdo desse arquivo é apresentado na Figura 4.3.

Os índices de carga descritos nas seções 3.5 e 3.6 são obtidos a partir de atributos do arquivo /proc/vmstat. Todos os atributos desse arquivo equivalem a uma quantidade acumulada desde que o sistema foi reiniciado, portanto, para obter a taxa de operações por unidade de tempo deve ser calculada a diferença entre uma medição e a medição anterior. O Índice 8, que mede a atividade de *page stealing*, equivale à soma de três atributos: `pgsteal_high`, `pgsteal_normal` e `pgsteal_dma`. A atividade alocação de páginas, representada pelo Índice 9, é obtida a partir da soma dos atributos: `pgalloc_high`, `pgalloc_normal` e `pgalloc_dma`. As ausências de páginas também são obtidas do arquivo /proc/vmstat. O Índice 10, que contabiliza as ausências menores corresponde à diferença entre os atributos pgfault e pgmajfault. O atributo pgmajfault contabiliza as ausências maiores, representados pelo Índice 11.

```
william@escrivao:~/$ cat /proc/vmstat
nr_dirty 16
nr_writeback 0
nr_unstable 0
nr_page_table_pages 165
nr_mapped 11250
nr_slab 2207
pgpgin 50845
pgpgout 11200
pswpin 0
pswpout 0
pgalloc_high 0
pgalloc_normal 205839
pgalloc_dma 1
pgfree 310544
pgactivate 6297
pgdeactivate 0
pgfault 465327
pgmajfault 476
pgrefill_high 0
pgrefill_normal 0
pgrefill_dma 0
pgsteal_high 0
pgsteal_normal 0
pgsteal_dma 0
pgscan_kswapd_high 0
pgscan_kswapd_normal 0
pgscan_kswapd_dma 0
pgscan_direct_high 0
pgscan_direct_normal 0
pgscan_direct_dma 0
pginodesteal 0
slabs_scanned 0
kswapd_steal 0
kswapd_inodesteal 0
pageoutrun 0
allocstall 0
pgrotated 0
nr_bounce 0
```

Figura 4.3 - Conteúdo do arquivo /proc/vmstat

4.3 Implementação de uma ferramenta de gerenciamento de carga

Uma das contribuições deste trabalho é o desenvolvimento de um módulo de gerenciamento de carga capaz de coletar todos os índices de carga citados anteriormente. Além de auxiliar no monitoramento realizado (descrito na seção 5.2), esse módulo pode auxiliar políticas de escalonamento que necessitem de informações de carga para tomar

decisões. O módulo fornece apenas os índices de carga pesquisados neste trabalho, mas o seu projeto foi realizado de forma a permitir a coleta de qualquer informação de carga fornecida pelo Linux. Diversas funcionalidades foram adicionadas a fim de tornar esse módulo o mais flexível e genérico possível.

Esse módulo possui uma forma de configuração que evita a recompilação do código quando se deseja alterar o índice de carga usado ou ajustar parâmetros relacionados aos índices. No momento da inicialização o programa lê um arquivo texto contendo todas as informações necessárias para configurar o sistema. A Figura 4.4 mostra o exemplo do conteúdo do arquivo de configuração do módulo.

```
# ID | Nome | Ativo | Freqüência Atualização | Mudança de estado | Período
EMA | Orientação
#         2                   3         4         5         6         7
#
# Atividade do processador
1         proc_prontos        S         2.0       5         10        1
2         proc_bloqueados     N         2.0       10        5         1
#
# Informação sobre a memória principal
3         memoria_ocupada     S         2.0       5         30        1
4         cache_buffers       N         2.0       5         30        2
5         memoria_ativa       N         2.0       5         30        1
6         memoria_inativa     N         2.0       10        30        2
#
# Informação sobre o swap
7         swap_ocupado        N         2.0       10        10        1
8         swap_cache          N         2.0       10        10        0
#
# Estatísticas de memória virtual
9         page_stealing       N         2.0       5         10        1
10        alocacao_pag        N         2.0       5         10        1
11        ausencias_menores   N         2.0       30        10        1
12        ausencias_maiores   N         2.0       30        10        1
```

Figura 4.4 – Exemplo de conteúdo do arquivo de configuração do módulo.

Cada linha do arquivo corresponde à configuração de um índice de carga disponível para uso, com exceção das linhas que começam com o símbolo “#” consideradas como comentário. São usadas 7 colunas para representar os parâmetros de cada índice. As colunas 1 e 2 fornecem um número identificador e um nome para o índice. As demais colunas são usadas para fornecer as seguintes funcionalidades:

- **Ativação e desativação de índices**

Através da coluna 3 é possível ativar (letra S) ou desativar (letra N) a coleta do índice. Para evitar sobrecarga excessiva causada pelo mecanismo de monitoramento a leitura a partir do /proc e os cálculos necessários são executados pelo módulo apenas quando o índice está ativado. O sistema foi projetado para que a ativação e a desativação também possam ser feitas em tempo de execução, através do envio de um comando para o módulo que irá chamar a função correspondente.

- **Ajuste da freqüência de atualização**

O valor da coluna 4 define o intervalo de tempo, em segundos, entre as atualizações do índice. Um índice atualizado mais freqüentemente pode, potencialmente, fornecer informações mais precisas, mas a sobrecarga gerada seria maior. Pode haver índices para os quais intervalos maiores entre as atualizações sejam suficientes para manter a precisão, evitando assim uma sobrecarga desnecessária.

- **Mudança de estado**

Uma política de escalonamento que esteja recebendo informações de carga do módulo pode informá-lo que deseja receber um índice apenas se o seu valor for alterado em uma determinada porcentagem. Geralmente, mudanças muito pequenas no estado de um índice não fazem diferença na tomada de decisão da política, portando é dispensável que a política tome conhecimento dessa mudança. O valor representado pela coluna 5 indica em qual porcentagem o valor do índice deverá mudar para se considerar uma mudança de estado. Por exemplo, se esse valor for definido como 20, o módulo enviará o índice para a política apenas quando o valor coletado for 20% maior ou menor do que o valor enviado anteriormente. Se essa porcentagem for definida como 0, o módulo enviará o índice após cada coleta, independente da mudança ocorrida.

- **Ajuste do período da média móvel exponencial**

O módulo calcula automaticamente uma média móvel exponencial que dá maior peso uma quantidade de medições definida na coluna 6 das linhas do arquivo de configuração. A média móvel exponencial é um mecanismo empregado para suavizar medições de valores que podem apresentar valores de pico momentâneos. Além disso, ela permite levar em conta valores de medições de um passado recente, que tendem a refletir a carga num futuro próximo (Zhou et al., 1993). Os valores mais recentes ganham mais peso no resultado da média do que os valores mais antigos. Como um dos principais objetivos dos índices de carga é poder prever a carga da máquina no futuro, o uso da média móvel exponencial pode ser visto como uma forma de adicionar qualidade na informação trazida pelos índices de carga.

O espaço de tempo passado considerado na média depende da freqüência de atualização do índice. Por exemplo, se a freqüência for definida com 2 segundos e o período da média for ajustado para 30, o valor da média terá influência de valores de carga ocorridos nos últimos 60 segundos. Caso se deseje obter valores instantâneos, basta definir o valor do período como 1.

O valor da média móvel exponencial de um índice no tempo t é dado pela equação 1, onde v_t é o valor instantâneo no instante de tempo t e N é a quantidade de medições passadas que receberão um peso maior.

$$mme_t = (v_t - mme_{t-1}) \cdot \frac{2}{1+N} + mme_{t-1} \quad (1)$$

- **Definição da orientação do índice**

Como discutido na seção 3.7 cada índice possui uma orientação, que pode assumir três valores: (0) indefinido, (1) quanto maior, mais carga ou (2) quanto menor, mais carga. Essa informação foi incluída no arquivo de configuração porque adiciona um significado útil ao índice e pode ser necessária para definir como a política de escalonamento irá inserir o índice

em um cálculo. Esse atributo é uma característica da métrica, portanto não deve ser configurado de qualquer maneira.

4.4 Comparação com outros sistemas operacionais

Outros sistemas operacionais Unix-*like,* tais como Solaris e FreeBSD também são dotados do sistema de arquivos /proc, mas ao contrário do Linux ele não é destinado a fornecer estatísticas do sistema. Nesses sistemas ele funciona como uma interface de programação através da qual é possível interagir com os processos do sistema (Opensolaris.org, 2005). Originalmente o /proc foi projetado para ser uma interface de comunicação com os processos. Outros sistemas operacionais fornecem algumas informações sobre o estado dos processos no /proc, mas informações sobre o sistema como um todo não estão disponíveis. O aproveitamento desse sistema de arquivos para fornecer informações do sistema é uma abordagem introduzida pelo Linux (Schrock, 2004).

Na maioria dos sistemas operacionais Unix-*like*, informações sobre o sistema são obtidas através do uso de APIs destinadas a este fim. Para os usuários essa tarefa é facilitada devido à grande disponibilidade de utilitários de monitoramento existentes que utilizam essas APIs. Por exemplo, no Solaris é possível utilizar as ferramentas iostat, vmstat e netstat para obter informações sobre as atividades de disco, memória virtual e interfaces de rede, respectivamente. O desenvolvimento de uma ferramenta de monitoramento em outros SOs seria possível através do processamento da saída de utilitários de monitoramento já existentes. Uma alternativa seria utilizar as APIs disponíveis diretamente, contudo, durante o desenvolvimento deste trabalho, não foram encontradas documentações sobre essa abordagem.

4.5 Considerações finais

O desenvolvimento da ferramenta de monitoramento descrita nesse capitulo foi bastante facilitado devido à disponibilidade do pseudo-sistema de arquivos /proc do Linux. Essa forma

de interface permitiu que todas as informações necessárias fossem coletadas como se arquivos comuns estivessem sendo lidos. Essa vantagem é específica do Linux, já que em outros sistemas operacionais Unix-*like* o diretório /proc é utilizado para outras finalidades.

A ferramenta foi projetada para permitir a coleta de 11 índices de carga, mas outros poderão ser adicionados sem dificuldade. Além de auxiliar nos experimentos realizados neste trabalho, essa ferramenta pode servir como provedora de índices de carga para políticas de escalonamento de processos. Isso faz com que a ferramenta desenvolvida constitua uma das contribuições deste trabalho.

O próximo capítulo apresenta uma análise do comportamento dos índices pesquisados que foi baseada num monitoramento realizado com o auxílio da ferramenta descrita neste capítulo.

5. Análise de comportamento dos índices de carga

5.1 Considerações iniciais

A motivação para a realização da análise descrita neste capítulo é estudar o comportamento dos índices de carga pesquisados e entender como eles podem ser utilizados para representar com precisão a carga de memória de uma máquina. Esse estudo também permite avaliar a relação entre alguns dos índices pesquisados.

Para tornar esta análise possível, foi realizado um monitoramento dos índices frente a uma carga de trabalho formada por aplicações *memory-intensive*. Em seguida, o comportamento de cada índice foi analisado individualmente para entender como o valor do índice varia com o andamento da execução da carga de trabalho. Esse entendimento permitiu a descoberta de relações importantes entre os índices e a identificação das vantagens e desvantagens de cada um. O monitoramento citado aqui foi realizado utilizando-se a ferramenta descrita no capítulo 4.

A seção 5.2 deste capítulo descreve a carga de trabalho utilizada e a seção 5.3 apresenta a análise detalhada de cada índice de carga feita a partir dos resultados do monitoramento.

5.2 Monitoramento da execução de uma carga de trabalho

A carga de trabalho corresponde à execução de um conjunto de aplicações em uma única máquina. A máquina utilizada nos testes possui a seguinte configuração: Processador

Intel Pentium 4 2.8 GHz (512KB *cache* L2), 512MB de memória DDR-SDRAM e HD 80GB SATA 7200RPM (1.5GB/s).

Três aplicações que têm como característica a necessidade de usar grandes quantidades de memória foram escolhidas para compor a carga de trabalho. Foram usadas versões seqüenciais das aplicações.

5.2.1 Descrição das aplicações

5.2.1.1 BRAMS

O BRAMS (*Brazilian Regional Atmospheric Modeling System*) é um modelo numérico de previsão de tempo e clima (BRAMS, 2005). Essa aplicação pode ser considerada *CPU-intensive*, *memory-intensive* e *I/O intensive*. Foram criadas duas configurações diferentes, chamadas de BRAMS-1 e BRAMS-2. As dimensões espaciais e temporais do modelo a ser simulado compõem o diferencial das configurações. Um espaço do BRAMS possui quatro dimensões, três para representar a atmosfera e uma para representar o tempo. As dimensões que definem o espaço da atmosfera são: latitude, longitude e altitude. As faixas de latitude e longitude, juntamente com um parâmetro que representa a "resolução" definem a "grade" do modelo.

A primeira configuração possui uma grade de 70 pontos de latitude por 70 de longitude, 32 níveis de altitude e resolução de 20 km. Na segunda configuração foram aumentados os pontos de latitude e longitude que passaram de 70 para 100; as configurações de altitude e resolução foram mantidas as mesmas. O tempo de simulação da primeira configuração equivale a 2 horas de previsão. Na segunda configuração o tempo foi definido como 1 hora, para evitar um tempo de execução muito longo. O objetivo da criação dessas duas configurações foi variar a quantidade de memória utilizada e não, necessariamente, o tempo de execução.

5.2.1.2 GAMESS

O GAMESS é uma aplicação de química computacional, que nasceu da junção de vários outros programas, especialmente o HONDO, o qual implementa diferentes técnicas de química quântica (Schmidt et al., 1993). A principal atividade fornecida pelo GAMESS é a predição quantitativa dos fenômenos químicos obtidos com métodos ab-initio. Essa aplicação possui características *CPU-intensive, memory-intensive* e *I/O-intensive*. A principal causa da alta demanda computacional do GAMESS é o cálculo de integrais, as quais correspondem à operação mais comumente executada pela ferramenta (Matos et al., 2006).

Três configurações dessa aplicação foram criadas para a execução dos experimentos. Cada configuração equivale à execução da aplicação com uma molécula diferente: (1) nonaceno, (2) decaceno e (3) ppv09. As configurações receberam o nome de GAMESS-Nonaceno, GAMESS-Decaceno e GAMESS-ppv09, respectivamente. Essas configurações possuem diferenças tanto na quantidade de memória utilizada como no tempo de execução.

5.2.1.3 PHOLD

O PHOLD (Fujimoto, 1990) é um modelo sintético de simulação de eventos discretos. A implementação foi construída no ambiente de simulação Warped (Martin et al., 2003). O PHOLD é uma aplicação *CPU-intensive e memory-intensive*.

Esse modelo simula a comunicação entre objetos dispostos numa topologia do tipo *torus* de duas dimensões. Para definir o número de objetos definem-se o número de linhas e colunas do *torus*. Nas três configurações as dimensões definidas foram de 200 linhas e 200 colunas, totalizando 4000 objetos na simulação. A simulação consiste basicamente de uma sucessão de trocas de mensagens entre os objetos. Cada objeto comunica-se com quatro outros objetos, que são os seus vizinhos. Para cada mensagem que um objeto recebe, uma única mensagem é enviada, avançando no tempo de simulação. A quantidade de mensagens é

definida como parâmetro dado no início da simulação. A simulação termina quando o tempo virtual atinge o tempo máximo definido.

As configurações criadas para essa aplicação tiveram como objetivo variar o tempo de execução, mantendo as exigências de memória iguais. As três variações do PHOLD são identificadas como PHOLD-1000, PHOLD-1500 e PHOLD-2000, para as quais o tempo virtual máximo foi definido, respectivamente como 1000, 1500 e 2000 unidades de tempo.

5.2.2 Submissão das aplicações

Para facilitar os testes foi criado um executor automático de tarefas que lê as informações de submissão de um arquivo texto. Cada linha desse arquivo contém três campos separados por vírgulas, que são: (1) o intervalo de tempo entre as submissões, (2) o diretório da aplicação e (3) o comando com os parâmetros da aplicação.

O executor é capaz de verificar o estado de utilização de memória da máquina antes de submeter cada aplicação. O valor considerado para definir o estado de utilização equivale à quantidade de memória principal ocupada mais a quantidade de *swap* usado. O objetivo dessa funcionalidade é controlar a carga do sistema durante a execução para impedir estados em que a máquina não suporte a carga e interrompa as aplicações por falta de memória. Esse controle permitiu definir quatro estados de carga em que a máquina foi mantida durante momentos distintos da execução. Nesses estados define-se um valor máximo permitido de quantidade de memória usada. Novas submissões são realizadas somente se a quantidade de memória utilizada está abaixo desse nível.

Para calcular a porcentagem de utilização de memória utilizaram-se quatro índices de carga do conjunto pesquisado. A quantidade de memória principal disponível equivale à diferença entre o Índice 2 (memória ocupada) e o Índice 3 (memória utilizada em *caches* e buffers). Essa combinação é uma métrica para medir com maior precisão a utilização de memória principal ocupada. A mesma será discutida com mais detalhes na seção 5.3.1, na qual é introduzida a métrica M1. A quantidade de *swap* utilizado equivale à diferença entre o

Índice 6 (*swap* ocupado) e o Índice 7 (*swap cache*). Esse cálculo também é uma métrica elaborada para permitir medições mais precisas. Essa métrica, chamada de M4, será discutida na seção 5.3.3.1. A memória total utilizada equivale à soma de M1 e M4. O tamanho da memória principal utilizado no cálculo da porcentagem é de 514232 KB, o que equivale ao total de memória principal disponível para as aplicações. A utilização dessas métricas no controle de carga do executor configura a abordagem incremental empregada nessa análise. Através do estudo do princípio de funcionamento de cada índice, pôde-se escolher índices promissores para criar um mecanismo de controle de carga mais preciso. Isso facilitou a análise dos demais índices avaliados.

A carga completa corresponde à execução de 7 iterações, com níveis de carga variados. Cada iteração corresponde à execução de uma seqüência de 12 submissões, conforme mostrado na Tabela 5.1. Dessa forma, a execução completa equivale à execução de 84 instâncias das aplicações. O intervalo entre as submissões de cada aplicação foi definido como 60 segundos. O mesmo intervalo foi definido entre uma iteração e outra. Esses parâmetros foram definidos com o objetivo de obter uma carga de trabalho significativa e com um tempo total de execução viável.

Tabela 5.1 – Seqüência de 12 instâncias de aplicações que forma uma iteração de submissões

1. PHOLD-1500	2. BRAMS-2	3. PHOLD-1000
4. GAMESS-Nonaceno	5. PHOLD-2000	6. GAMESS-Decaceno
7. BRAMS-1	8. GAMESS-ppv09	9. BRAMS-2
10. BRAMS-2	11. GAMESS-Nonaceno	12. GAMESS-Nonaceno

Para a 1ª e a 7ª iterações foi definido um estado de carga baixa; novas aplicações eram submetidas apenas se a quantidade de memória utilizada fosse menor que 50% da quantidade de memória principal disponível. No segundo estado de carga (2º e 6º iterações), foi permitido que novas aplicações fossem submetidas enquanto o nível de memória utilizada fosse menor

que 80% da quantidade de memória principal. Na 3º e 5ª iterações o uso máximo de memória para possibilitar novas execuções foi definido em 110% da memória principal disponível, permitindo, assim, o uso do espaço de *swap*. Na iteração 5 foi definido que novas aplicações não poderiam ser submetidas caso o total da memória utilizada fosse equivalente a 150% da quantidade de memória principal. A Tabela 5.2 resume a seqüência de iterações definida para a execução, mostrando também a quantidade de memória (em MB) aproximada, acima da qual novas aplicações não poderiam ser submetidas. O total de memória principal disponível é de 512MB.

As aplicações que não puderam ser submetidas, devido ao estado de carga, foram colocadas em uma fila de espera. No próximo intervalo para reavaliação da carga, caso houvesse condições para execução, essas aplicações eram submetidas novamente. Durante a execução do teste a carga foi reavaliada a cada 60 segundos. Intervalos maiores que 60 segundos, poderiam fazer com que a máquina ficasse ociosa, caso as aplicações que estivessem rodando terminassem e outras não fossem submetidas. Por outro lado, reavaliações muito freqüentes poderiam influenciar nas medições devido à sobrecarga gerada na máquina.

Tabela 5.2 – Descrição das 7 iterações de variação de carga usadas durante o monitoramento

Iteração	Percentual de Utilização	Memória permitida
1ª	50%	256 MB
2ª	80%	410 MB
3ª	110%	563 MB
4ª	150%	768 MB
5ª	110%	563 MB
6ª	80%	410 MB
7ª	50%	256 MB

O objetivo principal de limitar o uso da memória em menos de 100% em algumas iterações foi evitar, sempre que possível, o uso do espaço de *swap* para avaliar o uso da memória em estados de utilização moderada ou intensa, mas sem saturação. Nos trechos em que foi permitido atingir níveis de até 150% da quantidade memória principal, o objetivo foi analisar o uso da memória num cenário de saturação da memória principal e uso moderado/intenso do espaço de *swap*.

O objetivo não era garantir que o nível de utilização da memória não ultrapassasse os níveis estabelecidos, já que as aplicações submetidas quando os níveis estavam abaixo do permitido poderiam causar uma maior diminuição na quantidade de memória ou *swap* livre. Independentemente disso, esse controle mostrou-se efetivo em impedir situações de sobrecarga excessiva, que dificultariam uma análise realista do comportamento dos índices de carga em situações normais de uso. O segundo objetivo desse controle de carga foi avaliar o comportamento dos índices nas transições entre estados de baixa e alta utilização de memória.

O Gráfico 5.1 mostra os níveis reais de utilização de memória durante todo o monitoramento. Percebe-se que a porcentagem de utilização ficou acima do nível máximo definido para novas submissões. Por exemplo, na 2ª iteração, na qual o nível máximo foi definido como 80%, a utilização ficou próxima de 90% durante um tempo considerável, atingindo um pico próximo a 100% aproximadamente no instante de tempo 2000. Da mesma forma, na 4ª iteração, o valor total de memória usada chegou a exceder o tamanho da memória principal em aproximadamente 170%.

Assim como o Gráfico 5.1, a área dos outros gráficos mostrados nesse capítulo está subdivida em 7 partes, que representam o intervalo de duração das 7 interações descritas na Tabela 5.2.

O sistema de monitoramento foi configurado para coletar todos os índices de carga juntos em intervalos de 2 segundos. Esse intervalo é freqüente o bastante para detectar as

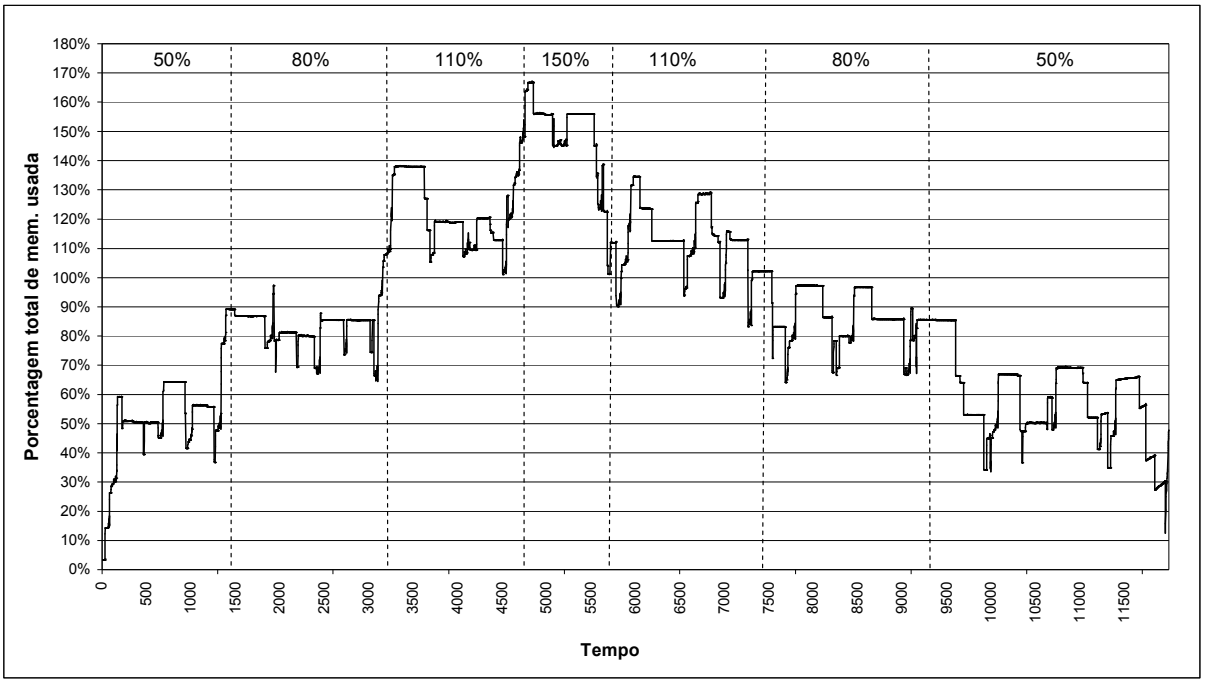

Gráfico 5.1 – Porcentagem total de memória usada.

mudanças de carga da máquina. O tempo de duração total da execução foi de 6 horas e 40 minutos, sendo realizadas, assim, 12000 medições para cada índice.

5.3 Análise dos resultados do monitoramento

Esta seção apresenta os resultados das medições feitas. Foram realizadas análises individuais do comportamento de cada índice para identificar padrões no comportamento dos mesmos. Também, foram feitas comparações entre dois ou mais índices, com o objetivo de identificar relações relevantes entre eles. A comparação de múltiplos índices é especialmente útil, porque em certos casos, foi difícil obter conclusões sobre o comportamento de um índice analisando os valores da medição isoladamente.

Os valores apresentados nesta seção correspondem à media móvel exponencial (*mme*), dando maior peso para 30 medições passadas, o que equivale aos valores coletados em 1 minuto.

5.3.1 Caracterização da quantidade de memória ocupada

A análise das métricas de utilização de memória visa encontrar quais informações de carga fornecem a informação mais precisa sobre a real utilização da memória. No contexto das aplicações *memory-intensive*, é possível dizer que a utilização de memória equivale à utilização do sistema computacional como um todo, porque a memória é o recurso que mais afeta do desempenho dessas aplicações. Isso reforça a necessidade de representar com precisão a carga desse recurso. Foi detectado que nem todas as métricas coletadas fornecem diretamente a informação que aparentam prover, tornando difícil a tarefa de caracterizar com precisão a quantidade real de memória usada. Um exemplo simples disso é o caso do índice fornecido pelo Linux para representar a quantidade de memória ocupada (Índice 2), que apresenta valores que podem não indicar a quantidade *real* de memória principal ocupada. No decorrer da maior parte do tempo de execução esse índice mostra que praticamente toda a memória principal está ocupada.

O Gráfico 5.2 mostra o comportamento do Índice 2 durante todo o monitoramento.

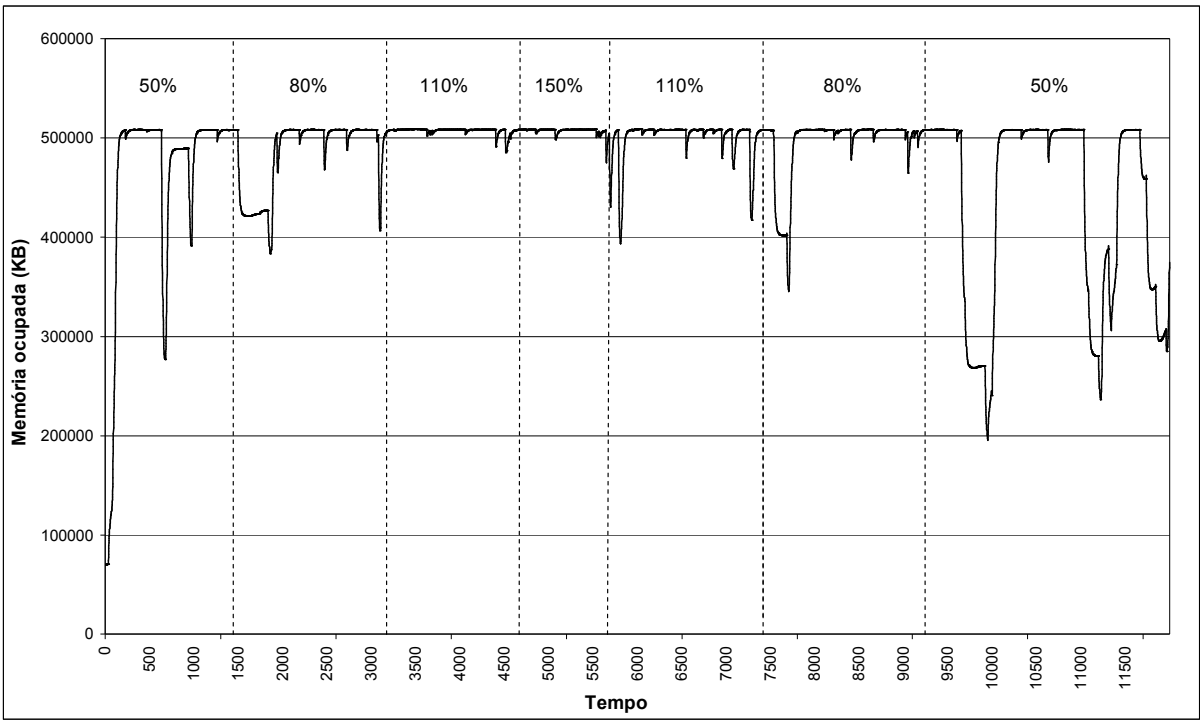

Gráfico 5.2 - Comportamento do Índice 2 (memória ocupada) durante o monitoramento.

Percebe-se que mesmo durante as iterações onde a carga foi mantida baixa, o nível de memória livre aparenta estar sempre baixo. Esse comportamento ocorre devido à ocupação de memória pelo *cache* de páginas e pelos *buffers* de I/O. Como o SO não libera a memória utilizada pelos *caches* e *buffers*, o espaço ocupado não é contabilizado como memória livre, mesmo que nenhuma aplicação esteja usando ativamente as páginas referentes a esse espaço.

Para entender este comportamento, a informação sobre a memória usada em *caches* e *buffers* foi incluída no monitoramento, através do Índice 3. O Gráfico 5.3 ilustra o comportamento do Índice 3 durante toda a execução. Os valores são expressos na forma de porcentagem de memória utilizada em *caches* e *buffers*. Percebe-se que nas iterações em que o limite máximo para novas submissões é mantido em níveis menores (50% e 80%), o valor do Índice 3 é significativamente maior do que nas situações de maior carga (110% e 150%). Por exemplo, quando o nível máximo está em 50%, o valor do Índice 3 chega 50% do total de memória principal disponível e nas interações onde o nível máximo foi mantido em 80% o valor desse índice ficou em torno de 30%. Por outro lado, na iteração de maior carga (150%)

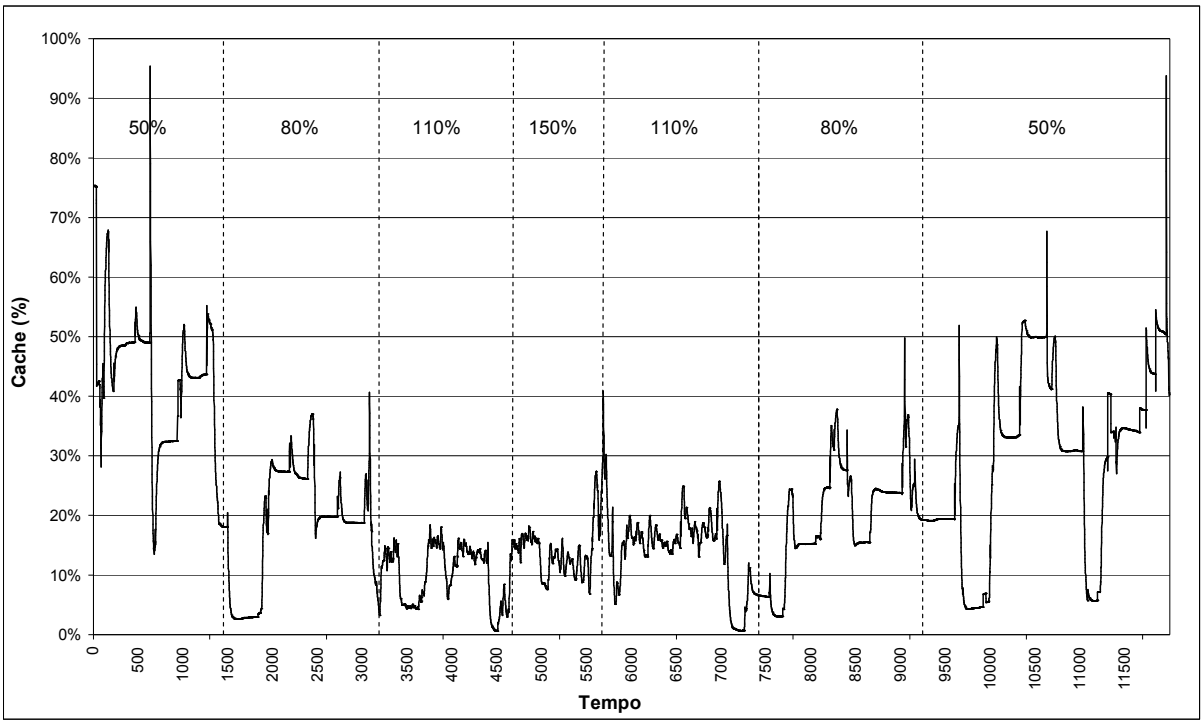

Gráfico 5.3 - Porcentagem da memória principal ocupada por *caches* e buffers

os valores do Índice 3 equivalem a menos de 20% do total de memória principal.

A explicação desse comportamento remete ao funcionamento da política de substituição de páginas. As páginas mantidas na memória como *cache* têm grandes chances de se tornarem inativas porque podem não ser requisitadas novamente. Isso faz com que, em situações de aumento de uso de memória, essas páginas sejam as primeiras a serem liberadas. Da mesma forma, as páginas ocupadas por *buffers* de I/O podem ser liberadas imediatamente quando há falta de memória, já que os dados mantidos nos *buffers* de escrita podem ser enviados definitivamente ao disco.

Uma hipótese a se considerar é o uso como métrica de memória utilizada de um valor equivalente à diferença entre o Índice 2 (memória ocupada) e o Índice 3 (*caches* e *buffers*). Dessa forma, a porção de memória considerada realmente ocupada não contabilizaria a memória ocupada por *caches* e *buffers.* Para analisar essa hipótese é necessária uma

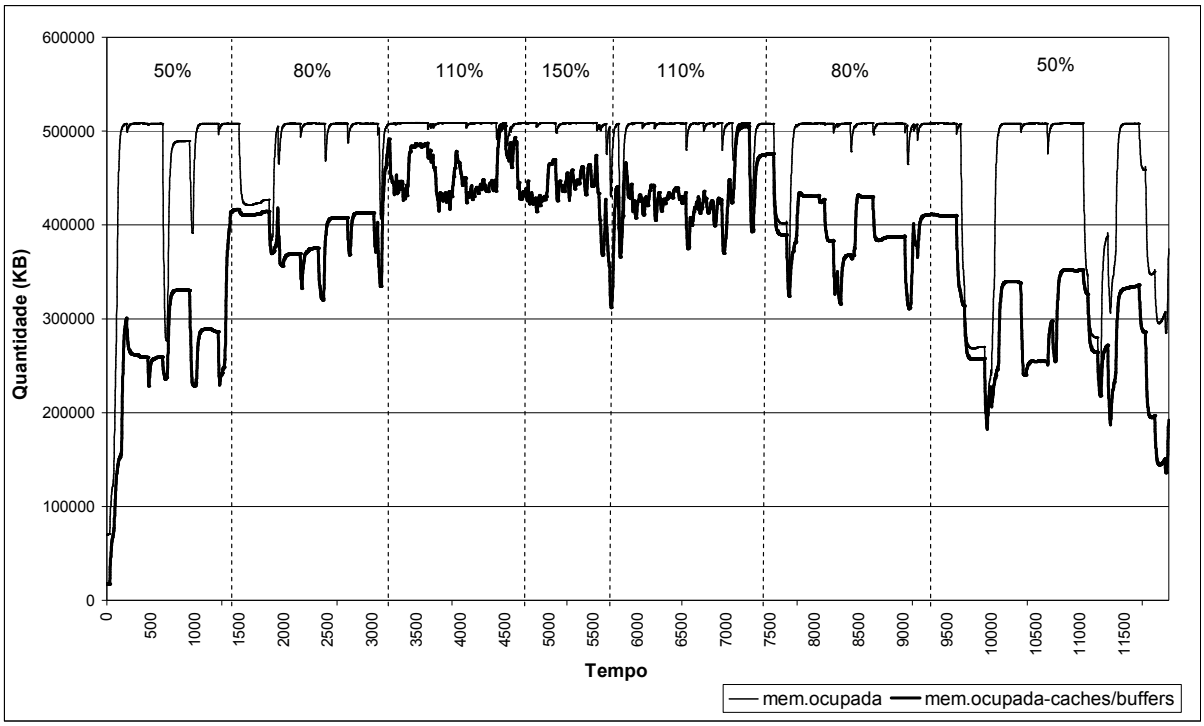

Gráfico 5.4 – Comparação entre a quantidade de memória ocupada incluindo e excluindo as páginas de *caches* e *buffers.*

comparação entre o comportamento dos índices em questão. O Gráfico 5.4 ilustra uma comparação da memória ocupada (Índice 2) e a diferença entre esse índice e o Índice 3, descontando assim, a quantidade ocupada por *caches e buffers*. A comparação desse gráfico mostra que a diferença entre os índices apresenta valores potencialmente mais precisos para representar a memória realmente utilizada, já que é possível diferenciar os níveis de utilização de memória nas diferentes situações monitoradas (iterações). Essa diferenciação não é possível através das informações fornecidas apenas pelo Índice 2.

Através da análise do comportamento do Índice 3, foi constatado que mesmo em situações de uso intenso da memória, o valor desse índice não é reduzido a zero. Isso acontece porque há páginas do *cache* que são requisitadas pelos programas e se mantêm ativas. Dessa forma, não se pode contar que o SO irá liberar essas páginas, retirando-as do *cache* de páginas. Pode-se concluir que, de fato, uma parte das páginas usadas em *cache* pode ser considerada como memória livre. Por outro lado, a porção do *cache* de páginas que está ativa deve ser considerada como memória ocupada. Então, torna-se importante saber a quantidade real de páginas que estão inativas.

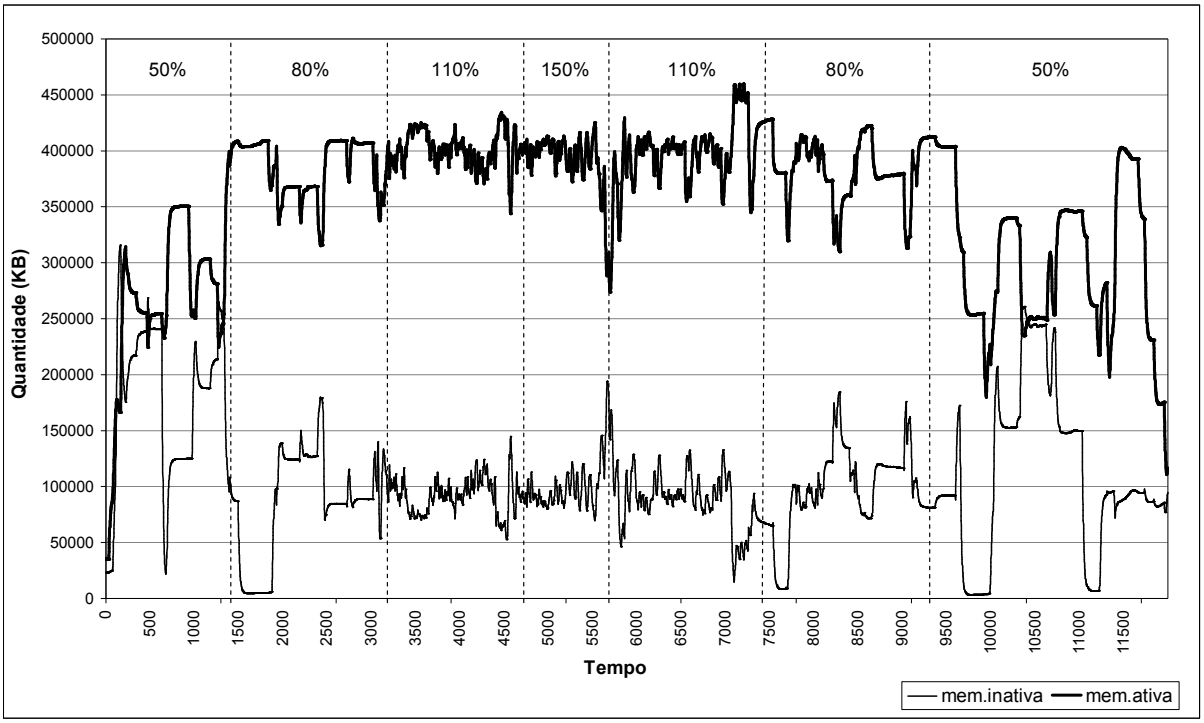

Gráfico 5.5 – Comportamento dos índices de memória ativa e inativa.

O Índice 4 e o Índice 5, que fornecem informações sobre memória ativa e inativa, respectivamente, podem ser usados para fornecer informações ainda mais precisas sobre a real utilização da memória. As páginas inativas formam o conjunto de páginas que serão escolhidas para substituição, assim que houver falta de memória. O Gráfico 5.5 mostra a representação do comportamento desses índices durante todo o monitoramento. Claramente, os valores são complementares, já que uma diminuição no número de páginas ativas, imediatamente causa um aumento no número de páginas inativas. Nota-se que a quantidade de memória inativa é significativamente menor nas iterações de carga elevada, enquanto que a memória ativa aumenta nessas iterações.

A lista de páginas inativas é formada tanto por algumas páginas usadas em *cache/buffers* como por páginas alocadas pelos programas. Portanto, o Índice 5 pode suprir a deficiência do Índice 3 (*cache*/*buffers*), que se mostra incapaz de informar quais páginas podem realmente ser consideradas livres. O Gráfico 5.6 mostra uma comparação de duas métricas elaboradas para representar a quantidade real de memória ativa. A primeira métrica

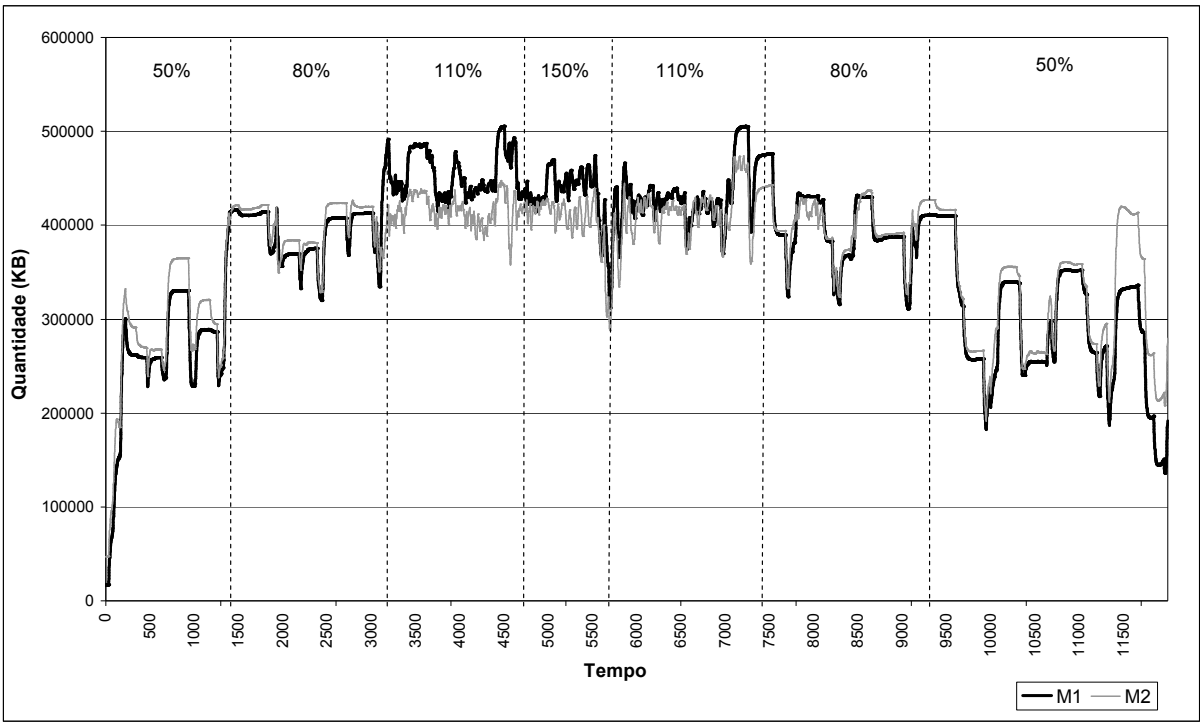

Gráfico 5.6 – Comparação entre métricas propostas para representar a quantidade de memória ocupada.

(M1), corresponde à diferença entre a quantidade total de memória usada e a memória utilizada em *caches* e *buffers*. A segunda métrica (M2) corresponde à diferença entre a quantidade total memória usada e a memória inativa. O objetivo dessa métrica é mostrar o comportamento dos índices caso a memória inativa seja considerada como memória livre.

Tabela 5.3 – Média das diferenças entre as métricas M1 e M2 em todas as iterações

Iteração	Diferença entre M1 e M2 (%)
1 (50%)	M1 15,84% **menor** que M2
2 (80%)	M1 2,5% **menor** que M2
3 (110%)	M1 8,5% **maior** que M2
4 (150%)	M1 7,13% **maior** que M2
5 (110%)	M1 4,01% **maior** que M2
6 (80%)	M1 0,33% **maior** que M2
7 (50%)	M1 10,23% **menor** que M2

Pode ser observado que as duas métricas apresentam valores bastante similares em alguns trechos do monitoramento e valores diferentes em outros trechos. A Tabela 5.3 apresenta as diferenças médias, em porcentagem, dos valores das métricas M1 e M2 em todas as iterações. Uma tendência observada é que nas iterações 1 e 7 (50%) a métrica M1 apresenta valores inferiores a M2; na iteração 1, M1 é 15,84% menor do que M2 e na iteração 7 a diferença é de 10,23%. A média das diferenças é consideravelmente maior na iteração 1 devido a um período de diferenças grandes que ocorre no início da execução das aplicações. Como a máquina utilizada nos testes havia sido reiniciada logo antes, a quantidade de memória inativa ainda era pequena, mas os *caches* e *buffers* já continham páginas utilizadas por aplicativos durante a inicialização do sistema. Nas primeiras 200 medições realizadas, a média das diferenças é de 58,71% e o valor máximo é de 173,44%. Se essas 200 primeiras medições forem ignoradas, a média passa de 15,84% para 8,25%. Nas iterações 3, 4 e 5 (110% e 150%), a tendência se inverte, sendo que os valores de M1 são superiores (até 8,5%

maior). Nas iterações 2 e 6 (80%) não há uma tendência bem definida; na iteração 2 a média de M1 foi apenas 2,5% menor que a média de M1, enquanto que na iteração 6, a métrica M1 foi em média 0,33% maior que M1.

Uma conclusão sobre a caracterização da memória ocupada é que ambas as métricas estudadas (M1 e M2) podem fornecer uma boa estimativa sobre a quantidade real de memória ocupada. Essas métricas são mais úteis do que o Índice 2 sozinho, que apresenta valores sempre altos, mesmo em situações onde a carga de memória foi mantida baixa. Pelo seu princípio, a métrica M2 aparenta ser mais precisa que M1 já que considera como livre as páginas que realmente estão inativas. A métrica M2 considera como livres as páginas usadas por *caches* e buffers, que de fato são os principais alvos do SO quando são necessárias páginas para alocação. Contudo, algumas páginas do *cache* podem permanecer ativas e não vale a pena retirá-las da memória porque uma nova leitura a partir do disco seria necessária. Portanto, essas páginas não devem ser consideradas livres. Se essas páginas forem excluídas da memória principal, pode haver uma penalidade no desempenho caso elas tenham que ser recarregadas a partir do disco.

5.3.2 Métricas para detecção da saturação da memória principal

Quando o espaço de memória principal disponível está todo ocupado, o sistema operacional utiliza o espaço de *swap* para armazenar páginas. Quando isso acontece diz-se que a memória principal está saturada. O termo “saturação” é usado nesse sentido no decorrer deste texto.

Além de saber qual a quantidade real de memória ocupada, é importante haver formas de detectar como o sistema de gerenciamento de páginas está se comportando. Para isso, foram estudadas algumas métricas que têm por objetivo fornecer informações que possibilitem prever o que acontecerá com a memória num futuro próximo. Essas métricas podem ser tanto derivadas de métricas que medem “quantidade” como podem ser formadas pelas métricas que medem a “atividade” do sistema de gerenciamento de páginas.

5.3.2.1 Proporção entre memória ativa e inativa

O Linux tenta manter uma relação de 2 para 1 entre o número de páginas ativas (Índice 4) e inativas (Índice 5), ou seja, o número de páginas inativas idealmente deve equivaler aproximadamente à metade do número de páginas ativas. Foi observado que essa relação não é mantida na maioria das situações, indicando uma incapacidade do sistema de gerenciamento de memória de retomar a proporção ideal. Isso acontece quando o sistema tem dificuldades em encontrar páginas para desativação (Linux Kernel, 2005).

O Gráfico 5.7 apresenta o comportamento de uma métrica (M3) que representa a proporção entre o número de páginas ativas e inativas. Nas iterações de carga elevada (110% e 150%), pode ser observado que o valor da proporção permanece constantemente superior a 4. Durante as iterações de carga moderada podem ser observados os valores mais próximos da relação ideal (igual a 2). Nessas iterações a maioria dos valores observados está entre 1,5 e 3, com exceção de alguns momentos em que o número de páginas ativas é até 100 vezes maior que o número de páginas inativas. Contudo, esses valores altos acontecem com pouca

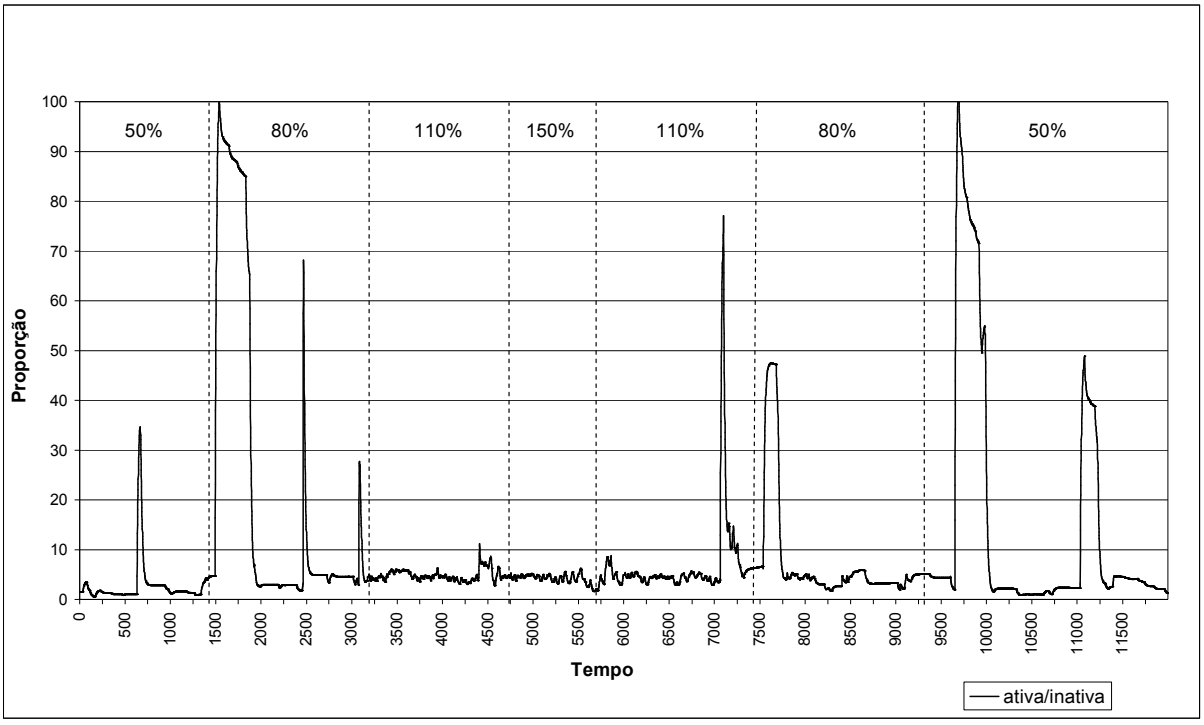

Gráfico 5.7 – Proporção entre os índices de memória ativa e inativa.

freqüência, especificamente em momentos onde muitas páginas são alocadas diminuindo rapidamente a quantidade de páginas inativas.

A partir dos dados analisados, pode-se concluir que a relação entre a quantidade de páginas ativas e inativas é útil para detectar a carga de uma máquina. Se a proporção entre essas duas métricas estiver próxima de 2, pode ser considerado que o sistema operacional não está tendo dificuldades em manter a proporção. Caso o valor permaneça por um longo período acima de 2, a máquina pode ser considerada sobrecarregada.

É importante observar que, de qualquer modo, os picos muito altos da métrica M3 indicam que decisões erradas podem ser tomadas pela política de escalonamento, dependendo de como esse índice for usado por ela. Isso pode ser visto como uma desvantagem, mas que pode desaparecer dependendo da execução de tarefas de refinamento da análise de comportamento do índice. Essas tarefas incluem: (1) determinar os limites acima dos quais os picos poderiam ser ignorados e (2) determinar a partir de qual duração dos períodos em que a carga permanece acima do nível ideal, a máquina deve ser considerada sobrecarregada. Tendo essas tarefas resolvidas, haveria uma forma de desprezar os picos e assim a métrica M3 poderia determinar precisamente se o sistema está mantendo ou não a proporção ideal entre o número de páginas ativas e inativas.

5.3.2.2 Utilização das ausências menores como alerta de saturação

Os resultados do monitoramento mostraram que o índice de carga que mede a quantidade de ausências menores (Índice 10) é capaz de fornecer informações importantes sobre a saturação da memória principal. O valor desse índice aumenta significativamente momentos antes do espaço de *swap* começar a ser utilizado. A detecção do aumento no número de ausências menores funciona como um sistema de alerta, identificando o momento em que possivelmente a memória principal irá se saturar e ausências maiores poderão ocorrer.

O Gráfico 5.8 mostra uma comparação entre as ausências menores e maiores num trecho do monitoramento que compreende as medições de número 2000 a 4000. Nesse trecho

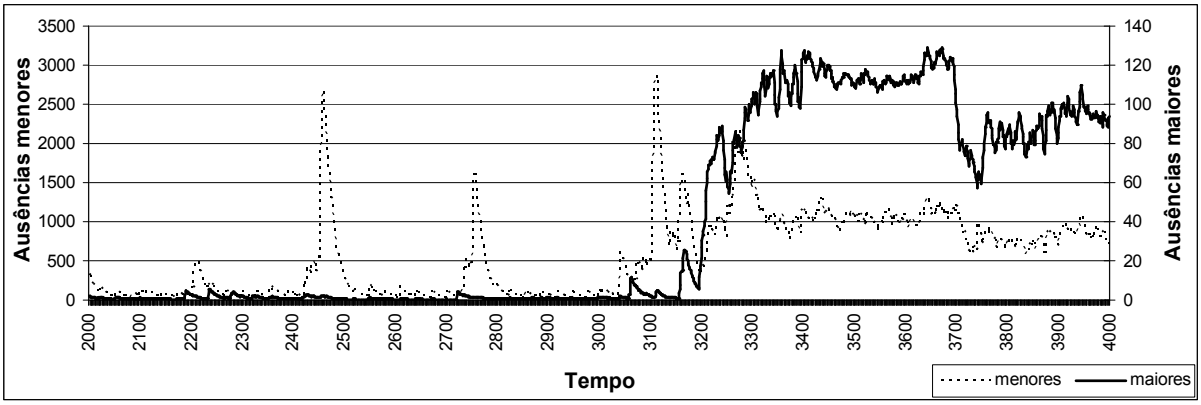

Gráfico 5.8 – Ausências maiores vs. ausências menores.

está incluído o momento de transição da 2ª para a 3ª iteração (instante de tempo 3100), onde a carga máxima permitida para novas submissões passou de 80% para 110%, fazendo com que o espaço de *swap* fosse utilizado.

Foi observado que o nível considerado normal para o Índice 10 é de aproximadamente 100 ausências menores por segundo; durante todo o monitoramento dificilmente o valor desse índice está abaixo de 100. Esse gráfico mostra que o valor do Índice 10 apresenta níveis fora do normal algum tempo antes de o índice que mede ausências maiores (Índice 11) aumentar significativamente. Os níveis de anormalidade das ausências menores iniciam por volta do tempo 3050 quando a taxa de ausências por segundo atinge o valor 500. A partir desse momento esse valor atinge picos de quase 3000 ausências menores e mantém-se superior a 500 até a ocorrência de ausências maiores (saturação da memória principal). Essa saturação é claramente visível no Gráfico 5.8 quando, por volta do instante de tempo 3200, a quantidade de ausências maiores passa de 10 ausências por segundo para 100 ausências aproximadamente. Logo em seguida, esse valor aumenta para aproximadamente 130 ausências por segundo, permanecendo nesse patamar por mais algum tempo.

O Gráfico 5.8 também mostra que alguns picos do Índice 11 ocorrem em outros trechos do monitoramento, por exemplo, por volta dos instantes de tempo 2400 e 2700. Essas ocorrências de ausências menores não estão associadas à ocorrência de ausências maiores, ou seja, a saturação da memória principal não ocorre logo após esses picos. Foi observado que

esses picos possuem duração menor do que os que ocorrem nos períodos que antecedem a saturação. Nesses períodos, ao contrário dos períodos que antecedem a saturação, o nível de ausências menores volta ao valor normal (em torno de 100) após a ocorrência do valor de pico.

Nesse ponto, é possível fazer uma analogia com um semáforo de trânsito, que possui três estados: sinal verde, sinal amarelo e sinal vermelho. Quando o nível de carga de uma máquina está baixo e ela é capaz de receber novos processos para executar considera-se que o *sinal verde* está aceso. Numa situação em que a carga da máquina está atingindo níveis altos e há a possibilidade dela se tornar sobrecarregada, o *sinal amarelo* pode ser aceso, indicando um estado em que as políticas devem escalonar processos levando em conta que a chance de saturação é maior. Por último, o *sinal vermelho* deve ser aceso quando é detectado que a máquina está sobrecarregada; nesse estado, o desempenho de aplicações escalonadas para essa máquina pode ser afetado negativamente.

A detecção de uma quantidade de ausências menores acima do normal funciona como um *sinal amarelo*, indicando que há a possibilidade de ocorrer saturação da memória principal num futuro próximo. A saturação da memória principal após a ocorrência de muitas ausências menores pode ou não ocorrer. Ou seja, nesse caso, após o sinal amarelo, pode se seguir uma transição para o sinal vermelho (detecção de ausências maiores) ou pode se obter um estado de sinal verde novamente.

Mesmo que a ocorrência de um aumento do Índice 10 não implique em uma saturação posterior da memória principal, a utilidade desse índice com um sistema de alerta é evidente. A utilização desse sistema pode auxiliar uma política de escalonamento que tenha por objetivo evitar o uso do espaço de *swap*. Durante o período em que a anormalidade estiver ocorrendo, a política pode decidir não enviar nenhum processo adicional para a máquina em questão. Assim, possivelmente a carga da máquina pode voltar ao normal mais rapidamente. Caso a política permita que mais processos sejam escalonados para a máquina enquanto o

nível de ausências menores está alto (sinal amarelo aceso), haverá uma chance maior de a memória principal dessa maquina se saturar.

5.3.2.3 Utilização das métricas de atividade de memória virtual

Essas métricas indicam os momentos em que há atividade intensa do sistema de alocação de páginas. O Índice 8 (*page stealing*) e o Índice 9 (páginas alocadas) podem fornecer informações que indicam quando esses momentos acontecem. O Gráfico 5.9 contém uma representação do comportamento dos dois índices. Pode-se perceber que as linhas desses índices estão praticamente sobrepostas. Isso significa que praticamente todas as novas alocações feitas foram supridas por páginas retiradas da área de *caches* e *buffers*. Quando não há memória livre disponível, o Linux retira páginas de *caches* e *buffers* para satisfazer os programas. Em situações em que o sistema está constantemente ocupado, há uma chance maior de o espaço ocupado por *caches* e *buffers* ser grande, fazendo com que não haja memória livre e as novas alocações sejam satisfeitas por páginas retiradas deste espaço.

O fato de esses dois índices apresentarem valores muito similares permite que eles sejam analisados como sendo iguais. Uma tendência observada é que a taxa de alocação é maior nas iterações de menor carga (50% e 80%). Em alguns momentos dessas iterações essa

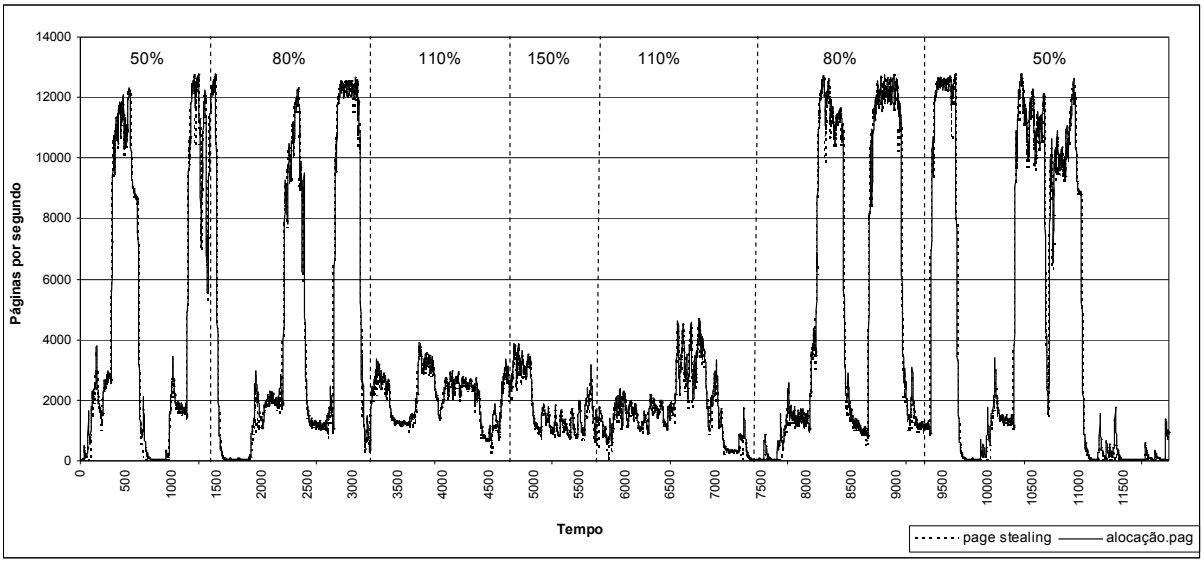

Gráfico 5.9 - Atividade de alocação de páginas vs. atividade de *page stealing.*

taxa chega a 12500 páginas alocadas por segundo. Nas iterações de carga de memória mais elevada (110% e 150%), a taxa de alocação dificilmente passa de 4000 páginas por segundo.

Esses índices podem servir para indicar momentos em que não é recomendado iniciar novos processos em uma determinada máquina. A utilidade desses índices é similar à utilidade do Índice 10 (ausências menores), que pode indicar o momento que o sinal amarelo deve ser aceso. Se o sistema estiver ocupado alocando páginas, o novo processo que chegar pode ser penalizado porque provocará uma maior concorrência. Pode ser recomendado que a política de escalonamento aguarde o término das alocações atuais e verifique se após isso o sistema se tornou saturado. Além disso, um novo processo requisitando memória enquanto há muitas alocações sendo feitas, pode contribuir para que o sistema de memória se sature mais rapidamente.

Os dados do monitoramento indicam que há um limite máximo de alocações por segundo que uma máquina é capaz de tratar. A máquina utilizada nos testes deste trabalho é capaz de tratar a alocação de aproximadamente 12500 páginas por segundo, considerando os valores da média móvel exponencial. O maior valor instantâneo detectado foi de 22110 páginas. Caso o valor do Índice 9 esteja próximo do limite, o sistema pode entrar em um período de alerta. Nesse caso, a política de escalonamento pode decidir não enviar novos processos enquanto o sistema não volta ao estado normal. A determinação do limite deve ser feita para cada máquina que fizer parte do sistema paralelo.

A utilização do limite é válida apenas nos momentos em que a carga da máquina está moderada. Quando a memória já está saturada a quantidade de alocações por segundo fica abaixo do limite máximo que a máquina suporta. Mas isso não significa que a máquina irá suportar alocações de um novo processo submetido. Nas situações de sobrecarga, a taxa de alocações é baixa porque a máquina consegue tratar menos alocações por segundo do que em situações de carga moderada e também porque nesses momentos há menos páginas disponíveis para serem alocadas. Essa constatação indica que a utilização do Índice 9 pode

não ser recomendado para auxiliar política de escalonamento em situações de alta atividade de troca de páginas, já que pode informar um valor de carga baixo enquanto, na verdade, a máquina está sobrecarregada. Uma outra desvantagem percebida é que tanto o Índice 8 como o Índice 9 apresentam informações que aparentam ser de difícil medição, já que apresentam valores de pico muito altos em alguns momentos e valores muito baixos em outros momentos.

5.3.3 Métricas para medir o uso e a atividade do espaço de *swap*

Há diferentes maneiras de se detectar que o espaço de *swap* está sendo utilizado, já que a atividade de troca de páginas reflete em vários índices de carga. O Índice 6, que mede a quantidade de Bytes usada para armazenar páginas no *swap*, tem seu valor alterado sempre que novas páginas são enviadas para o *swap* ou são removidas dele. O Índice 7, que mede a quantidade de Bytes usada pelo *swap cache,* tem seu valor alterado sempre que uma ou mais páginas são transferidas para a memória, mas uma cópia da página continua armazenada no *swap*. Da mesma forma, o Índice 10 (ausências maiores) apresenta valores fora do normal quando o sistema de gerenciamento de páginas está fazendo uso do espaço de *swap* e há a necessidade de transferir páginas da memória para o disco e do disco para a memória. As métricas obtidas a partir destes índices irão auxiliar as políticas de escalonamento indicando o momento em que o *sinal vermelho* deve ser aceso. No decorrer desta seção, será discutido como estes índices cumprem seu papel na detecção da saturação do sistema.

5.3.3.1 Medição da quantidade real de *swap* usado

O Índice 6 mede a quantidade de Bytes referentes às páginas da memória virtual que se encontram armazenadas no espaço de *swap*. Esse índice também contabiliza as páginas que ficam armazenadas no *swap* mesmo que já tenham sido levadas de volta para a memória (*swap cache*). Os Bytes usados pelo *swap cache* são medidos separadamente através do Índice 7. O Gráfico 5.10 mostra uma representação do Índice 6 e do Índice 7 juntos.

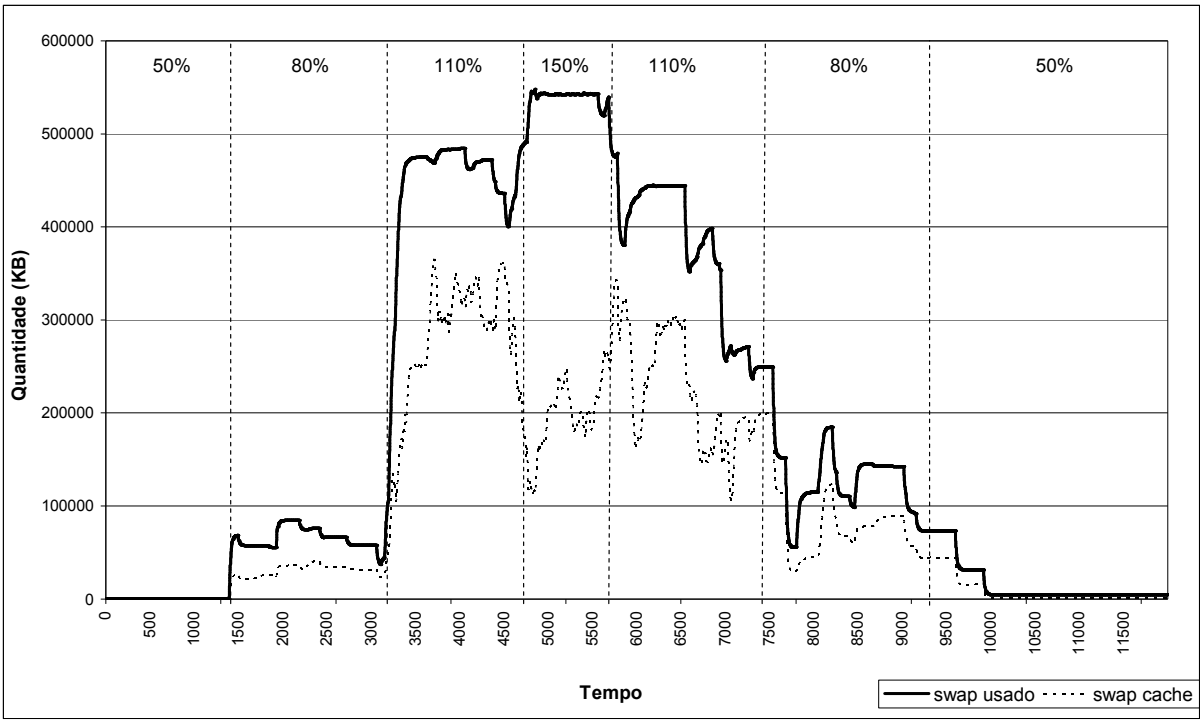

Gráfico 5.10 – Representação dos índices que medem a quantidade total de *swap* usado e a porção do *swap* usada pelo *swap cache.*

O Índice 6 (*swap* usado) apresenta um comportamento que indica pouca variação da quantidade de *swap* usado considerando cada iteração separadamente. O comportamento esperado para esse índice seria acompanhar o dinamismo das aplicações, aumentando e diminuindo conforme a atividade de troca de páginas. Além disso, pode-se perceber que durante todo tempo em que o *swap* está sendo usado, uma parte expressiva do espaço é usada para *cache*. Essa quantidade é visivelmente maior nas iterações 3 e 5 (110%), nas quais em média 60% do espaço utilizado equivale ao *swap cache*. Na iteração 4 (150%), onde a utilização do *swap* é mais intensa uma parte menor (em média 40%) é utilizada pelo *cache*. O Índice 7 (*swap* cache) apresenta uma atividade com variações consideráveis, indicando que, ao contrário do Índice 6, seus valores acompanham o dinamismo das aplicações. Essas constatações indicam que é interessante utilizar o valor do Índice 7, em conjunto com o Índice 6, para encontrar uma forma precisa de medir o uso do *swap*.

As páginas que residem no *swap* e que não estão na memória estão sujeitas a serem transferidas do *swap* para a memória na ocorrência de uma ausência de página. A porção do *swap* ocupada por essas páginas pode ser considerada como a porção realmente utilizada. A parte do espaço do *swap* que corresponde ao *swap cache* pode ser desconsiderada no cálculo do *swap* realmente utilizado. Isso pode ser obtido subtraindo o valor do Índice 7 (*swap cache*) da quantidade total de *swap* usado, representada pelo Índice 6.

O Gráfico 5.11 apresenta a métrica M4, proposta para representar a quantidade real de *swap* usado. Essa métrica equivale à diferença entre o Índice 6 e o Índice 7. Pode-se perceber que a métrica M4 exibe valores mais próximos dos esperados para a quantidade de *swap* usado, acompanhando o dinamismo das aplicações.

Quando se considera apenas o Índice 6 para representar o *swap* usado, não se observa uma diferença significativa na utilização do *swap* entre as iterações 3 e 5 (110%) e a iteração 4 (150%). Isso ocorre porque a porcentagem do espaço de *swap* utilizada pelo *swap cache* é significativamente menor na iteração 4 em comparação com as iterações 3 e 5, fazendo com

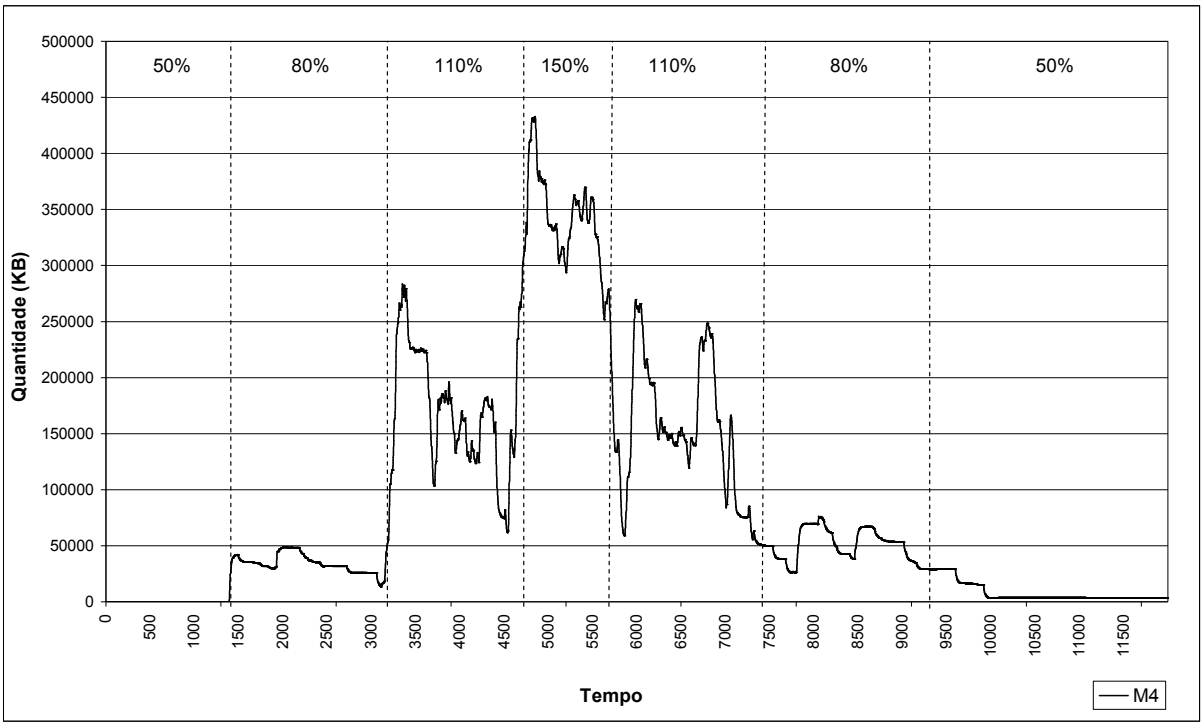

Gráfico 5.11 – Métrica que representa a quantidade real de *swap* usado

que a quantidade total de *swap* usado na iteração 4 fique próxima à quantidade usada nas iterações 3 e 5. Quando se observam os valores da métrica M4, os níveis de utilização nas iterações 3, 4 e 5 estão mais coerentes com as configurações do monitoramento. Nesse caso, a quantidade de *swap* utilizada na iteração 4 é significativamente maior do que a quantidade usada nas iterações 3 e 5, que é o comportamento esperado. A partir dessas observações é possível concluir que a métrica M4 representa com mais precisão o estado de utilização do *swap* e pode ser utilizada por uma política de escalonamento que deseje avaliar a carga de uma máquina baseando-se na quantidade de *swap* usado.

No Gráfico 5.11 observa-se também que uma pequena quantidade de *swap* é utilizada nas iterações 2, 6 e 7. Nessas iterações havia o objetivo de manter a utilização de memória em níveis que evitassem a utilização do espaço de *swap*. Como foi mencionado na seção 5.2.2, não se pôde garantir que o uso memória não ultrapassou os níveis máximos definidos para novas submissões. Contudo, a atividade de troca de páginas não foi significativa, o que pode ser visto no Gráfico 5.12, que representa a ocorrência de ausências maiores na iteração 2. Nesse gráfico também é mostrado o final da iteração 1 e o início da iteração 3. Verifica-se que

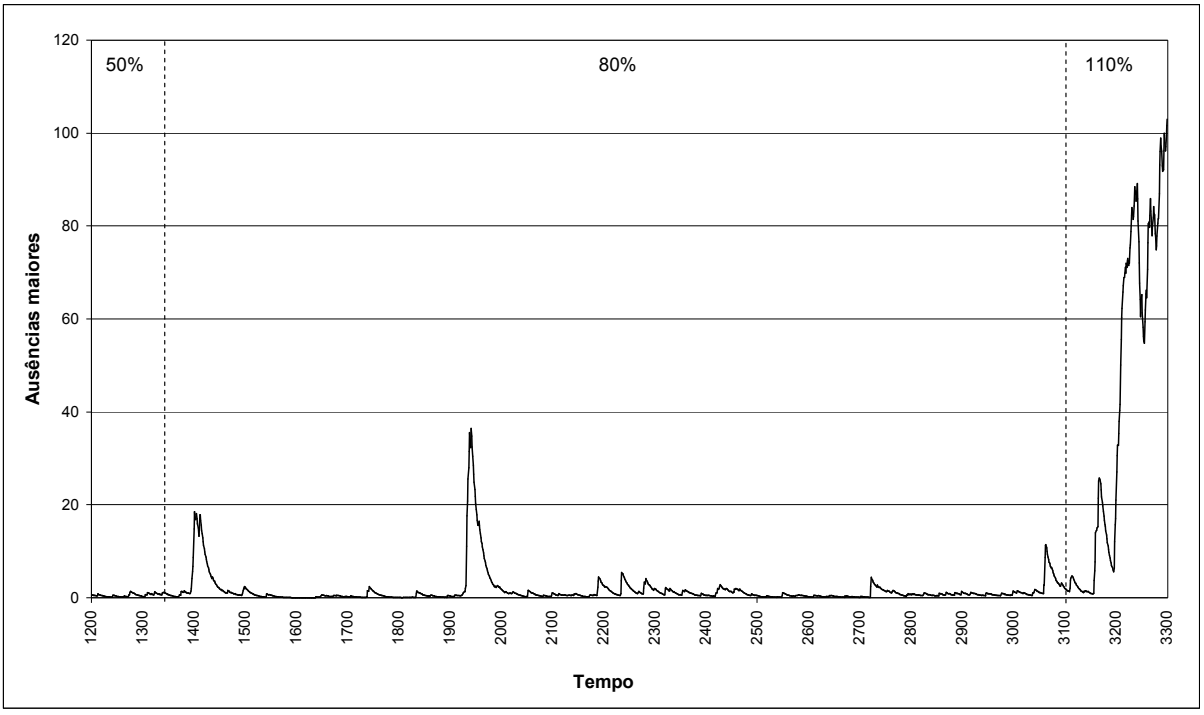

Gráfico 5.12 – Ausências maiores durante a segunda iteração.

há apenas dois períodos de curta duração em que ocorrem ausências maiores na iteração 2. Um aumento mais significativo na ocorrência de ausências maiores é percebido apenas no início da iteração 3. Mesmo assim, percebe-se, pelo Gráfico 5.11 que certa quantidade de *swap* permanece sendo utilizada durante toda a iteração 2. O mesmo comportamento é válido para a iteração 6; a métrica M4 indica que o *swap* se torna livre apenas no início da iteração 7.

Esse comportamento acontece porque algumas páginas transferidas para o *swap* não são requisitadas pelos programas e não voltam para a memória durante o período observado. Essas páginas podem pertencer a alguns processos do sistema operacional que possuem pouca atividade e tiveram suas páginas mantidas no *swap* por um longo período. A porção do *swap* ocupada por páginas que têm pouca chance de voltar à memória não deveria ser considerada realmente utilizada. Uma desvantagem da métrica M4 é que ela não detecta esse tipo de comportamento. Nas pesquisas realizadas neste trabalho, não foi encontrado um índice de carga disponibilizado pelo Linux que auxilie na detecção desse comportamento específico. Contudo, foram encontradas informações de carga que representam a atividade de ausências de páginas. Essas informações podem fornecer estatísticas mais precisas sobre o estado real de utilização do *swap*, do que as métricas baseadas em quantidade de *swap* usado.

5.3.3.2 Utilização da taxa de ausências maiores para detecção de *thrashing*

Os dados analisados na seção 5.3.3.1 indicam que medir a quantidade exata de *swap* usado pode não oferecer muitas vantagens para as políticas de escalonamento. Em certos casos pode ser suficiente saber apenas se há atividade de *swap* ou não. Quando é detectada alguma atividade intensa de *swap*, a máquina pode ser considerada sobrecarregada independente da quantidade de espaço de *swap* usado e não vale a pena enviar qualquer novo processo para ser executado nela. Em experimentos com diversas aplicações, realizados por Xiao et al. (2002), foi mostrado que quando o uso da memória excedia em aproximadamente 5% o tamanho da memória principal, a fatia do tempo de CPU usada para tratar ausências maiores era de 37%. Quando o uso passava para 25% e depois para 65%, o tempo para tratar

as ausências passava para 43% e 91%, respectivamente. Isso evidencia a necessidade de impedir o uso intenso do *swap*, porque mesmo que o uso de memória ultrapasse o tamanho da memória principal em uma pequena porcentagem, o tempo utilizado para realizar computações úteis é reduzido consideravelmente.

Mais importante do que saber a quantidade exata de *swap* usado é poder detectar se está ocorrendo *thrashing* (situação na qual grande parte do tempo de processamento é gasto com operações de troca e pouca computação útil é realizada). A questão nesse caso é sobre quais informações usar para detectar o *thrashing*.

A informação sobre as ausências maiores (Índice 11) pode ser considerada uma boa métrica para representar a atividade do sistema de memória porque indica o nível de intensidade em que os programas em execução estão competindo pela memória principal disponível. Um alto nível de competição acontece quando os programas não conseguem estabelecer o seu conjunto mínimo de páginas de trabalho (*working set*) para poder executar alguma computação útil, causando *thrashing*. Sabe-se que o *thrashing* é o principal responsável pela queda do desempenho de aplicações que dependem de grandes quantidades de memória, como foi discutido na seção 2.5.

O Gráfico 5.13 apresenta o comportamento do Índice 11 durante todo o monitoramento. O comportamento desse índice indica que a taxa de ausências maiores por segundo aumenta significativamente logo que a utilização de memória ultrapassa o tamanho da memória principal, iniciando o uso do espaço de *swap*. Isso ocorre no início da iteração 3, quando o monitoramento foi configurado para tolerar novas submissões de aplicações enquanto a utilização de memória estivesse abaixo de 110% do tamanho da memória principal. Esse comportamento é esperado já que o espaço de memória alocado pelas aplicações não é comportado pela memória principal. Nesse caso, o uso de memória por aplicações que estão trabalhando ativamente resulta na ocorrência de ausências maiores.

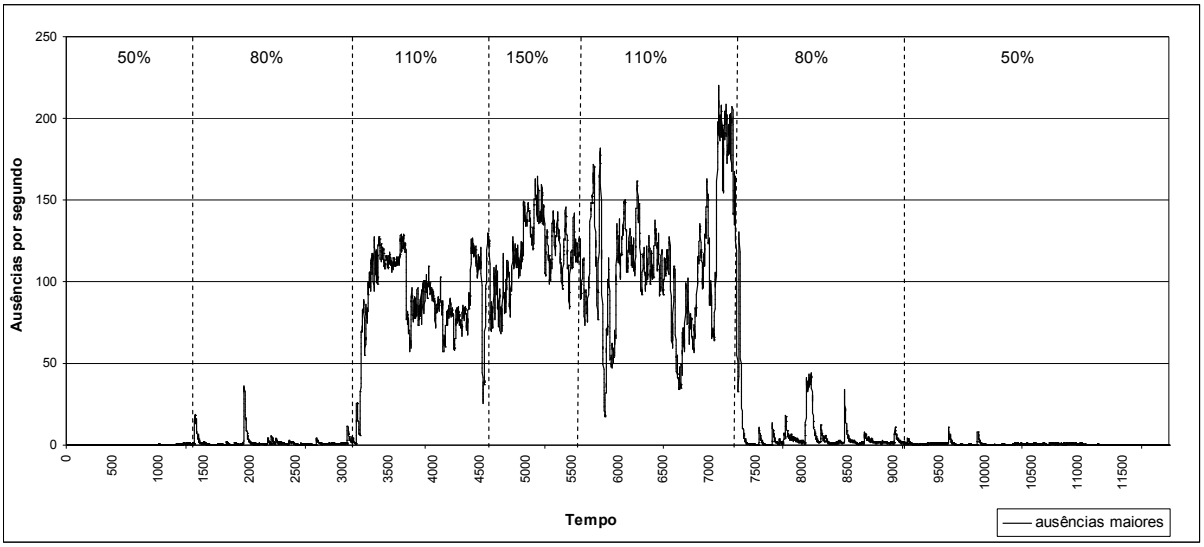

Gráfico 5.13 – Índice que representa a ocorrência de ausências maiores.

Nota-se que logo que a utilização do espaço de *swap* inicia, o nível de ausências maiores atinge níveis consideravelmente altos. Os níveis na iteração 3, ficam em torno de 100 ausências por segundo na maioria do tempo. Esses níveis são menores do que os níveis da 4ª iteração, na qual ocorrem até 150 ausências maiores por segundo. Contudo, essa diferença não é expressiva em comparação com a diferença observada na quantidade de *swap* usado, mostrada anteriormente no Gráfico 5.11. Era esperado que a taxa de ausências maiores por segundo fosse maior na 4ª iteração, do que nas iterações 3 e 5, nas quais o uso de *swap* foi menor. Mas é na 5ª iteração que essa taxa atinge o maior valor observado durante o monitoramento. No instante de tempo 7250 a taxa chega a aproximadamente 210 ausências por segundo, considerando a média móvel exponencial. O maior pico instantâneo detectado foi de 450 ausências maiores tratadas em um segundo. Esses dados indicam que, no caso do monitoramento realizado neste trabalho, o comportamento desse índice depende *mais* das características das aplicações e da interação entre elas e *menos* do nível de utilização do espaço de *swap*. Isso explica o fato de a taxa de ausências maiores atingir o valor máximo na iteração na qual a utilização de *swap* é menor.

Durante as iterações 2, 6 e 7 podem ser detectados pequenos valores acima de 0 na taxa de ausências maiores. Isso se deve a alguns valores instantâneos que aparecem várias vezes

após a primeira vez em que o *swap* é usado na iteração 2. Como o espaço de *swap* utilizado nessas iterações provavelmente contém páginas pouco requisitadas, a quantidade de ausências maiores causadas é pequena. A média móvel exponencial nessas situações fica geralmente abaixo de 1. Dessa forma, conclui-se que há um nível de ausências maiores que pode ser tolerado. Isso também indica que o Índice 11 não deve ser usado isoladamente e sim, em conjunto com outros índices. Por exemplo, quando outras métricas indicam que o sistema está no sinal verde, a taxa de ausências maiores, que será baixa, poderá ser ignorada. Por outro lado, num momento em que o sinal amarelo estiver aceso, um aumento nos valores do Índice 11, indicará que o sistema está ficando realmente sobrecarregado. Nesse caso, ocorrerá a transição para o sinal vermelho.

A principal conclusão da análise do Índice 11 é que um valor alto no número de ausências maiores é um indicativo de que está ocorrendo *thrashing*. Quanto mais próximo esse índice estiver do limite máximo que a máquina suporta, maior será o tempo gasto pelo sistema para tratar as ausências maiores. O limite máximo é um valor dependente das características do hardware da máquina, sobretudo do desempenho do disco rígido. Portanto, não é possível atribuir um valor de referência válido para qualquer máquina. Para obter esse valor, uma análise deve ser feita para cada máquina do sistema heterogêneo, de forma a descobrir qual é a taxa máxima de ausências por segundo que o sistema de gerenciamento de memória consegue tratar. Por exemplo, os dados do monitoramento realizado neste trabalho indicam que a máquina utilizada nos testes é capaz de tratar no máximo 450 ausências em um segundo, que foi o valor máximo detectado.

5.3.3.3 Detecção de *thrashing* através das filas de processos

O Índice 1, que mede a quantidade de processos bloqueados devido ao excesso de operações de I/O, também pode ser útil na detecção da ocorrência de *thrashing*. Quando a memória principal está saturada e ocorrem muitas ausências maiores, o sistema não é capaz de tratar todas as operações de I/O ao mesmo tempo, causando o bloqueio de alguns processos. O

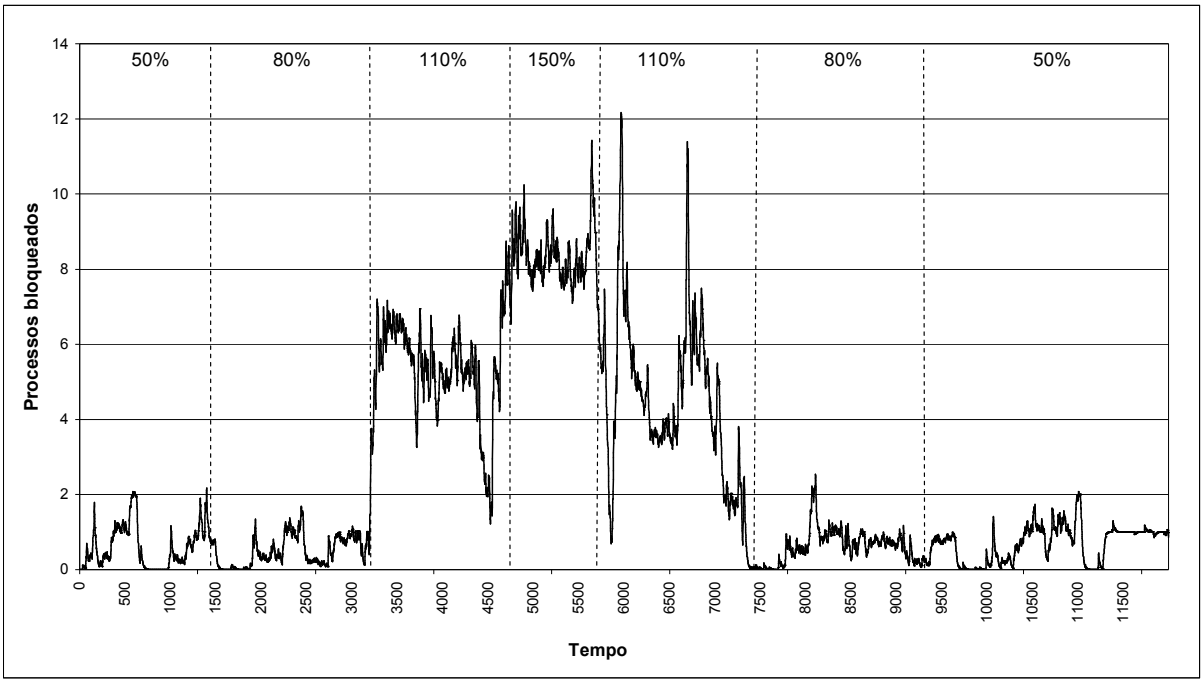

Gráfico 5.14 - Comportamento do índice que mede a quantidade de processos bloqueados

Gráfico 5.14 apresenta o comportamento do Índice 1 durante todo o monitoramento. Esse gráfico mostra que a quantidade de processos bloqueados é maior nas iterações 3, 4 e 5, nas quais o uso de *swap* é intenso. É interessante observar que a quantidade de processos bloqueados é significativamente maior na iteração 4, em comparação com as iterações 3 e 5. Na iteração 5 podem ser percebidos dois picos mais altos desse índice, mas na média os valores são similares aos valores da iteração 3.

Esse comportamento não havia sido observado com relação ao Índice 11, que mede as ausências maiores. Nesse caso, as diferenças entre as iterações 3, 4 e 5 não eram expressivas. Quando o nível de ausências maiores encontra-se em seu valor máximo, qualquer novo processo colocado para executar na máquina tem grandes chances de contribuir para o aumento do tamanho da fila de processos bloqueados. O fato do Índice 1 representar esse aumento de carga pode ser visto como uma vantagem a favor desse índice. Caso seja necessário diferenciar a carga de duas máquinas sobrecarregadas, a máquina cuja fila de processos bloqueados foi maior poderá ser considerada mais sobrecarregada.

O tratamento de ausências maiores não é a única causa do bloqueio de processos. A disputa por acessos ao disco ou à rede feitos diretamente pelas aplicações também pode

causar bloqueios. Isso pode ser visto através do Gráfico 5.14 com atenção especial às iterações 1, 2, 6 e 7. Nessas iterações o Índice 1 apresenta uma atividade significativa, mas a atividade de *swap* é insignificante. Mesmo quando o sistema de memória da máquina não está saturado, mas há vários processos bloqueados, os novos processos que chegarem, podem sofrer uma penalidade caso necessitem executar operações de I/O ou acessos à rede de comunicação.

O Índice 1 (processos bloqueados) aparenta ser um bom índice de carga para o escalonamento de aplicações *memory-intensive* e *I/O-intensive*. Contudo, algumas informações adicionais seriam necessárias para acrescentar mais significado a esse índice. Por exemplo, assim como em outros casos, a determinação de limites é uma tarefa necessária. Os dados do monitoramento sugerem que quando esse índice está abaixo de 2, a máquina não está sobrecarregada. Mas esses dados dependem das peculiaridades do monitoramento.

Sozinho, o Índice 1 não é capaz de fornecer informações que permitam avaliar a ocorrência de *thrashing*. Para permitir avaliar a quantidade de computação útil realizada, foram coletadas durante o monitoramento informações sobre o tamanho da fila de processos prontos. Esse índice não foi incluído inicialmente na lista de índices pesquisados porque não reflete o uso de memória. A fila de processos prontos corresponde aos processos que, de acordo com o Linux, se encontram no estado R (*running*) e estão realmente consumindo o processador. Essa fila é um dos índices de carga mais utilizados por sistemas encontrados na literatura e geralmente é descrito como um bom índice para representar a carga de máquinas de plataformas distribuídas (Shivaratri et al., 1992). Esse índice é mais usado por políticas de escalonamento e balanceamento de carga quando o foco são as aplicações que dependem mais do processador, mas freqüentemente é utilizado de maneira genérica no escalonamento de qualquer tipo de aplicação.

O Gráfico 5.15 mostra a representação da fila de processos prontos durante todo o monitoramento. Os valores apresentados indicam um padrão bem definido no comportamento

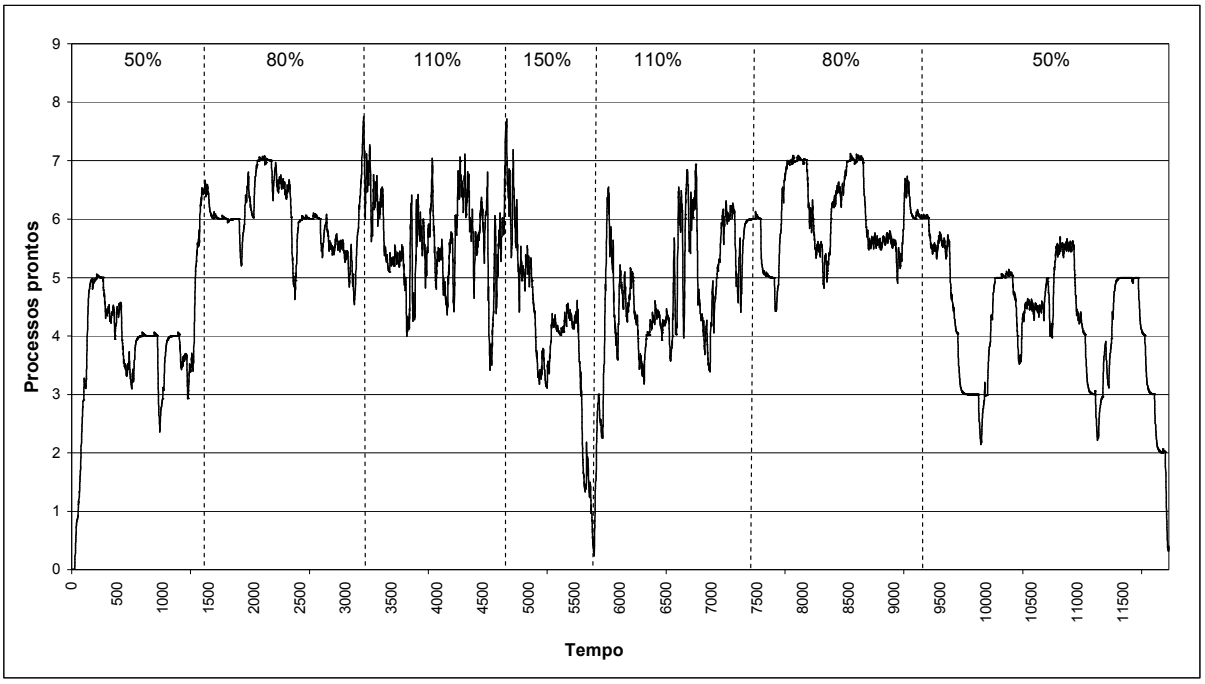

Gráfico 5.15 - Fila de processos prontos.

desse índice. Quando ocorre a transição da 1ª para a 2ª iteração, percebe-se um aumento no nível de processos prontos. Quando o monitoramento passa para a 3ª iteração esse nível permanece estável, não sendo percebido aumento ou diminuição significativa. Por outro lado, quando ocorre a transição para a 4ª iteração pode ser percebida uma diminuição no número de processos prontos. Os comportamentos das iterações 5, 6 e 7 se aproximam dos comportamentos das iterações 1, 2 e 3, respectivamente. Uma comparação dos comportamentos da fila de processos prontos com o Índice 1 (processos bloqueados) ajuda a entender esse comportamento.

O Gráfico 5.16 mostra com detalhes um trecho do monitoramento que compreende o período entre os instantes de tempo 4000 e 7000. Esse gráfico foi elaborado para comparar as filas de processos prontos e bloqueados. O objetivo é entender porque a fila de processos prontos diminui quando a carga de memória aumenta. O gráfico mostra que os momentos em que a quantidade de processos prontos diminui coincidem com aumentos da quantidade de processos bloqueados. Durante a iteração de maior carga (4) e no início da iteração 5, a fila de processos prontos é significativamente menor do que a fila de processos bloqueados. Por

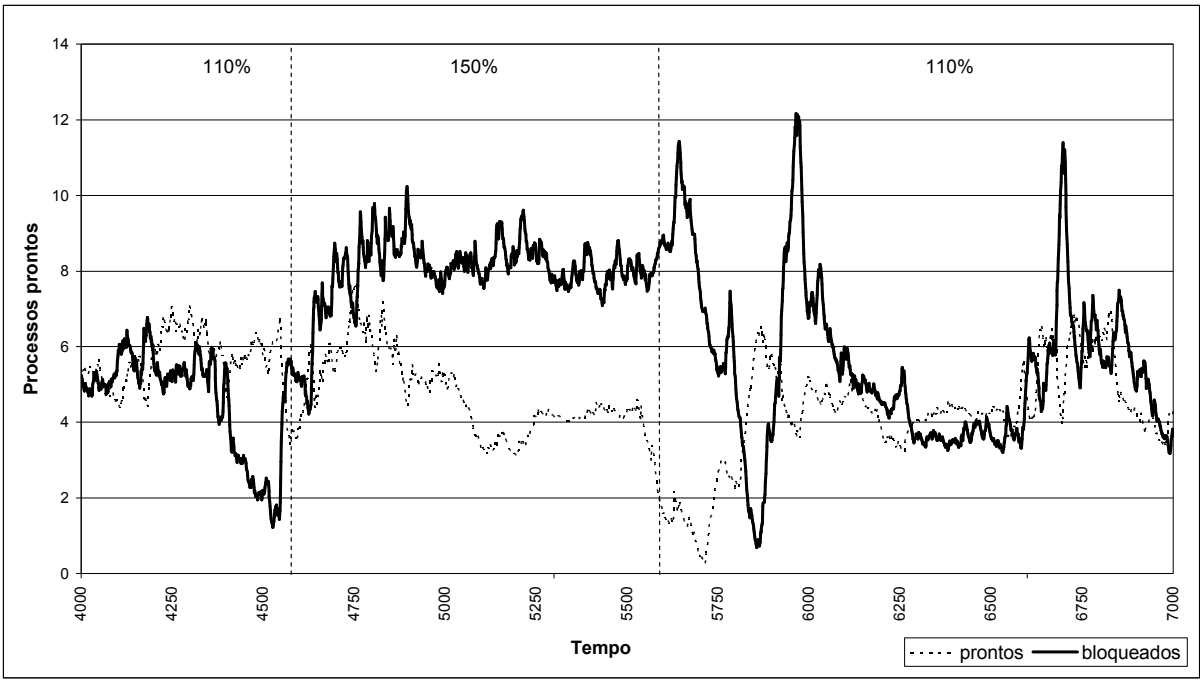

Gráfico 5.16 - Processos prontos vs. processos bloqueados.

exemplo, por volta do instante de tempo 5700, a quantidade de processos prontos chega próxima a zero enquanto há aproximadamente 12 processos bloqueados. Nos momentos em que a utilização do *swap* é muito intensa, a maioria dos processos é incapaz de rodar (consumir o processador) e permanecem bloqueados. Isso é um sinal de diminuição da quantidade de computação útil e de aumento de tempo gasto com tratamento de operações de I/O.

Nas outras iterações, nas quais a carga é menor, essa tendência não é observada. Nas iterações em que não há atividade de *swap*, a quantidade de processos bloqueados é sempre menor do que nas iterações onde o *swap* é usado ativamente. Na iteração 3 e em parte da iteração 5, onde o uso de *swap* foi menos intenso, os tamanhos das duas filas ficaram em nível similares. Nesses períodos, a quantidade de processos prontos caiu abaixo do número de processos bloqueados menos freqüentemente.

A partir da relação entre o tamanho dessas duas filas de processos pode ser criada uma nova métrica de medição de carga. Essa métrica, chamada de M5, equivale à divisão entre o Índice 1 (processos bloqueados) e o tamanho da fila de processos prontos. O valor de M5 será maior do que 1 quando houver mais processos bloqueados do que processos prontos e menor

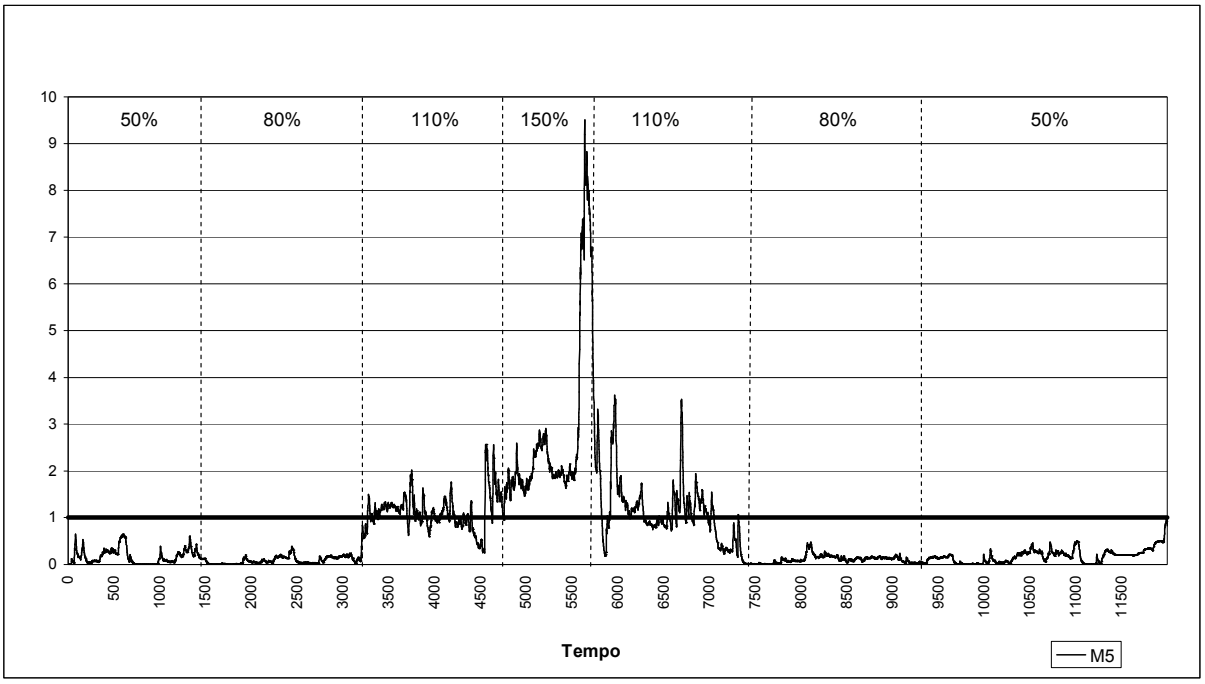

Gráfico 5.17 - Representação da métrica M5.

do que 1 no caso oposto. Dessa forma, inicialmente o número 1 é o limiar de carga definido para essa métrica. No caso de o divisor (processos prontos) ser igual a zero, o resultado da divisão poderá ser o próprio dividendo (processos bloqueados), ou algum valor maior, por exemplo, o dobro do dividendo. Contudo, durante o monitoramento realizado, não houve nenhum momento em que havia pelo menos um processo bloqueado e o número de processos prontos era igual a zero. O comportamento da métrica M5 durante todo o monitoramento pode ser visto no Gráfico 5.17.

A hipótese levantada aqui é que essa métrica pode indicar os momentos em que o sistema está gastando mais tempo no tratamento de operações de I/O do que com computações úteis. Isso indicaria o momento em que o sinal vermelho deveria ser aceso. O limiar de carga também poderia ser definido para valores menores do que 1, deixando essa métrica mais conservadora, ou para valores maiores do que 1 para deixar a métrica mais otimista. Esses valores dependem das necessidades das políticas de escalonamento que fariam uso dessa métrica.

A métrica M5 possui uma vantagem clara em relação à utilização da fila de processos prontos. Apesar de esse índice ser muito usado na prática, ele apresentaria uma visão errada

da carga do sistema na maior parte do monitoramento, sobretudo nos trechos onde foi detectada atividade de troca de páginas. Por exemplo, quando a atividade de *swap* está intensa, a fila de processos prontos indicaria que a carga do sistema está diminuindo, enquanto na verdade ela está aumentando. Isso deixa claro que a fila de processos prontos não é um bom índice para avaliar a carga nessa situação.

5.4 Considerações finais

Este capítulo apresentou a análise do comportamento dos índices de carga pesquisados. As vantagens e desvantagens de cada índice foram apresentadas. Várias relações entre os índices foram descritas possibilitando a criação de cinco métricas originadas da combinação entre índices. Algumas vezes, as desvantagens de um índice desapareceram através da junção com outro índice.

Os índices de carga e métricas têm o objetivo de estimar a carga de uma máquina em momentos específicos. Uma analogia com semáforos de trânsito foi utilizada para facilitar o entendimento da aplicação de diferentes índices. As informações fornecidas pelos índices que caracterizam a quantidade de memória principal utilizada têm utilidade apenas enquanto o sinal verde está aceso. As informações que funcionam como alertas de saturação da memória principal, são úteis para indicar quando o sinal amarelo será aceso. Quando for detectado que o sinal vermelho foi aceso, a carga da máquina deve ser avaliada pelas métricas que avaliam o uso e atividade do espaço de *swap*.

Neste capítulo foram discutidas algumas sugestões de como as políticas poderiam se beneficiar dos índices pesquisados. A maneira exata de usar esses índices e métricas dependerá das necessidades das políticas. Diversas soluções de escalonamento poderão ser desenvolvidas utilizando um ou mais índices. Por exemplo, uma política de balanceamento de carga dotada de um mecanismo de migração de processos pode ser auxiliada por várias das métricas pesquisadas. Baseada nas máquinas para as quais o sinal vermelho estiver aceso, a política poderá identificar as fornecedoras de carga, enquanto as máquinas que estiverem no

sinal verde serão receptoras. Se a política for mais conservadora, poderá escolher fornecedoras de carga logo que detectar máquinas que estejam no sinal amarelo.

A Tabela 5.4 mostra um quadro comparativo que resume as principais características dos índices pesquisados. Nesse quadro são apresentadas as informações: (1) número e nome do índice, (2) o nível de carga que o índice é capaz de representar, de acordo com a analogia com sinais de trânsito, (3) a principal utilidade do índice e (4) a principal desvantagem, se houver.

Tabela 5.4 – Quadro comparativo dos índices pesquisados.

#	Nome	Sinal	Utilidade	Desvantagem
1	Processos bloqueados	Verm	Detecção de thrashing	Difícil determinar limites
2	Memória ocupada	Verd/ Am	Medir utilização em níveis moderados	Não representa quantidade real de memória usada
3	*Cache* e buffers	Verd/ Am/	Complementar o índice 2 na métrica M1	Inclui algumas páginas que pode estar ativas
4	Memória ativa	Verd/ Am/ Verm	Formar a métrica M3 junto com índice 5	Causa picos muito altos em M3
5	Memória inativa	Verd/ Am/ Verm	Complementar o índice 2 na métrica M2. Formar a métrica M3	Causa picos muito altos em M3
6	*Swap* usado	Verm	Medir quantidade de espaço de swap usado	Não mede a quantidade real de swap usado
7	*Swap cache*	Verm	Complementar o índice 6 na métrica M4	Não há
8	*Page stealing*	Verd/ Am	Detectar atividade de alocação intensa	Apresenta picos muito altos. É difícil determinar limites
9	Páginas alocadas	Verd/ Am	Detectar atividade de alocação intensa	Apresenta picos muito altos. É difícil determinar limites
10	Ausências Menores	Verd/ Am	Prever ocorrência de ausências maiores	Apresenta valores elevados em momentos que não antecedem a ausências maiores

11	Ausências Maiores	Vermelho	Detecção de thrashing (métrica M5)	Atinge o valor máximo muito rapidamente

6. Conclusão

Esse capítulo apresenta as conclusões sobre o trabalho realizado, destaca as contribuições desta dissertação e sugere alguns trabalhos futuros.

6.1 Ponderações finais da dissertação

Este trabalho apresentou um estudo de índices de carga de memória, incluindo uma análise de comportamento dos índices identificados. Essa análise resultou na identificação de diversas opções de métricas de carga aplicáveis em diferentes situações.

A necessidade de um estudo como este é motivada por diversas constatações, sendo que as principais são: (1) a crescente demanda das aplicações por maiores quantidades de memória, (2) o baixo desempenho dos sistemas de memória em comparação com os processadores, (3) os efeitos negativos causados pelo mau gerenciamento de memória em sistemas paralelos e (4) a presença de poucos estudos sobre índices de carga de memória na literatura.

A pesquisa dos índices foi guiada por dois requisitos: primeiro, os índices deveriam refletir direta ou indiretamente a utilização de memória e segundo, a obtenção dos índices deveria ser fácil do sistema operacional Linux em nível de usuário. O primeiro requisito foi definido para mostrar claramente o foco da pesquisa e tornar a análise dos índices mais objetiva. É possível que os índices pesquisados sejam usados em conjunto com outros índices que não reflitam a utilização da memória, mas este trabalho não esteve focado em encontrar

essas relações. Contudo, uma informação de carga, a fila de processos prontos, foi utilizada para auxiliar a análise do Índice 1 (Processos bloqueados), resultando na definição da métrica M5. O segundo requisito foi definido porque a obtenção dos índices deveria ser feita sem a necessidade de modificações do núcleo do Linux melhorando, assim, a flexibilidade dos índices e facilitando sua utilização na prática. Esse requisito foi facilitado pela disponibilidade do sistema de arquivos /proc, que fornece, de forma simples, uma grande gama de informações sobre o sistema.

Foram identificados 11 índices de carga que satisfazem os requisitos definidos. Esses índices incluem tanto informações em termos de *quantidade* de espaço usado como informações de *atividade* de memória. Essas duas categorias de informações mostraram ser complementares, já que as vantagens dos índices de uma categoria puderam minimizar as desvantagens dos índices da outra. Por exemplo, o índice que mede a quantidade de memória ocupada é útil apenas enquanto a memória não está sobrecarregada. A partir daí, é interessante utilizar índices que reflitam a utilização do *swap*. O índice que mede a quantidade de swap usado demonstrou algumas desvantagens, que foram superadas pelo índice que mede a atividade de ausências maiores (índice de atividade). Esse índice demonstrou vantagens que sugerem a sua adoção para representar a carga de uma máquina quando a memória principal está sobrecarregada.

Um monitoramento da execução de uma carga de trabalho composta de três aplicações reais foi realizado para coletar os índices de carga. A carga de memória durante o monitoramento foi mantida em diferentes níveis, para capturar o comportamento dos índices em situações de carga moderada a alta. Esse mecanismo se mostrou útil para o entender do comportamento de alguns índices em momentos de transição entre os níveis de utilização. Por exemplo, o índice que mede a quantidade de ausências menores (Índice 10) se mostrou capaz de prever a saturação da memória principal e a ocorrência de ausências maiores. Esse

comportamento foi identificado a partir da observação de que o valor do Índice 10 aumenta significativamente momentos antes da ocorrência de ausências maiores.

A elaboração de métricas a partir da combinação de um ou mais índices resultou em maneiras mais precisas de representar a carga de memória. Para representar a quantidade de memória realmente utilizada, não é suficiente adotar o valor fornecido pelo Linux para esse propósito. Para isso, foram propostas duas métricas (M1 e M2) que consideram como ocupado apenas o espaço de memória que está ativo. As duas métricas apresentam comportamentos similares, não sendo possível afirmar qual é a mais precisa. Contudo, pelo seu princípio, a métrica M2 aparenta ser mais precisa porque considera como livre apenas a quantidade de memória que está realmente inativa, enquanto que M1 considera que todo o espaço ocupado por *caches* e *buffers* está livre. De forma semelhante, a quantidade de swap realmente utilizado não deve ser medida diretamente através da informação fornecida pelo Linux. Através da combinação com o índice que mede o *swap cache*, foi criada a métrica M4, que representa com precisão a quantidade de swap realmente usado. A métrica M3 representa a proporção entre os índices que medem páginas ativas e inativas. Como o Linux tenta manter uma relação de 2 para 1 entre esses índices, um valor consideravelmente maior que 2 dessa relação pode indicar sobrecarga de memória. A última métrica proposta (M5) tem o objetivo de quantificar o nível de *thrashing* calculando uma proporção entre os tamanhos das filas de processos bloqueados e prontos. De acordo com essa métrica, considera-se que o SO está executando menos computações úteis quando o número de processos bloqueados é maior do que o número de processos rodando (prontos).

6.2 Contribuições

As principais contribuições dos estudos apresentados nesta dissertação são:

- Análise da bibliografia da área com ênfase na identificação dos principais problemas referentes ao gerenciamento de memória. A partir dessa revisão

bibliográfica é possível entender quais características devem ser encontradas em novos índices de carga.

- Identificação de 11 índices de carga relacionados ao uso e à atividade de memória que podem ser obtidos facilmente do sistema operacional Linux. Vários desses índices não haviam sido considerados em outros trabalhos. Além disso, o conjunto de índices corresponde a informações específicas para medir a carga de memória, ao contrário de vários trabalhos encontrados na literatura, que têm objetivos mais abrangentes e não se focam em um recurso específico.
- Realização de uma análise de comportamento dos valores apresentados pelos índices durante a execução de aplicações reais. Isso possibilitou entender o comportamento de cada índice em diferentes situações de carga de memória. O uso de aplicações reais torna a análise mais realista e aumenta a confiabilidade dos índices de carga.
- Identificação dos níveis de carga nos quais os índices são úteis. Isso é importante para definir como as políticas de escalonamento, dependendo de seus objetivos, podem utilizar um determinado índice.
- Identificação de métricas de desempenho que combinam mais de um índice de carga. As métricas criadas correspondem a formas mais precisas de representar a carga de memória do que índices isolados. Algumas estatísticas fornecidas pelo sistema operacional poderiam levar a decisões erradas, caso não fosse combinadas com outras estatísticas.
- Implementação de uma ferramenta de monitoramento para coletar índices de carga e fornecê-los a políticas de escalonamento. Essa ferramenta permite que políticas empreguem os índices pesquisados e os incluam em suas decisões de escalonamento sem muito esforço.

6.3 Sugestões para trabalhos futuros

A análise de comportamento dos índices pesquisados sugere a necessidade de alguns estudos mais especializados para adicionar mais significado aos índices. Além disso, alguns passos precisam ser executados para que os índices possam ter sua eficiência comprovada na prática. Nesse sentido, as sugestões para trabalhos futuros baseados nesta dissertação são:

- Realizar estudos específicos para encontrar limites (*thresholds*) para alguns índices. Isso requer uma análise incluindo a execução de algumas aplicações reais, a fim de verificar qual o impacto dos valores dos índices no desempenho das aplicações;
- Aplicar os índices pesquisados em políticas de escalonamento existentes e executar testes reais ou simulados para verificar se os índices melhoraram as decisões de escalonamento dessas políticas;
- Criar políticas de escalonamento que sejam focadas em *aplicações memory-intensive* e aplicar os índices pesquisados;
- Combinar os índices de memória com outros tipos de índices para verificar seu efeito no escalonamento de outros tipos de aplicações. A maioria das aplicações depende de quantidades consideráveis de memória para executar, mesmo não sendo consideradas *memory-intensive*. Portanto, índices de carga de memória, potencialmente, podem ajudar no escalonamento dessas aplicações;
- Estender o módulo de monitoramento para permitir a comunicação com várias políticas de escalonamento. Isso envolve definir um protocolo de comunicação para definir as mensagens que o módulo poderá trocar com as políticas. O módulo funcionaria como um agente de monitoramento que receberia pedidos (mensagens) das políticas requisitando os índices.

7. Referências bibliográficas

Acharia, A.; Setia, S. (1999). Availability and Utility of Idle Memory in Workstation Clusters. *In: Proceedings of ACM SIGMETRICS*, Atlanta, USA, p.35-46.

Alderson, A.; Lynch, W.; Randell, B. (1972) Thrashing in a Multiprogrammed Paging System. In: Hoare, C. A. R.; Perrot, R. H. (editores), *Operating Systems Techniques*, London: Academic Press, p. 152-167.

Ali S.; Braun, T. D.; Siegel, H. J.; Maciejewski, A. A.; Beck, N.; Bölöni, L.; Maheswaran, M.; Reuther, A. I.; Robertson, J. P.; Theys, M.D.; Yao. B. (2005) Characterizing Resource Allocation Heuristics for Heterogeneous Computing Systems. In: Zelkowitz, M; Hurson A. (Editores). *Advances in Computers: Volume 63: Parallel, Distributed, and Pervasive Computing*, Elsevier, p. 93-129.

Amir, Y.; Awerbuch, B.; Barak, A.; Borgstrom, R. S.; Keren, A. (2000). On Opportunity Cost Approach for Job Assignment in a Scalable Computing Cluster. *IEEE Transactions on Parallel and Distributed Systems*, v. 11, n. 7, p. 760-768.

Amiri, K.; Petrou, D.; Ganger, G. R.; Gibson, G. A. (2000). Dynamic Function Placement for Data-intensive Cluster Computing. *In: Proc. USENIX 2000 Annual Technical Conference*, San Diego, CA.

Anderson, T. E.; Culler, D. E.; Patterson, D. A. (1995). A Case for NOW (Networks of Workstations). *IEEE Micro*, v. 15, n. 1, pp. 54-64.

Barak, A; Braverman, A. (1998). Memory Ushering in a Scalable Computing Cluster. *Journal of Microprocessors and Microsystems*, v. 22, n. 3-4, p. 175-182.

Becker, D. J.; Sterling, T.; Savarese, D.; Dorband, J. E.; Ranawak, U. A.; Packer, C. V. (1995) Beowulf: A Parallel Workstation for Parallel Computing. *In: Proc. the 24th International Conference on Parallel Processing*. Urbana-Champain, Illinois.

BRAMS - Brazilian Regional Atmospheric Modeling System. (2005) Disponível em <http://www.cptec.inpe.br/brams>. Acessado: dez. 2005.

Branco, K. R. L. J. C. (2004) Índices de carga e desempenho em ambientes paralelos/ distribuídos: modelagem e métricas. Tese (Doutorado). ICMC-USP, São Carlos, Dez. 2004.

Braun, T. D.; Siegel, H. J.; Beck N.; Blni, L. L.; Maheswaran, M.; Reuther, A. I.; Robertson, J. P.; Theys, M. D.; Yao, B.; Hensgen, D.; Freund, R. F. (2001). A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems, *Journal of Parallel and Distributed Computing,* v. 61, n. 6, p. 810-837.

Buyya, R. (1999) High Performance Cluster Computing: Architectures and Systems, v. 1, New Jersey: Prentice Hall, 1999.

Casavant; T. L.; Kuhl, J. G. (1988). A Taxonomy of Scheduling in General-Purpose Distributed Computing Systems. *IEEE Transactions on Software Engineering*, v. 14, n. 2, p. 141-154.

Cockcroft A. (2005). Help! I've lost my memory. Disponível em: <http://www.itworld.com/Comp/2377/UIR951001perf>. Acesso: Dez. 2005.

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

Denning, P. J. (1968b). Thrashing: its causes and prevention. In: *Proceedings of AFIPS Conference 1968*, p. 915-922.

Feitelson, D. G.; Rudolph, L.; Schwiegelshohn, U.; Sevcik, K. C.; Wong, P. (1997). Theory and Practice in Parallel Job Scheduling. *In: IPPS' 97 Workshop on Job Scheduling Strategies for Parallel Processing, LNCS 1291*, Geneva, Switzerland.

Ferrari, D.; Zhou, S. (1987). An Empirical Investigation of Load Indices for Load Balancing Applications. *In: Proceedings 12th Int'l Symposium on Computer Performance Modeling, Measurement and Evaluation (Performance'87)*, p. 515-528.

Ferstl, F. (1996). Job-and resource-management systems in heterogeneous clusters. *Future Generation Computer Systems - FGCS*, v. 12, p. 39-51.

Fujimoto, R. M. (1990). Performance of Time Warp under Synthetic Workloads. *In: Proceedings of the SCS Multiconference on Distributed Simulation*, p.23-28.

Gorman, M. (2004). Understanding the Linux Virtual Memory Manager. Upper Saddle River: Prentice-Hall, 2004. 832 p.

Graham, P. A. (2001). DSM Cluster Architecture Supporting Aggressive Computation in Active Networks. *In: Proc. the 3rd International Workshop on Distributed Shared Memory (DSM 2001)*. Brisbane, Australia.

Gropp W.; Lusk E.; Skjellum A. (2000). Using MPI: portable parallel programming with the message-passing interface. 2ª edição. Cambridge: MIT Press, 2000. 350 p.

Harchol-Balter, M.; Downey, A. B. (1997). Exploiting Process Lifetimes Distributions for Dynamic Load Balancing. *ACM Transactions on Computer Systems*, v. 15, n. 3, p. 253-285.

Hennesy, J. L.; Patterson, D. A. Computer Architecture: A Quantitative Approach, 2 ed., San Francisco: Morgan Kaufmann, 1996.

Ishii, R. P.; Mello, R. F.; Senger, L. J.; Santana, M. J.; Santana, R. H. C. (2005). Improving Scheduling of Communication Intensive Parallel Applications on Heterogeneous Computing Environments. *Parallel Processing Letters*, v. 15, n. 4, p. 423-438.

Jiang, S.; Zhang, X. (2001). Adaptive Page Replacement to Protect Thrashing in Linux. *In: Proceedings of the 5th Annual Linux Showcase & Conference*. Oakland, California.

Jiang, S.; Zhang, X. (2005). Token-ordered LRU: an effective page replacement policy and its implementation in Linux systems, *Performance Evaluation*, v. 60, n 1-4, 2005, p. 5-29.

Kunz, T. (1991). The Influence of Different Workload Descriptions on a Heuristic Load Balancing Scheme. *IEEE Transactions on Software Engineering*, v. 17, n. 7, p.725-730.

Lin, W.; Reinhardt, S. K.; Burger, D. (2001). Reducing DRAM Latencies with an Integrated Memory Hierarchy Design. *In: Proceedings Seventh Int'l Symp. High-Performance Computer Architecture (HPCA-7)*. Monterrey, Mexico.

Linux Kernel. (2005). Código-fonte do kernel do Linux versão 2.6.11. Disponível em: <http://www.kernel.org>. Acesso: Nov. 2005.

Love, R. (2004). Linux Kernel Development: A practical guide to the design and implementation of the Linux kernel, Indianapolis: Sams Publishing, 2004.

Martin D. E.; Wilsey, P. A.; Hoekstra, R. J; Keiter, E. R.; Hutchinson, S. A.; Russo, T. V.; Waters, L. J. (2003). Redesigning the WARPED Simulation Kernel for Analysis and Application Development. *In: Annual Simulation Symposium 2003*. Orlando, Florida, p. 216-223.

Matos, J.; Bortolato, E.; Camilo Jr, A.; Martini, J. A.; Gonçalves R. A. L.; Souza, P. S. L. (2006). Binary SCF: GAMESS improvements for energy evaluation based on SCF methods. *Computer Physics Communications*, v. 174, n. 1, p. 1-16.

Mehra, P.; Wah, B. W. (1997) Automated learning of load-balancing strategies in multiprogrammed distributed systems. *International Journal of Systems Science*, v. 28, n. 11, p. 1077-1099.

Murray, N.; Horman N. (2004). Understanding Virtual Memory. *Red Hat Magazine*. Issue 1, Nov. 2004. Disponível em: <http://www.redhat.com/magazine/001nov04/features/vm/>. Acesso: Nov. 2005.

Opensolaris.org (2005). Procfs. Disponível em: http://www.opensolaris.org/os/community/observability/process/procfs. Acesso: Dez. 2005.

Pielke, R. A.; Cotton W. R.; Walko, R. L.; Tremback, C. J.; Lyons, W. A.; Grasso, L. D.; Nicholls, M. E.; Moran, M. D.; Wesley, D. A.; Lee, T. J.; Copeland, J. H. (1992). A comprehensive meteorological modeling system-RAMS. *Meteorology of Atmospheric Physics*, v. 49, p. 69-91.

Red Hat, Inc. (2003a). Tips & Tricks Featured Article: /proc/meminfo Explained. Disponível em: <http://www.redhat.com/advice/tips/meminfo.html>. Acesso: Jan. 2005.

Red Hat, Inc. (2003b). Red Hat Linux 9 Manual. Disponível em: <http://www.redhat.com/docs/manuals/linux/RHL-9-Manual/>. Acesso: Jan. 2006.

Riel, R. (2001). Page replacement in Linux 2.4 memory management. *In: Proceedings of the 2001 USENIX Annual Technical Conference*. Boston, MA.

Schmidt, M. W.; Baldridge, K.K.; Boatz, J.A.; Elbert, S.T.; Gordon, M.S.; Jensen, J.H.; Koseki, S.; Matsunaga, N.; Nguyen, K.A.; Su, S.J.; Windus, T.L.; Dupuis, M.; Montgomery, J.A. (1993). The general atomic and molecular electronics structure systems. *Journal of Computational Chemistry*, v. 14, n. 11, p. 1347-1363.

Schrock, E. (2004). A brief history of /proc. Disponível em: <http://blogs.sun.com/roller/page/eschrock/20040625>. Acesso: Dez. 2004.

Senger, L. J. (2004). Escalonamento de processos: uma abordagem dinâmica e incremental para a exploração de características de aplicações paralelas. Tese (Doutorado), ICMC-USP, São Carlos. Dez. 2004.

Shirazi, B.; Hurson, A. R. (1992). Special Issue on Scheduling and Load Balancing: guest editor's introduction. *Journal of Parallel and Distributed Computing*, v. 16, n. 4, p. 271-275.

Shivaratri, N.G.; Krueger, P.; Singhal, M. (1992). Load Distributing for Locally Distributed Systems. *IEEE Computer*, v. 25, n. 12.

Smith, W.; Taylor, V.; Foster, I. (1999). Using run-time predictions to estimate queue wait times and improve scheduler performance. In: *Job Scheduling Strategies for Parallel Processing,* LNCS 1659. p. 202-219.

Souza, P. S. L. (2000) AMIGO: Uma Contribuição para a Convergência na Área de Escalonamento de Processos. Tese (Doutorado). IFSC-USP, São Carlos. Jun. 2000.

Stallings, W. (1999). Computer Organization and Architecture, 5. ed, Upper Saddle River: Prentice-Hall, 1999. 768 p.

Tanenbaum, A. S. (2003). Modern Operating Systems. Upper Saddle River: Prentice-Hall, 2003. 976 p.

Xiao, L; Chen, S; Zhang, X. (2002) Dynamic cluster resource allocations for jobs with known and unknown memory demands. *IEEE Transactions on Parallel and Distributed Computing*, v. 13, n. 3, p. 223-240.

Xu, C.; Lau, F. C. M. (1997). Load Balancing in Parallel Computers: Theory and Practice, Boston: Kluwer Academic Publishers, 1997.

Zhang, Z.; Zhu, Z.; Zhang, X. (2001). Cached DRAM for ILP Processor Memory Access Latency Reduction. *IEEE Micro*, v. 21, n. 4, p. 22-32.

Zhou, S.; Wang, J.; Zheng, X.; Delisle P. (1993) Utopia: A load-sharing facility for large heterogeneous distributed computing systems. *Software - Practice and Experience,* v. 23, n. 12, 1993.