Universidade Tecnológica Federal do Paraná
Programa de Pós-Graduação em Computação Aplicada

FELLIPE MEDEIROS VEIGA

ESTUDO DA EFETIVIDADE DOS MECANISMOS DE COMPARTILHAMENTO DE MEMÓRIA EM HIPERVISORES

DISSERTAÇÃO

Curitiba PR
Agosto de 2015

FELLIPE MEDEIROS VEIGA

ESTUDO DA EFETIVIDADE DOS MECANISMOS DE COMPARTILHAMENTO DE MEMÓRIA EM HIPERVISORES

Dissertação submetida ao Programa de Pós-Graduação em Computação Aplicada da Universidade Tecnológica Federal do Paraná como requisito parcial para a obtenção do título de Mestre em Computação Aplicada.

Área de concentração: *Engenharia de Sistemas Computacionais*

Orientador: Carlos Alberto Maziero

Curitiba PR
Agosto de 2015

Dados Internacionais de Catalogação na Publicação

V426e
2015

Veiga, Fellipe Medeiros
Estudo da efetividade dos mecanismos de compartilhamento de memória em hipervisores / Fellipe Medeiros Veiga.-- 2015.
71 p.: il.; 30 cm

Texto em português, com resumo em inglês
Dissertação (Mestrado) - Universidade Tecnológica Federal do Paraná. Programa de Pós-graduação em Computação Aplicada, Curitiba, 2015
Bibliografia: p. 65-68

1. Gerenciamento de memória (Computação). 2. Sistemas de memória de computadores. 3. Computação em nuvem. 4. Sistemas de computação virtual. 5. Sistemas operacionais (Computadores). 6. VMware. 7. Computação - Dissertações. I .Maziero, Carlos Alberto, orient. II. Universidade Tecnológica Federal do Paraná - Programa de Pós-graduação em Computação Aplicada. III. Título.

CDD: Ed. 22 -- 621.39

Biblioteca Central da UTFPR, Câmpus Curitiba

ATA DE DEFESA DE DISSERTAÇÃO DE MESTRADO Nº 37

Aos 28 dias do mês de agosto de 2015 realizou-se na sala B-204 a sessão pública de Defesa da Dissertação de Mestrado intitulada "Estudo da Efetividade dos Mecanismos de Compartilhamento de Memória em Hipervisores", apresentada pelo aluno **Fellipe Medeiros Veiga** como requisito parcial para a obtenção do título de Mestre em Computação Aplicada, na área de concentração "Engenharia de Sistemas Computacionais", linha de pesquisa "Redes e Sistemas Distribuídos".

Constituição da Banca Examinadora:

Prof. Dr. Carlos Alberto Maziero, UTFPR - CT (Presidente) _____

Prof. Dr. Luis Carlos Erpen de Bona, UFPR _____

Prof. Dr. Luiz Nacamura Júnior, UTFPR - CT_____

Em conformidade com os regulamentos do Programa de Pós-Graduação em Computação aplicada e da Universidade Tecnológica Federal do Paraná, o trabalho apresentado foi considerado _____ (aprovado/reprovado) pela banca examinadora. No caso de aprovação, a mesma está condicionada ao cumprimento integral das exigências da banca examinadora, registradas no verso desta ata, da entrega da versão final da dissertação em conformidade com as normas da UTFPR e da entrega da documentação necessária à elaboração do diploma, em até _____ dias desta data.

Ciente (assinatura do aluno): _____

_____

(para uso da coordenação)

A Coordenação do PPGCA/UTFPR declara que foram cumpridos todos os requisitos exigidos pelo programa para a obtenção do título de Mestre.

Curitiba PR, _____/_____/_____ _____

"A Ata de Defesa original está arquivada na Secretaria do PPGCA".

Agradecimentos

Agradeço inicialmente a Deus por me permitir chegar até esse momento. A minha família e namorada pelo incentivo, apoio e paciência em todo o período do mestrado, principalmente nos momentos mais difíceis.

Agradeço ao professor Carlos Alberto Maziero, meu orientador, pelo apoio, confiança e paciência durante todo o período do curso de mestrado. Poder contar com sua orientação foi uma honra e um aprendizado muito grande.

Aos meus colegas da Companhia de Tecnologia da Informação e Comunicação do Paraná – CELEPAR pelas sugestões e pela compreensão durante minhas ausências principalmente no período das disciplinas.

Aos demais professores do PPGCA pelas revisões e contribuições nos Seminários de Acompanhamento e também por todo o conhecimento adquirido nas suas disciplinas.

Aos colegas de mestrado e colaboradores da UTFPR que contribuíram direta ou indiretamente para que este trabalho fosse realizado e concluído.

Resumo

VEIGA, Fellipe Medeiros. Estudo da efetividade dos mecanismos de compartilhamento de memória em hipervisores. 71 f. Dissertação (Mestrado) - Programa de Pós-Graduação em Computação Aplicada, Universidade Tecnológica Federal do Paraná. Curitiba, 2015.

A crescente demanda por ambientes de virtualização de larga escala, como os usados em *datacenters* e nuvens computacionais, faz com que seja necessário um gerenciamento eficiente dos recursos computacionais utilizados. Um dos recursos mais exigidos nesses ambientes é a memória RAM, que costuma ser o principal fator limitante em relação ao número de máquinas virtuais que podem executar sobre o mesmo *host* físico. Recentemente, hipervisores trouxeram mecanismos de compartilhamento transparente de memória RAM entre máquinas virtuais, visando diminuir a demanda total de memória no sistema. Esses mecanismos "fundem" páginas idênticas encontradas nas várias máquinas virtuais em um mesmo quadro de memória física, usando uma abordagem *copy-on-write*, de forma transparente para os sistemas convidados. O objetivo deste estudo é apresentar uma visão geral desses mecanismos e também avaliar seu desempenho e efetividade. São apresentados resultados de experimentos realizados com dois hipervisores populares (VMware e KVM), usando sistemas operacionais convidados distintos (Linux e Windows) e cargas de trabalho diversas (sintéticas e reais). Os resultados obtidos evidenciam diferenças significativas de desempenho entre os hipervisores em função dos sistemas convidados, das cargas de trabalho e do tempo.

Palavras-chave: Gerência de memória. Compartilhamento de memória. Virtualização. Nuvens Computacionais.

Abstract

VEIGA, Fellipe Medeiros. Study of the effectiveness of memory sharing mechanisms in hypervisors. 71 f. Dissertação (Mestrado) - Programa de Pós-Graduação em Computação Aplicada, Universidade Tecnológica Federal do Paraná. Curitiba, 2015.

The growing demand for large-scale virtualization environments, such as the ones used in cloud computing, has led to a need for efficient management of computing resources. RAM memory is the one of the most required resources in these environments, and is usually the main factor limiting the number of virtual machines that can run on the physical host. Recently, hypervisors have brought mechanisms for transparent memory sharing between virtual machines in order to reduce the total demand for system memory. These mechanisms "merge" similar pages detected in multiple virtual machines into the same physical memory, using a copy-on-write mechanism in a manner that is transparent to the guest systems. The objective of this study is to present an overview of these mechanisms and also evaluate their performance and effectiveness. The results of two popular hypervisors (VMware and KVM) using different guest operating systems (Linux and Windows) and different workloads (synthetic and real) are presented herein. The results show significant performance differences between hypervisors according to the guest system workloads and execution time.

Keywords: Memory management. Memory sharing. Virtualization. Cloud computing.

Lista de Figuras

Lista de Tabelas

Lista de Símbolos

(α) Fator de compartilhamento

Lista de Abreviações

ASLR	Address space layout randomization
CBPS	Content-base Page Sharing
GB	Gigabyte
GUI	Graphical User Interface
KB	Kilobyte
KSM	Kernel Samepage Merging
KVM	Kernel-based Virtual Machine
MB	Megabyte
RAM	Random Access Memory
VM	Virtual Machine
VMM	Virtual Machine Monitor
TPS	Transparent Page Sharing
UTFPR	Universidade Tecnológica Federal do Paraná

Sumário

Capítulo 1

Introdução

Virtualização é um assunto que atualmente recebe grande atenção; seu uso consiste em executar vários sistemas operacionais dentro do mesmo equipamento físico. Cada máquina virtual ou sistema convidado (*guest system*) funciona como um sistema independente do sistema nativo ou sistema hospedeiro (*host system*). O hipervisor ou VMM *(Virtual Machine Monitor)* constitui a camada responsável pela virtualização. Ficando situada entre o sistema hospedeiro e os sistemas convidados, o hipervisor separa a execução dos processos executados nos ambientes virtualizados, sendo que esses não interferem no sistema hospedeiro.

A arquitetura de máquinas virtuais foi proposta e utilizada inicialmente na década de 1960 [Chen and Noble, 2001] e constantemente vem passando por aprimoramentos. Seu propósito inicial era prover um sistema monousuário exclusivo para cada usuário.

Uma abordagem que permite uma economia potencial de recursos é o compartilhamento transparente dos mesmos entre as várias máquinas virtuais. Por "compartilhamento transparente", entende-se aquele compartilhamento que é realizado sem alterar o funcionamento dos participantes, ou seja, sem que estes percebam.

Ao reduzir a demanda de memória física por cada máquina virtual, mais sistemas podem potencialmente ser alocados em um dado servidor, o que resulta em uma melhor utilização do *hardware* disponível [Barker et al., 2012].

1.1 Motivação

O compartilhamento de recursos entre os sistemas convidados tem sua importância, pois nestes ambientes, uma grande quantidade de arquivos binários e bibliotecas de sistema são comumente usados por várias máquinas virtuais. Além disso, cada sistema convidado mantém páginas de cache de memória no seu espaço de memória, isolado das demais [Kim et al., 2009]. Para evitar a redundância de componentes sendo carregados em memória resultando em desperdício de recursos, são propostas técnicas de compartilhamento de memória.

A memória RAM, costuma ser o principal fator limitante em relação ao número de máquinas virtuais que podem executar sobre um mesmo *host* físico, sendo um dos recursos mais exigidos em ambientes virtualizados. Dentro de cada máquina virtual, o sistema operacional convidado gerencia sua própria memória RAM, usando diversas técnicas. Entretanto, esse gerenciamento se limita ao escopo da própria máquina virtual, que é isolada das demais pelo hipervisor. Alguns hipervisores implementaram mecanismos de compartilhamento transparente de memória RAM entre suas máquinas virtuais, visando diminuir a demanda total de memória

RAM no sistema físico [Chang et al., 2011]. Essa forma de compartilhamento é distinta daquela usada para a comunicação entre processos ou máquinas virtuais.

A presente pesquisa busca dimensionar a efetividade dos mecanismos de compartilhamento de memória nos hipervisores mais usados no mercado, bem como a avaliação desses mecanismos com vários sistemas convidados disponíveis. Trata-se de um estudo experimental, com testes e medições para avaliar além do potencial de compartilhamento a efetividade do compartilhamento entre máquinas virtuais, pretende-se com esse trabalho evidenciar as diferenças significativas de desempenho entre os hipervisores, em função dos sistemas convidados, das cargas de trabalho e do tempo de execução.

É importante avaliar a eficiência desses mecanismos em ambientes computacionais distintos, de forma a verificar como os hipervisores atuais tratam essa questão de acordo com os sistemas convidados utilizados. A forma de avaliação será por meio de experimentos e coleta das informações resultantes para que posteriormente essas possam gerar indicadores de compartilhamento. Outro fator que merece atenção na pesquisa é avaliar como os níveis de compartilhamento evoluem ao longo do tempo, com a mudança nos estados internos das máquinas virtuais.

1.2 Objetivos

O objetivo geral da pesquisa é avaliar a efetividade dos mecanismos de compartilhamento de memória em ambiente de virtualização, levando em consideração os diferentes hipervisores disponíveis no mercado.

Dentre os principais objetivos específicos destacam-se:

- Compreender de forma profunda o funcionamento os mecanismos de compartilhamento de memória em ambientes virtualizados;
- Avaliar como o compartilhamento se comporta ao longo do tempo, ou seja, conforme a evolução das máquinas virtuais;
- Avaliar como os diferentes sistemas operacionais se comportam em relação ao compartilhamento de memória entre máquinas virtuais;
- Avaliar o desempenho desses mecanismos de compartilhamento, medindo as taxas de compartilhamento obtidas e o custo computacional exigido;
- Avaliar os fatores (família e versão do sistema operacional convidado e aplicações) que influenciam de forma significativa no compartilhamento de memória em ambientes virtualizados;
- Comparar o potencial teórico de compartilhamento com o compartilhamento real obtido por cada um dos hipervisores estudados;

1.3 Estrutura do Documento

Os capítulos subsequentes deste documento estão estruturados da seguinte forma: O capítulo 2 descreve os principais conceitos sobre virtualização e a fundamentação teórica necessária para realização do trabalho; o capítulo 3 apresenta os mecanismos de compartilhamento e

os trabalhos relacionados; o capítulo 4 descreve a metodologia de experimentação empregada neste estudo, o capítulo 5 apresenta e analisa os resultados do estudo do potenciais teórico e prático, além do compartilhamento obtido em cada hipervisor e por fim, o capítulo 6, apresenta as conclusões e considerações finais do estudo e trabalhos futuros.

Capítulo 2

Fundamentação Teórica

Neste capítulo serão apresentados os tópicos relacionados a virtualização. Inicialmente, serão apresentados as definições de virtualização com um breve histórico. Depois são descritos os tipos, técnicas e aplicações da virtualização, na sequência serão descritas algumas técnicas de gerenciamento de memória em ambiente de virtualização e finalmente as considerações finais desse capítulo.

2.1 Introdução

A ideia fundamental por trás da virtualização é a introdução de uma camada adicional, onde os recursos de nível inferior podem ser mapeados de modo transparente para vários sistemas operacionais de alto nível ao mesmo tempo. [Chiueh and Brook, 2005] define que cada máquina virtual é uma instância da máquina física, proporcionando aos usuários uma ilusão de acessar a máquina física diretamente. A tecnologia de virtualização permite combinar ou dividir os recursos de *hardware* e *software* em um ou mais ambientes operacionais convidados.

Ambientes de computação de larga escala, como as nuvens computacionais, têm na virtualização uma tecnologia fundamental. Apesar do custo do *hardware* ser cada vez menor, o uso cada vez maior da virtualização e da computação em nuvem em substituição a outros sistemas, permite uma economia maior de recursos sendo um fator importantíssimo nos ambientes computacionais. A possibilidade de configurar, instanciar e encerrar máquinas virtuais sob demanda propicia uma grande flexibilidade no gerenciamento desses ambientes, visando o uso eficiente dos recursos computacionais disponíveis. Hipervisores modernos permitem inclusive a migração de máquinas virtuais entre *hosts* físicos sem perda de estado, o que abre a possibilidade de balanceamento dinâmico de carga no sistema.

A pesquisa aqui apresentada está inserida na tecnologia de virtualização e no gerenciamento de recursos em máquinas virtuais, portanto nos tópicos seguintes são descritas as definições da tecnologia de virtualização e técnicas de gerenciamento de memória em máquinas virtuais.

2.2 Virtualização

Na década de 60, a IBM desenvolveu o sistema operacional M44/44X, esse sistema experimental simulava múltiplos computadores IBM 7044, por meio do uso de particionamento

lógico, que permitia o *mainframe* executar vários processos ao mesmo tempo como se fossem vários mainframes diferentes, posteriormente o estudo do M44/44X foi aproveitado como base para a construção do sistema operacional OS/360 [Creasy, 1981]. A tendência dominante nos sistemas naquela época era fornecer a cada usuário um ambiente monousuário completo, com seu próprio sistema operacional e aplicações, completamente independente e desvinculado dos ambientes dos demais usuários [Laureano et al., 2007].

Apesar de já implementado o conceito sobre a virtualização não estava totalmente formalizado. Na década de 70, os pesquisadores Gerald J. Popek e Robert P. Goldberg formalizaram vários conceitos relacionados à virtualização e as condições necessárias para que um conjunto de *hardware* suporte virtualização [Popek and Goldberg, 1974]. Nesse estudo, a máquina virtual é definida como um sistema eficiente, isolado e duplicado da máquina real. Um VMM deve ter como características:

- Fornecer uma abstração para os programas idêntica à máquina original;
- Os programas executados no ambiente virtualizado devem ser em último caso levemente mais lentos em comparação ao sistema original;
- O VMM terá controle total sobre os recursos do sistema;

As definições de Popek e Goldberg caracterizam a forma de desenvolvimento de um hipervisor que é a camada de *software* responsável pela virtualização. Porém a plataforma de *hardware* existente pode não ser compatível com as premissas definidas nesse estudo.

Na década de 80, a técnica de virtualização ficou um bom tempo em segundo plano, pois o objetivo inicial que era fornecer ambientes distintos dentro de um *hardware* de grande porte, foi sendo perdida com o passar do tempo principalmente pela evolução dos microcomputadores e facilidade de aquisição do *hardware* [Barham et al., 2003]. Posteriormente os computadores modernos passaram a contar com recursos suficientes para usar a virtualização apresentando a ilusão de várias máquinas virtuais menores. Cada qual executando um sistema operacional em separado. Essa evolução contribuiu para que a virtualização retomasse sua importância. A evolução do *hardware* possibilitou a utilização de sistemas virtualizados com baixo custo de desempenho para o sistema hospedeiro.

Visando a redução da complexidade dos componentes dos sistemas operacionais e para fornecer uma abstração de outras arquiteturas de *hardware*, foi proposto na década de 90 um sistema operacional denominado SimOS, com o objetivo de proporcionar um ambiente de simulação de programas significativamente mais rápido do que os simuladores de máquina existentes até o momento. A abordagem utilizada no SimOS seria simular o *hardware* de máquinas de diferentes arquiteturas (MIPS, Alpha e SPARC) usando o *hardware* e serviços de uma estação de trabalho Unix Genérica. Com isso SimOS pode criar um ambiente de simulação amplamente confiável e simples [Rosenblum and Varadarajan, 1994].

O sistema SimOS serviu de base para simulações mais complexas que as aplicações inicialmente projetadas e posteriormente foi possível a simulação de um sistema operacional inteiro. A experiência com o SimOS permitiu o desenvolvimento de outras soluções de virtualização para outras plataformas [Rosenblum et al., 1995].

Até aquele momento (final da década de 90), a virtualização não estava difundida na arquitetura x86, sendo restrita às arquiteturas de um único fornecedor com as instruções específicas para virtualização. O fator que dificultou o uso da virtualização na arquitetura x86

foi que a mesma não possuía suporte as instruções específicas de virtualização e não existia o interesse da indústria oferecer a virtualização para a arquitetura x86. A alternativa foi construir uma solução de forma independente e de compatibilidade com a grande quantidade de *hardware* disponível, o VMware Workstation foi a primeira utilização de virtualização para plataforma x86 [Bugnion et al., 2012].

Desde então a virtualização se tornou um termo em evidência nas comunidades de pesquisa. Tendo suas classificações e suas aplicações explicadas a seguir.

2.2.1 Classificação de Máquinas Virtuais

Máquinas virtuais podem ser classificadas de diversas formas. Segundo [King et al., 2003] pode-se dividir as máquinas virtuais em: máquinas virtuais de aplicação, que têm por objetivo proporcionar um ambiente de virtualização para uma aplicação, ou até mesmo um processo[1]; máquinas virtuais de sistema que proporcionam a estrutura para virtualização de um sistema operacional completo. No contexto deste trabalho serão descritas informações referentes as máquinas virtuais de sistema.

2.2.2 Tipos de Virtualização

É possível efetuar a classificação dos tipos de virtualização de acordo com a arquitetura que a mesma foi construída, assim têm-se os hipervisores nativos ou tipo I e hipervisores convidados ou tipo II e a abordagem híbrida de máquinas virtuais que serão definidos a seguir:

- **Hipervisores Nativos:** Os hipervisores nativos ou máquinas virtuais de tipo I caracterizam-se por ter o hipervisor implementado diretamente no *hardware* físico, separando o *hardware* das máquinas virtuais, onde cada sistema convidado poderia se comportar como uma máquina física. Como exemplos de hipervisores nativos é possível destacar o Xen, KVM e VMWare ESX Server.

  A Figura 2.1, representa um hipervisor nativo. Nota-se que o fato do mesmo executar diretamente sobre o *hardware* caracteriza essa abordagem como sendo a que obtém melhor performance.

- **Hipervisores Convidados:** Nos hipervisores convidados ou máquinas virtuais de tipo II, o VMM é executado na camada superior do sistema operacional do hospedeiro. Os sistemas convidados funcionam dentro de processos do sistema operacional hospedeiro. Essa abordagem caracteriza-se principalmente pela presença de uma camada adicional entre o VMM e os sistemas convidados. Exemplos de hipervisores convidados são o VMware Workstation e o VirtualBox.

  A Figura 2.2 a representação de um hipervisor convidado.

- **Abordagem Híbrida:** A abordagem híbrida combina as funções dos hipervisores nativos e convidados, segundo [King et al., 2003] caracteriza-se por operar principalmente direto no *hardware* físico (nativo) e utilizar o sistema operacional hospedeiro para realizar operações sensíveis como eventos de entrada e saída para acessar alguns dispositivos.

[1] Um exemplo de máquina virtual de aplicação é a máquina virtual Java.

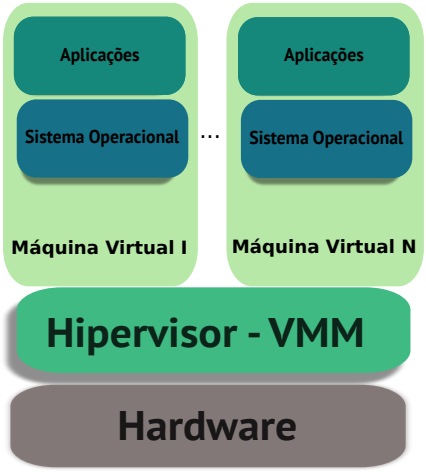

Figura 2.1: Arquitetura de Hipervisor Nativo

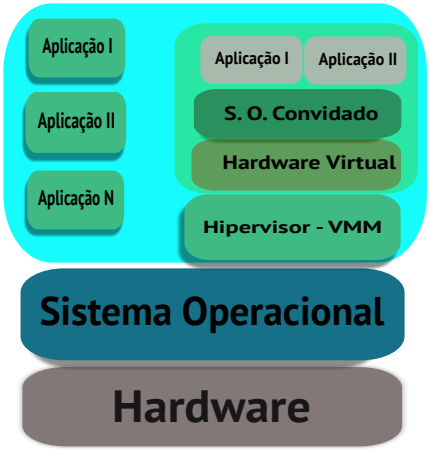

Figura 2.2: Arquitetura de Hipervisor Convidado

2.2.3 Técnicas de Virtualização

As técnicas de virtualização visam eliminar problemas de compatibilidade entre os componentes de arquitetura dos sistemas operacionais convidados e nativos. A seguir são tratadas algumas técnicas de virtualização.

- **Virtualização Total:**

  A virtualização total fornece ao sistema convidado a "imitação" do hardware do sistema hospedeiro. O sistema operacional convidado é executado sem modificações sobre o hipervisor, com isso o desempenho nesses sistemas é significativamente pior. O hipervisor precisa testar e utilizar alternativas para que instruções privilegiadas possam ser executadas em arquiteturas que não suportem virtualização e posteriormente executar a operação de entrada e saída desejada.

  A arquitetura x86 passou a contar com o suporte nativo à virtualização posteriormente, quando as empresas Intel e a AMD desenvolveram tecnologias em seus processadores para suportar a virtualização (Intel VT e AMD-V) [Bugnion et al., 2012]. O principal

objetivo foi a diminuição do *overhead* da virtualização, quando o *hardware* não possui essas tecnologias, utilizam-se a técnica de virtualização total ou tradução dinâmica.

- **Para-virtualização:**

  A técnica de para-virtualização consiste em modificar a estrutura tradicional de virtualização total para uma nova forma, com o objetivo da obtenção de maior desempenho. O sistema operacional convidado é modificado sempre que acessar uma instrução sensível, essas alterações permitem que o sistema convidado consiga acessar por meio de *drivers* diretamente o *hardware* [Barham et al., 2003]. Devido as alterações realizadas no sistema convidado o uso da para-virtualização não é recomendado no uso de virtualização de sistemas operacionais legados [Chiueh and Brook, 2005].

- **Virtualização Aninhada:**

  A virtualização aninhada ou *nested virtualization* propõe o uso de uma máquina virtual dentro de outra máquina virtual, ou seja, um hipervisor irá executar sobre outro hipervisor. Segundo [Ben-Yehuda et al., 2010] essa abordagem permite testes em ambiente de virtualização e proporciona que os clientes possam gerenciar sua infra-estrutura de máquinas virtuais no ambiente de computação em nuvem. A grande desvantagem é que nesse modelo o *overhead* das camadas de virtualização é maior.

  A Figura 2.3 refere-se a arquitetura de virtualização aninhada.

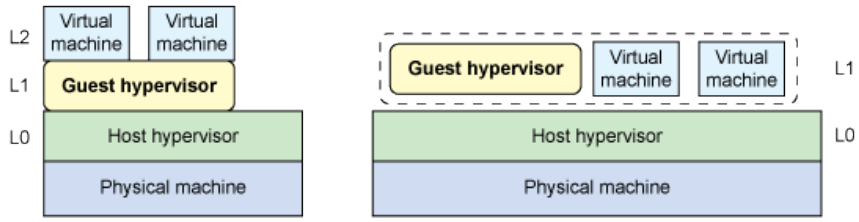

Figura 2.3: Técnica de Virtualização Aninhada [IBM Inc, 2012].

- **Container de Virtualização:**

  Como alternativa as soluções de virtualização tradicionais, o conceito de *container* de virtualização surge principalmente com o objetivo de redução do *overhead* nas operações de entrada e saída em ambientes virtuais. Os *containers* oferecem uma camada leve de virtualização, permitindo desempenho mais próximo ao do sistema nativo [Xavier et al., 2013].

  Cada sistema virtualizado no *container* recebe uma abstração do sistema operacional provendo um ambiente protegido para execução de aplicações. Cada ambiente possui sua identificação e ficam isolados. Como resultado, cada sistema convidado comporta-se como se estivesse executando em seu próprio sistema operacional. Essa técnica virtualiza servidores no topo do próprio sistema operacional, sendo executado sempre no mesmo *kernel* do sistema nativo, ou seja, as máquinas virtuais executam uma ou mais aplicações virtualizadas escritas para o sistema operacional nativo.

  Na abordagem [Xavier et al., 2013] o *container* de virtualização requer modificações no núcleo do sistema operacional, mas sua grande vantagem é a obtenção de maior desempenho.

Alguns exemplos do uso da abordagem de *container* são encontrados nos sistemas OpenVZ, FreeBSD Jails e LXC.

2.2.4 Aplicação da Virtualização

A virtualização pode ser útil de diversas formas para os ambientes computacionais, a seguir são descritas algumas formas de aplicação da tecnologia de virtualização.

- **Segurança**

  Os processos em execução nas máquinas virtuais não influenciam no ambiente físico da máquina hospedeira, ou *host system*. Garantindo essa independência entre o sistema hospedeiro e o convidado ocorre o isolamento das aplicações e problemas ou *bugs* dos programas não são propagados entre os ambientes físicos e virtuais [Chen and Noble, 2001].

- **Sandbox**

  Hoje uma das técnicas de isolamento de aplicações mais utilizadas e promissoras recebe o nome de *sandbox*. Segundo [Wen et al., 2012] Sandbox provê um ambiente seguro, isolado para execução de aplicações não confiáveis ou de origem desconhecida.

  Ao executar uma aplicação dentro de uma *sandbox*, essa possui acesso total ao sistema, funcionando como se fosse executada nativamente. As aplicações não são visíveis do lado de fora da *sandbox*, ou seja, os demais processos do sistema não enxergam o processo em execução na *sandbox*. Qualquer ameaça executada em ambiente seguro é descartada quando o aplicativo é encerrado. Caso alguma informação obtida em ambiente seguro for necessária para o sistema nativo, mecanismos adicionais são necessários para obtenção dessa informação.

- **Portabilidade**

  Os sistemas virtualizados podem prover ambientes totalmente diferentes e independentes. Isso acaba beneficiando testes com programas que serão utilizados em vários sistemas operacionais diferentes e gera uma independência com o *hardware*, dispensando a utilização de vários equipamentos físicos para efetuar esses testes. É possível também a criação de cenários complexos de teste de software [Chiueh and Brook, 2005].

- **Gerenciamento de Estados**

  Outra vantagem é com relação ao gerenciamento dos estados nas máquinas virtuais. Pois a qualquer momento ela pode ser movida, restaurada, duplicada, etc. Funcionalidades que não são alcançadas tão facilmente em ambientes puramente físicos [Chen and Noble, 2001].

  Quando ocorre um comprometimento do sistema operacional nativo, a recuperação do ambiente e o restabelecimento dos serviços é uma tarefa complexa que pode exigir reinstalação de sistemas com o objetivo de acabar com a ameaça efetuando uma "limpeza" no sistema. Em ambientes virtualizados, essas ações podem ser efetuadas com maior eficiência, por meio de ferramentas que salvam o estado atual do sistema. No caso de um ataque, a máquina virtual pode ser restaurada no ponto recuperação antes da ameaça e após a restauração, pode ser efetuada alguma operação para prevenir que o ataque ocorra novamente de maneira mais rápida que o seu equivalente físico.

- **Consolidação de Servidores**

  A consolidação de recursos é um benefício muito utilizado da virtualização. A possibilidade de simplificação da infraestrutura física de servidores, centralizando diversos servidores de aplicações ou de serviços de rede que poderiam estar divididos em vários ambientes físicos distintos, em um mesmo equipamento. A consolidação de servidores têm como grande vantagem o melhor aproveitamento do *hardware* disponível. Recursos adicionais podem ser alocados, pois a máquina virtual pode ser modificada mais facilmente que a máquina real [Chen and Noble, 2001].

- **Consolidação de Aplicações**

  Da mesma forma que a consolidação de servidores, também é possível consolidar as aplicações num único sistema. Essa abordagem pode ser aplicada em ambientes com programas legados de plataformas de *hardware* incompatíveis. A centralização de aplicações e a facilidade de manutenção justificam a utilização da consolidação de aplicações num único ambiente virtualizado projetado para esse propósito.

- **Desktops**

  A virtualização de *desktops* e aplicações pode ser encarada como o próximo passo para os ambientes virtualizados, centralizando o acesso e aproximando a tecnologia de virtualização do usuário comum proporcionando mobilidade no acesso as aplicações. Com relação a segurança, os ambientes voltados para o usuário final podem ganhar mais confiabilidade devido a facilidade de montagem de políticas de acesso para os usuários [Garfinkel and Warfield, 2007]. Controlando a capacidade de limitar o acesso dos usuários aos recursos como dispositivos de armazenamento externos ou a determinado endereço de rede.

- **Migração de Máquinas Virtuais**

  A migração de máquinas virtuais permite que um sistema convidado seja executado em outro *host* físico de forma transparente, segundo [Clark et al., 2005] o *host* original deve permanecer disponível para atender a certas chamadas de sistema ou acessos à memória em processos migrados. Nesse processo, a migração ocorre de forma completa, incluindo o estado da memória, que é transferido de forma consistente para outro *host* físico. A migração de máquinas virtuais permite o uso de estratégicas de balanceamento de sistemas convidados visando o maior equilíbrio no uso de recursos disponíveis. Nos ambientes de virtualização de grande porte, a entrada, evolução e finalização de máquinas virtuais pode provocar o desequilíbrio de recursos computacionais. Conforme [Muchalski, 2014], balancear os sistemas convidados disponíveis é permitir aos *hosts* um estado de equilíbrio, diminuindo a carga de trabalho dos servidores caso eles estejam sobrecarregados por conta de uma ou mais VMs que podem fazer uso excessivo de recursos como memória, processamento, disco ou rede.

2.3 Gerenciamento de Memória em Ambientes Virtualizados

Uma das principais características de um sistema operacional é gerenciar a memória RAM disponível, assegurando que cada processo e o próprio sistema operacional tenham

a quantidade de memória necessária. Como a memória é um recurso relativamente escasso (em relação aos demais recursos), muitas técnicas foram desenvolvidas para otimizar seu uso, como a memória virtual, bibliotecas dinâmicas compartilhadas, carga de páginas sob demanda, alocação com *copy-on-write*, alocação preguiçosa e compressão de memória [Vahalia, 1996].

Em um sistema virtualizado, cada máquina virtual recebe do hipervisor uma fração da memória da máquina, gerenciada pelo sistema operacional convidado. Os mecanismos internos do SO convidado se limitam à memória da própria máquina virtual, que é isolada das demais pelo hipervisor.

Alguns termos usuais devem ser claramente definidos para uma melhor compreensão: *memória virtual da VM* diz respeito à memória vista pelos processos da máquina virtual (páginas); *memória física da VM* é a memória oferecida pelo hipervisor e gerenciada pelo sistema operacional da máquina virtual; memória de máquina (ou *host memory*) é a memória RAM real disponível no *hardware* e gerenciada pelo hipervisor [VMware Inc., 2010].

Nos sistemas virtualizados, para poder atender a demanda de memória dos sistemas convidados, mecanismos adicionais de alocação de memória são necessários, visando a manutenção do equilíbrio de recursos nos ambientes, a economia de recursos físicos e melhor aproveitamento do *hardware* em ambientes de grande porte com várias máquinas virtuais.

O recurso de *memory overcommit* é um deles e se refere à prática de oferecer um espaço de memória além da memória física disponível, sem qualquer garantia de que essa quantia do armazenamento físico efetivamente exista, sendo um recurso também utilizado nos sistemas operacionais.

Inicialmente pode ser verificado que o *overcommit* represente um risco para o ambiente caso a memória física se esgote. Na prática, o comprometimento de recursos pode ser considerado menor, já que a maioria dos sistemas convidados usa apenas uma pequena parte da memória física alocada para eles, deixando os hipervisores com a tarefa de identificar trechos de memória ociosa e dinamicamente realocar esses para outras VMs que necessitam de mais memória naquele determinado momento [VMware Inc., 2010].

Outras técnicas de melhor aproveitamento e de recuperação de memória como *Ballooning*, *Memory Compression* e *Swap* são explicadas na sequência.

2.3.1 Ballooning

A técnica chamada de *ballooning* está presente em vários hipervisores, e é descrita como sendo uma forma da máquina virtual se comunicar com o hipervisor quando esse está demandando mais memória. Segundo [Wen et al., 2012] devido ao isolamento, o sistema operacional convidado não está ciente de que está sendo executado dentro de uma máquina virtual e não tem conhecimento dos estados das outras máquinas virtuais no mesmo *host* físico. Quando o hipervisor executa várias máquinas virtuais e a quantidade total de memória livre no *host* torna-se baixa, nenhuma das máquinas virtuais pode liberar memória física porque o sistema operacional convidado não está ciente da falta de memória do *host*. Para que o hipervisor seja notificado, é necessária a instalação de um *driver balloon* na máquina virtual que possui a capacidade de interagir com o hipervisor.

Quando o hipervisor está com pouca memória, [Amit et al., 2014] esse define uma quantidade de páginas necessárias ao *driver balloon*. Caso haja um máquina virtual que possua páginas de memória não utilizadas e que poderia ser alocada no host, o sistema operacional convidado indica quais páginas de memória serão desalocadas e o hipervisor pode então transferir

a memória física da máquina virtual para o host. Caso a memória oferecida seja insuficiente, o sistema convidado pode iniciar o processo de *swap* de páginas de memória com o objetivo de liberar mais memória ao *host*. Esta atividade pode potencialmente afetar o desempenho, dependendo da quantidade de memória que necessita ser recuperada.

A Figura 2.4 ilustra o funcionamento do mecanismo *ballooning*, vale ressaltar que o mecanismo é ativado somente quando o *host* está com pouca memória e o *driver balloon* determina quais páginas de memória as máquinas virtuais podem "abrir mão" com o objetivo do servidor fazer *swap* para disco.

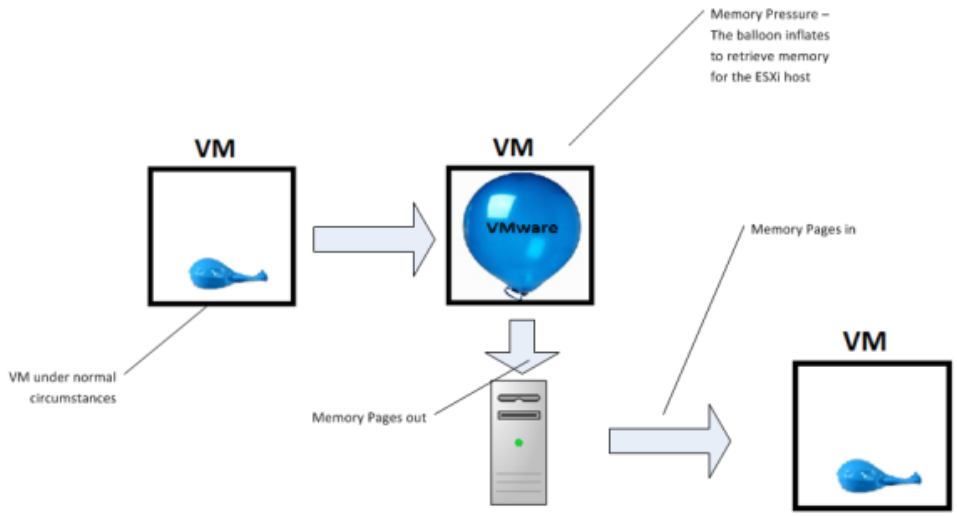

Figura 2.4: Ballooning [VMware Inc., 2010]

2.3.2 Memory Compression

Mecanismos de compressão de memória são utilizados em momentos de dificuldade (falta de memória) do hipervisor. Como qualquer sistema, na falta de memória as páginas vão para a área de troca ou *swap*, caso não seja possível recuperar memória suficiente utilizando *ballooning* ou outras técnicas de gerenciamento de memória, as páginas que iriam para área de troca podem ser comprimidas e armazenadas numa unidade controlada do hipervisor denominada de cache de compressão [Waldspurger, 2002], essa fica localizada na memória do host. Para serem enviadas para o cache de compressão as páginas marcadas para a área de troca necessitam de um percentual de ao menos 50% de compressão, caso esse valor não seja alcançado essa página não é candidata ao cache de compressão [VMware Inc., 2010].

Quando uma página comprimida for necessária a mesma é descompactada e acessada. Por se situar na memória do host, o acesso é mais rápido que o acesso ao disco [Yun et al., 2014]. O mecanismo reserva uma parte da memória para o uso do cache de compressão, sendo um recurso limitado que visa principalmente evitar o processo de paginação para disco, o que resultaria em uma operação de entrada e saída, bem mais custosa para o sistema.

Mecanismos como `zswap` [Zswap, 2013] e `zram` [zRAM, 2014], trabalham dessa forma nativamente no sistema operacional, eles trabalham evitando a entrada de páginas de memória no disco e comprimindo essas páginas por meio de uma área de memória residente na memória RAM, conforme visto anteriormente.

A Figura 2.5 ilustra o processo de armazenamento de páginas e o cache de compressão, nota-se que com o uso do cache de compressão não é utilizada a área de troca:

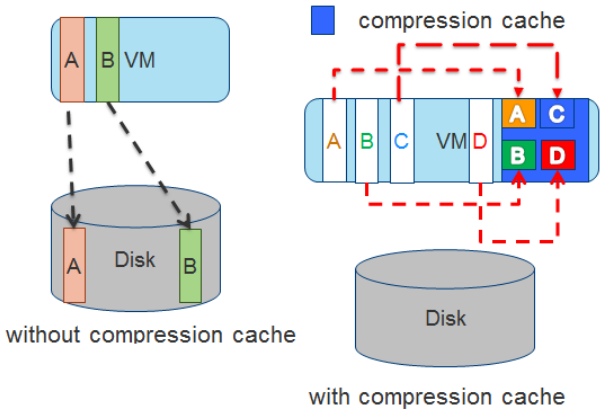

Figura 2.5: Técnica Memory Compression [VMware Inc., 2010]

2.3.3 Swap

Após esgotamento de todas as possibilidades de recuperação de páginas de memória vistas anteriormente e caso não seja possível alocar mais memória para as máquinas virtuais e para o host de virtualização, o uso da área de troca é o último recurso para recuperar memória [Yun et al., 2014]. Cada um dos hipervisores utiliza mecanismos para otimizar esse processo, normalmente é criado um arquivo auxiliar na estrutura de diretórios do hipervidor e nesse arquivo são transferidas as páginas de memória da máquina virtual, liberando essa área de memória que agora está em disco. Nesse processo o sistema convidado não interage, sendo o hipervisor quem decidirá quais páginas serão enviadas para *swap* [Waldspurger, 2002].

O grande problema reside no hipervisor que não possui acesso as páginas que seriam mais recomendadas de serem enviadas para *swap*, portanto uma máquina virtual pode estar acessando uma página no momento que a mesma é enviada para área de troca, afetando drasticamente o desempenho do sistema convidado.

Recomenda-se a utilização de mídias com bom tempo de acesso para área de troca em hipervisores, assim minimizando o impacto no desempenho das máquinas virtuais [Amit et al., 2014], pois o processo afeta significativamente o desempenho do sistema, por esse motivo é sugerido o uso de mídias de acesso como SSD *Solid State Drive* ou *storages* para maior segurança e performance.

2.4 Conclusão

Este capítulo apresentou tópicos relacionados a virtualização, com os tipos, técnicas e aplicações, devido ao contexto onde o trabalho está inserido, foi apresentado também o gerenciamento de memória em ambientes virtualizados. Mesmo havendo mecanismos de recuperação de memória, os ambientes virtualizados necessitam de uma quantidade de memória adicional para controlar o funcionamento do hipervisor, além o *overhead* que a camada responsável pela virtualização impõe.

O próximo capítulo tem por objetivo apresentar mecanismos adicionais de recuperação de memória e analisar as técnicas de compartilhamento de memória e os trabalhos relacionados que tratam do assunto.

Capítulo 3

Mecanismos de Compartilhamento de Memória em Hipervisores

No ambiente de virtualização, os hipervisores utilizam mecanismos para gerenciar a memória virtual disponibilizada para as máquinas virtuais e recuperar memória em caso de sobrecarga ou pressão por parte do *host físico*. Este capítulo apresenta um estudo sobre o mecanismo do compartilhamento de memória nas máquinas virtuais, onde será apresentado o funcionamento do mecanismo e as abordagens propostas na literatura que possuem relação com os objetivos dessa pesquisa. Por fim, uma pequena conclusão desse capítulo.

3.1 Introdução

Partindo da hipótese de que máquinas virtuais executando sistemas operacionais convidados e/ou aplicações similares podem ter muitas páginas de memória idênticas entre si. Alguns hipervisores implementaram mecanismos de compartilhamento transparente de memória RAM entre suas máquinas virtuais, visando diminuir a demanda total de memória RAM no sistema físico [Chang et al., 2011]. É de se imaginar que máquinas virtuais executando os mesmos sistemas operacionais, bibliotecas e/ou aplicações (situação típica em ambientes de *clouds*) possuam muitas páginas idênticas entre si, que poderiam ser compartilhadas.

Várias técnicas visam evitar páginas físicas redundantes, ou seja, que tenham o mesmo conteúdo. Nesses casos, as páginas virtuais que têm um mesmo conteúdo são mapeadas em uma única página física (*frame*), reduzindo a quantidade de memória real usada pelo sistema.

Os mecanismos de compartilhamento implementados nos hipervisores "fundem" páginas idênticas das várias máquinas virtuais em um único quadro de memória da máquina, usando uma abordagem *copy-on-write*[1], de forma transparente para os sistemas convidados. Essa abordagem permite eliminar páginas de memória redundantes que não seriam detectadas pelos sistemas operacionais convidados, uma vez que ocorrem entre máquinas virtuais distintas [Sindelar et al., 2011].

O compartilhamento de páginas pode ser *intra-VM* ou auto-compartilhamento, quando ocorre entre páginas idênticas dentro de uma mesma máquina virtual, ou *inter-VM*,

[1]Copy-on-write (COW) é uma forma de compartilhamento de memória utilizada pelos sistemas operacionais para referenciar as mesmas áreas de memória entre processos. Evitando assim a cópia de grandes áreas de memória, até o momento que determinada página é alterada. [Fábrega et al., 1995]

quando ocorre entre páginas idênticas em máquinas virtuais distintas. Deve-se observar que, enquanto um sistema operacional convidado só consegue gerenciar o compartilhamento intra-VM, o hipervisor pode tratar ambos [Barker et al., 2012].

A seguir são apresentados os principais mecanismos de compartilhamento de memória presentes em hipervisores como *Content-Based Page Sharing* (CBPS), *Transparent Page Sharing* (TPS), *Kernel Samepage Merging* (KSM) e *Xen Difference Engine* (Xen DE).

3.2 Content-Based Page Sharing

Hipervisores populares de código aberto ou comerciais utilizados atualmente empregam mecanismos de compartilhamento de memória baseados no príncipio de CBPS(*Content-Based Page Sharing*). Os hipervisores periodicamente varrem a memória em busca de páginas duplicadas, identificando os conteúdos das páginas de forma transparente ao sistema operacional convidado. Segundo [Waldspurger, 2002] essa técnica de compartilhamento pode representar até 33% de economia de memória no sistema. A técnica utiliza a premissa que várias máquinas virtuais residentes no mesmo *host* físico podem compartilhar páginas de memória, CBPS realiza o compartilhamento por meio da técnica de deduplicação dos dados em memória.

O mecanismo de comparação das páginas utiliza um valor de *hash* atribuído a cada uma das páginas de memória alocada, caso haja correspondência desse *hash*, essas páginas se tornam candidatas ao compartilhamento. Posteriormente essas são comparadas entre si por meio de verificação byte a byte. Caso a verificação byte a byte não resulte em correspondências o mecanismo não é aplicado. Se os valores verificados na comparação forem correspondentes, as páginas podem ser compartilhadas sem afetar as máquinas virtuais. Quando ocorre a situação de páginas duplicadas num *host* de virtualização, as cópias redundantes dessa página de memória são eliminadas [Chen et al., 2014], ficando somente uma das páginas em memória e as máquinas virtuais que necessitam dessa página acessam essa por meio de apontamento criado pelo hipervisor.

O uso desse mecanismo é recomendado em máquinas virtuais que possuam semelhanças entre si, com isso seria gerada uma quantidade significativa de páginas compartilháveis. Em ambientes de grande porte, esse recurso é explorado visando a otimização do uso de memória. O mecanismo de compartilhamento de memória sugere que os *hosts* possuam o máximo de similaridade possível, principalmente no sistema operacional utilizado [Barker et al., 2012]. Esses conceitos são base para os mecanismos de compartilhamento utilizados por grande parte dos hipervisores, cada qual implementa o conceito de CBPS juntamente com outros mecanismos adicionais desenvolvidos para a necessidade específica do hipervisor.

O uso de *hashes* para comparação do conteúdo das páginas é justificável, pois segundo [Gröninger, 2011] seria impraticável o uso da comparação *byte a byte* para verificar as correspondências pelo seu custo computacional elevado. No primeiro momento é realizada a comparação de *hashes* e posteriormente é utilizada a comparação *byte a byte* apenas para evitar que páginas com mesmo *hash* e conteúdo diferentes sejam compartilhadas ocasionando uma colisão de *hashes*.

A Figura 3.1 ilustra o funcionamento do mecanismo CBPS. No primeiro momento (a) o mecanismo de compartilhamento não está ativo, as duas VMs possuem acesso de leitura e escrita na página. Posteriormente, as duas VMs estão compartilhando memória e seu atributo de acesso passa a ser de somente leitura (b) e a cópia redundante foi eliminada. No momento

(c), o *Guest* B efetuou uma alteração na página de memória e essa deixou de ser compartilhada para essa VM permanecendo compartilhada para o *Guest* A.

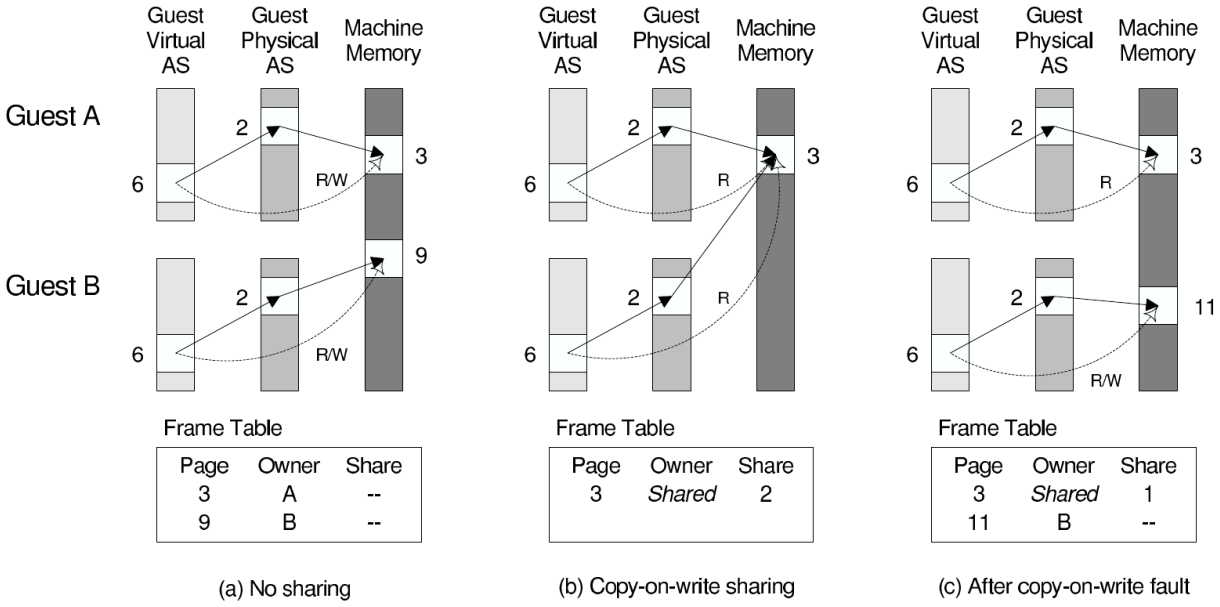

Figura 3.1: Content-Based Page Sharing [Vrable et al., 2005].

3.2.1 Transparent Page Sharing

O mecanismo TPS (*Transparent Page Sharing*) foi proposto pela primeira vez no sistema Disco [Bugnion et al., 1997] e é utilizado no hipervisor VMware e segundo [Waldspurger, 2002] deve ser utilizado quando o hipervisor possuir CPU disponível para o mecanismo. O TPS lê dados de páginas de memória de um espaço de memória *copy-on-write* e verifica se os mesmos dados de página já estão presentes na memória principal. Caso haja páginas correspondentes, o TPS cria um mapeamento compartilhado entre os sistemas convidados para a página existente.

Nesse momento as páginas compartilhadas ficam como atributo de somente leitura para as máquinas virtuais. Caso haja necessidade de alterar as páginas compartilhadas por um dos sistemas convidados, o hipervisor copia a página necessária e realoca esse espaço de memória que deixa de ser compartilhado para aquele sistema convidado específico, permanecendo compartilhado nos demais. A técnica utilizada é a de *Copy-on-Write* (COW), essa técnica gera um *overhead* quando utilizada, pois quando ocorre a necessidade de escrever na memória compartilhada é gerada uma falta de página antes do processo de cópia.

A página compartilhada pelo hipervisor é disponibilizada para as máquinas virtuais e registrado numa tabela de *hashes* mantida pelo hipervisor eliminando as duplicidades [Chen et al., 2014]. Outra questão relevante desse mecanismo de compartilhamento é que esse estará sempre ativo, comparando os valores dos *hashes* das páginas com a tabela de páginas global do sistema. O mecanismo de comparação verifica páginas de tamanho fixo de 4 KB [VMware Inc., 2010]. Ignorando super-páginas (*huge pages*, com 2 MB), pois o custo de comparação das super-páginas é elevado e a probabilidade de encontrar réplicas diminui consideravelmente.

Como as operações de varredura, cálculo de *hash* e comparação de páginas demandam muito processamento, o intervalo de varredura é definido de forma a não degradar o desempenho do sistema. Os parâmetros `Mem.ShareScanTime` e `Mem.ShareScanGHz` controlam a intensidade do mecanismo na busca por oportunidades de compartilhamento. [VMware Inc., 2010].

3.3 Xen Difference Engine

O mecanismo Difference Engine [Gupta et al., 2010], utilizado experimentalmente pelo hipervisor Xen, funciona baseado no conceito CBPS, ele adiciona algumas funcionalidades ao mecanismo CBPS tradicional. No processo de comparação de páginas, quando a mesma impressão digital de páginas de memórias representado pelo *hash* é encontrada, ele compara o conteúdo das duas páginas e compartilha se as mesmas forem idênticas. Até aí nenhuma novidade, porém o *differencial engine*, não realiza somente a desduplicação das páginas de memória, mas também comprime as páginas quando essas são quase idênticas e as diferenças entre elas são tratada como um *patch*. As páginas quase idênticas são mescladas, com o conteúdo das duas e suas diferenças.

O mecanismo de compressão de memória 2.3.2 é presente nessa abordagem e é utilizado quando determinada página não está ativa por um longo período de tempo. No estudo [Gupta et al., 2010] é apresentado que o uso do mecanismo *Difference Engine*, pode reduzir o consumo de memória em ambientes com *workload* homogêneo em até 90%. O mecanismo não é aplicado nas versões oficiais do hipervisor Xen.

Na figura 3.2 é feito um comparativo do uso sem o compartilhamento de memória, com o compartilhamento de memória, adicionando o *patching* e a compressão, de acordo com imagem, ocorre a economia de pelo menos 50% da memória física.

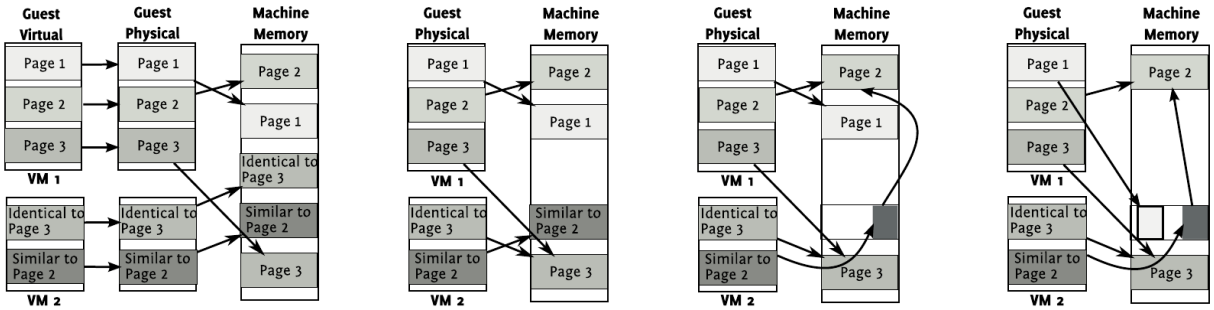

Figura 3.2: Comparativo Xen Differential Engine [Gupta et al., 2010].

3.4 Kernel Samepage Merging

O núcleo Linux abriga um hipervisor nativo chamado KVM (*Kernel-based Virtual Machine*) [Rachamalla et al., 2013], que implementa um mecanismo de CBPS denominado *Kernel Samepage Merging* (KSM) [Arcangeli et al., 2009]. O KSM não é aplicado somente no contexto das máquinas virtuais e está presente na árvore do *kernel* Linux desde a versão 2.6.32, sendo necessário apenas ser habilitado.

O KSM utiliza a chamada de sistema `madvise` [Madvise, 2015], que permite a uma aplicação notificar o *kernel* sobre quais páginas de memórias são candidatas ao compartilhamento. Assim o *kernel* pode utilizar mecanismos para antecipar as leituras dessas páginas e o armazenamento de dados em cache de forma apropriada para aquela aplicação. Com isso, o `madvise` pode evitar o desperdício de CPU ao varrer todas as páginas do sistema, utilizando portanto, as áreas virtuais registradas pelos aplicativos que provavelmente contêm páginas duplicadas.

A chamada *madvise* não influencia em como a aplicação é executada, porém pode influenciar no desempenho dessa aplicação. Portanto ela funciona como um conselheiro ao *kernel* do sistema e o *kernel* possui liberdade para ignorar esse conselho.

O KSM trabalha nas áreas de endereços de memória que tenham sido indicadas a serem candidatas prováveis para o compartilhamento. O *daemon* ksmd periodicamente realiza verificações nas páginas buscando identificar as páginas duplicadas, para assim liberar essa área em memória alocada em duplicidade. Os parâmetros `sleep_time` e `scan_rate` afetam a intensidade na busca por correspondências do mecanismo KSM.

As páginas são "fundidas" (*merged*) e marcadas como somente leitura. Por se basear no modelo CBPS, caso um sistema convidado necessite alterar essa página ele irá receber uma cópia dessa página que deixa de ser compartilhada para aquela máquina virtual.

O KSM possui mudança no processo de comparação das áreas em comparação com o CBPS tradicional. Ao invés da geração de *hashes* das páginas e posterior varredura da memória comparando por similaridades entre elas, o mecanismo do KSM permite as aplicações registrarem as áreas de memória que são susceptíveis de conter páginas duplicadas. Segundo [Arcangeli et al., 2009], o armazenamento das páginas ocorre em duas estruturas de dados de árvore rubro-negra (*Red-black tree*) utilizada em buscas binárias balanceadas[2]. Uma dessas árvores é transitória e é chamada de instável, ela é utilizada para armazenar páginas que ainda não podem ser consideradas estáveis, são as páginas candidatas à "fusão", as páginas nessa árvore não são protegidas contra gravação. Já a árvore chamada de estável, armazena as páginas que foram candidatas para ser estáveis (não são voláteis) e foram registradas pelo KSM. Para identificar se uma página é volátil ou não-volátil, KSM usa um *checksum* de 32 bits. Quando uma página é registrada, seu *checksum* é calculado e armazenado juntamente com a página.

Nas verificações posteriores, caso um *checksum* recentemente calculado for diferente do gerado anteriormente, significa que a página está mudando e não é, portanto, um bom candidato para a fusão. Após isso, o próximo passo é comparar uma página de memória para verificar se a mesma página pode ser encontrado na árvore de páginas estáveis. Posteriormente é efetuada um `memcmp` (comparação byte-a-byte) entre a página que está no nó e as que estão na árvore. Segundo [Chen et al., 2014] esse modo de verificação gera um considerável overhead de CPU nas operações de comparação. O `memcmp` trabalha com o retorno dos valores 0 -1 e 1.

- 0 - A comparação entre as páginas resultou em uma correspondência;
- 1 - A página candidata é maior que as página de nó atual ;
- -1 - A página candidata é menor que as páginas do nó atual.

[2]É uma árvore de busca binária com um bit extra de armazenamento por nó: a sua cor, que pode ser vermelho ou preto. Ao restringir a forma como os nós podem ser coloridos desde a raiz até uma folha, árvores vermelhas e pretas garantem que o caminho mais longo da raiz a qualquer folha não é mais longo que o dobro do caminho mais curto da raiz a qualquer outra folha naquela árvore [Cormen et al., 2001]

O fluxo do KSM está descrito na figura 3.3.

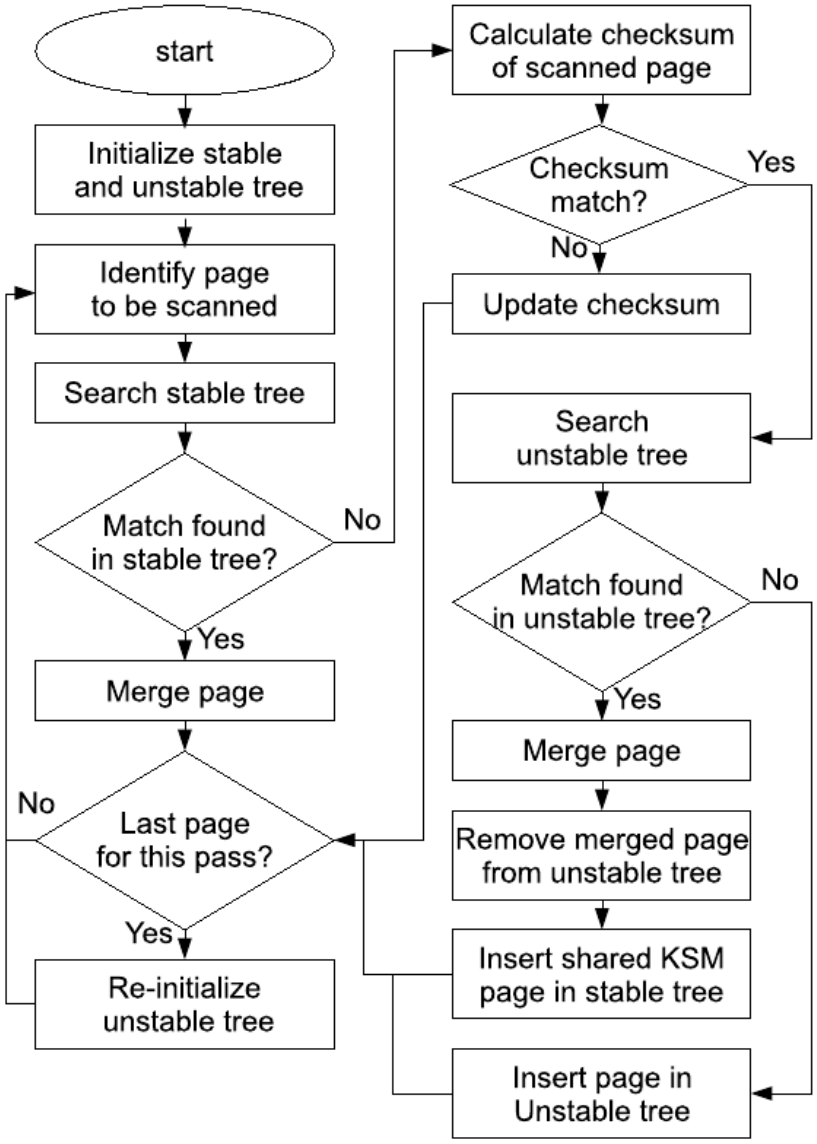

Figura 3.3: Fluxo do Mecanismo KSM [Arcangeli et al., 2009].

3.5 Trabalhos Relacionados

Esta seção apresenta alguns trabalhos de pesquisa relacionados a eficiência dos mecanismos de compartilhamento de páginas de memória em hipervisores, esses trabalhos estudaram, utilizaram, implementaram e avaliaram o compartilhamento de memória para determinada finalidade.

3.5.1 Design and implementation of page sharing scheme between guests in virtualization environments

O artigo [Kim et al., 2009], apresenta um mecanismo alternativo ao CBPS, envolvendo o compartilhamento baseado em *shadow pages*. O mecanismo apresentado no estudo parte da premissa que em condições normais, cada sistema convidado deve utilizar seu próprio espaço de memória para realizar *caches* de páginas de arquivos abertos por processos no sistema convidado. No estudo apresentado, todos os sistemas convidados podem utilizar somente uma página física para cada página de arquivos binários comuns aos sistemas. Esse mecanismo está aplicado e avaliado no hipervisor Lguest[3].

Um exemplo apresentado no estudo seria utilizando dois sistemas convidados utilizando o binário `bash`, o host físico armazena (retém) e gerencia as páginas de memória do arquivo. Se ocorrer uma falta de página no espaço de endereço mapeado para o arquivo, o *host* não comunica a falta para o sistema convidado e cria um apontamento da página comum para o *cache* de páginas. Quando a página requerida não está na memória, o hipervisor aloca uma nova página de memória e armazena o endereço da página na *shadow page* e comunica a falta de página para o *kernel* do sistema convidado.

O resultado apresentado nesse artigo apresenta um número considerável de páginas que não necessitaram ser alocadas, para o caso da biblioteca de sistema `libc6`, chegando a 1000 páginas de memórias "salvas", resultado numa economia de 4MB.

3.5.2 Sharing-aware algorithms for virtual machine colocation

O artigo [Sindelar et al., 2011], examina algoritmos de alocação de máquinas virtuais em ambientes de virtualização voltados para a computação em nuvem, os sistemas convidados são balanceados na entrada (colocação) de uma máquina virtual no ambiente para que essa não afete o equilíbrio de carga nos *hosts* físicos. Essa avaliação leva em conta recursos de CPU, memória, armazenamento, o potencial de compartilhamento de páginas de memória por meio de uma estrutura hierárquica e a carga de trabalho nos *hosts* físicos que compõem o ambiente. O algoritmo proposto determinada onde uma máquina virtual será alocada em determinado *host* sem afetar o equilíbrio do ambiente. Nesse estudo é examinado o compartilhamento inter-VM, destacando a necessidade de utilização do sistema operacional do mesmo tipo, pois assim foram alcançados os melhores resultados do estudo, comparando com com sistemas operacionais de famílias diferentes ou versões diferentes. Segundo os autores, duas máquinas virtuais, sendo uma com Windows XP e outra com Windows 7 não são tão efetivas como duas máquinas Windows 7. Nesse estudo, o mecanismo de comparação utilizando também foi por meio de *memory traces*.

A figura 3.4 representa a estrutura hierárquica para o sistema operacional que será colocada na estrutura de virtualização, cada passagem na estrutura da árvore representa uma similaridade adicional que implica no maior compartilhamento, fatores como arquitetura, versão e qual o sistema operacional são avaliados.

[3]Lguest é projetado para ser um hipervisor simples de usar, modificar e com baixo consumo de recursos com poucas linhas de código (cerca de 5000 linhas)[Lguest, 2007]

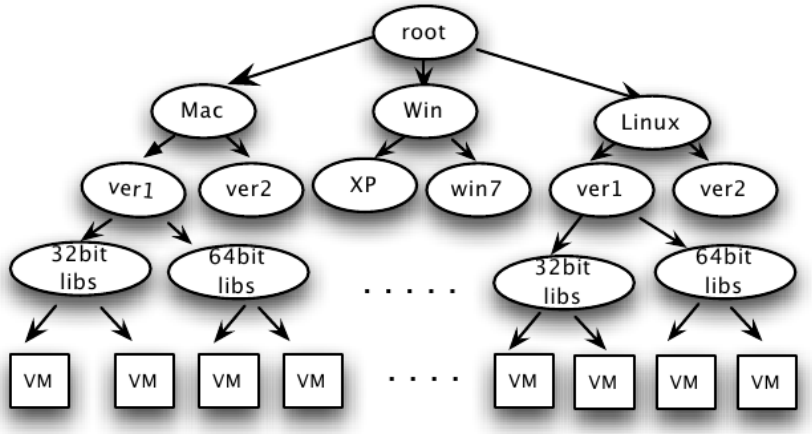

Figura 3.4: Hierarquia de Similaridades [Sindelar et al., 2011].

3.5.3 Alocação de Máquinas Virtuais em Ambientes de Computação em Nuvem Considerando o Compartilhamento de Memória

O estudo [Muchalski, 2014] também utiliza o compartilhamento de memória como parâmetro para alocação de máquinas virtuais no ambiente de computação em nuvem, além dos recursos de CPU, memória, disco e uso da banda de rede. O algoritmo *VectorAlloc* com compartilhamento é comparado com outros algoritmos de colocação de VMs nos cenários propostos. A forma de avaliação desse estudo é por meio do aplicativo *CloudSim*, nele foi possível inserir o algoritmo proposto e os cenários das máquinas utilizadas.

O algoritmo possui um indicador do fator do compartilhamento(α), porém a forma de cálculo se baseia em níveis de similaridade organizados de forma hierárquica entre os sistemas convidados, com valores de 0 e 1 para cada nível, que somados determinam o valor do indicador. O uso do algoritmo *VectorAlloc* com compartilhamento de memória apresentou um melhor equilíbrio no uso de memória e CPU em comparação ao *first fit* e o *VectorAlloc* sem o compartilhamento, conforme visto nas figuras 3.5 e 3.6.

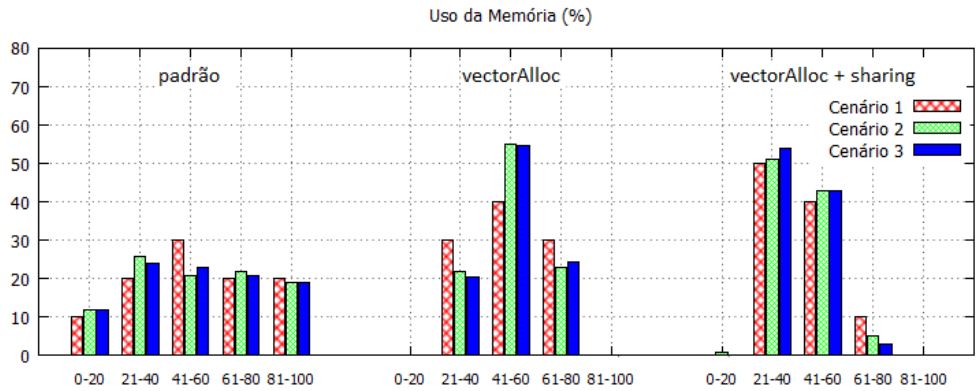

Figura 3.5: Comparativo do Uso de Memória [Muchalski, 2014]

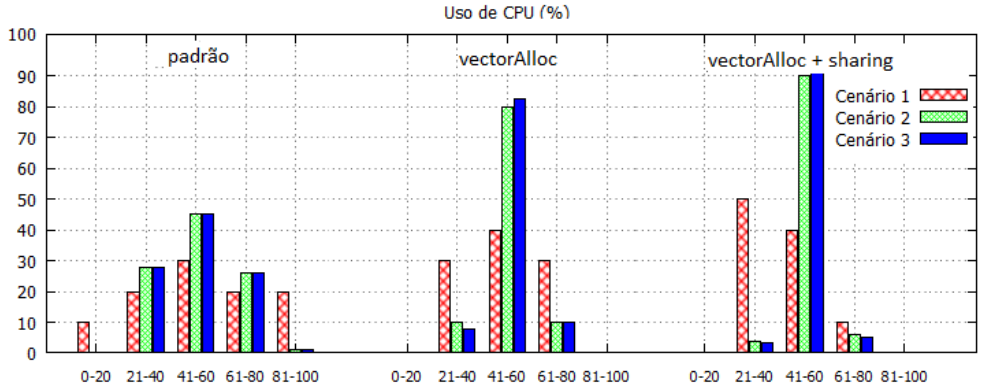

Figura 3.6: Comparativo do Uso de CPU [Muchalski, 2014]

Ao final do estudo é sugerido que o valor do fator de compartilhamento pode ser mais preciso ou até mesmo ser avaliada a aplicação em execução na VM para efetuar a colocação de determinada VM, pois a aplicação não é considerada no cálculo de (α).

3.5.4 Memory Buddies: Exploiting Page Sharing for Smart Colocation

O estudo [Wood et al., 2009], apresenta a problemática que não é possível saber a efetividade do compartilhamento de memória antes de determinado sistema ser colocado em funcionamento e do hipervisor detectar a ocorrência de páginas duplicadas. É proposta uma estratégia composta por um controlador e um núcleo em cada um dos sistemas convidados que gera um *hash* para cada página de memória das VMs, esses *hashes* são avaliados pelo controlador executando em um sistema separado e dedicado que irá analisar e organizar as máquinas virtuais para que possa haver o máximo de possibilidades de compartilhamento de memória de acordo com os sistemas disponíveis para analisar qual seria a melhor distribuição das VMs nos *hosts* físicos. Na Figura 3.7 a arquitetura do mecanismo é representada, valendo ressaltar a separação dos ambientes de controle e execução das VMs.

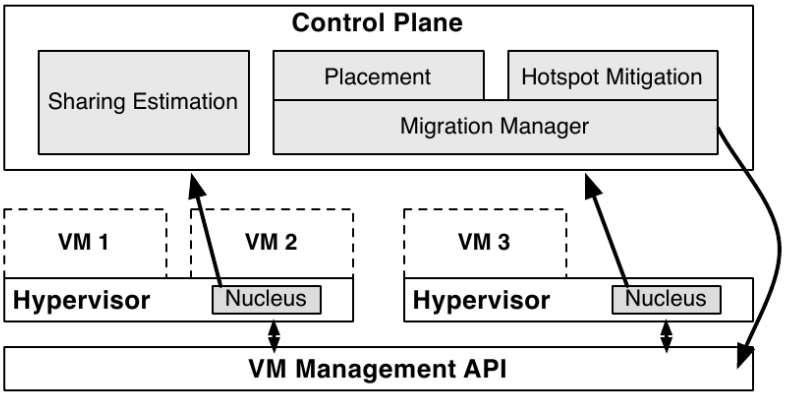

Figura 3.7: Arquitetura Memory Buddies [Wood et al., 2009].

Foram utilizadas cargas de trabalho heterogêneas para estudar o funcionamento do mecanismo de acordo com a aplicação executada. Ao final do estudo, foi possível economizar 17% de memória, balanceando as VMs com maior potencial de compartilhamento, esse processo ocorre após a VM ser colocada em funcionamento, ficando situada numa área de avaliação e posteriormente sendo migrada para o melhor local para essa VM aproveitar melhor o compartilhamento de memória.

3.5.5 Satori: Enlightened page sharing

O trabalho [Miłós et al., 2009] apresenta o mecanismo Satori, com o objetivo de analisar as possibilidades de compartilhamento em páginas de memória com baixo ciclo de vida e com menor *overhead* possível, nesse estudo, os sistemas convidados são para-virtualizados para otimizar as possibilidades de compartilhamento sendo uma mudança em comparação a abordagem tradicional de buscas periódicas por correspondências.

Uma questão levantada no artigo é sobre a reutilização das áreas de memórias recuperadas. A abordagem comum consiste em adicionar a memória que deixou de ser utilizada em um conjunto global, que pode ser usado para criar novas VMs. No entendimento dos autores, apenas as VMs envolvidas no processo de compartilhamente deveriam poder utilizar os benefícios das páginas de memória recuperadas, criando um incentivo para VMs compartilharem memóia. Satori questiona sobre a possibilidade de compartilhamento para cada VM por segundo. É mantido um *ranking* com os direitos de utilização de determinada página compartilhada, que não permanece constante dependendo da evolução de determinado sistema e também se foram encontradas duplicatas adicionais dessa página podendo alterar o direito de uso da página compartilhada. Assim, um das propostas do estudo seria a distribuição da memória recuperada proporcionalmente de acordo com a quantidade compartilhada com cada uma das VMs. Os sistemas convidados reivindicam suas páginas compartilhadas por meio do mecanismo *balooning*. O balão esvazia e libera páginas adicionais ao sistema convidado.

3.5.6 Analyzing Shared Memory Opportunities in Different Workloads

O trabalho [Gröninger, 2011] avalia o desempenho do hipervisor KVM e as oportunidades de compartilhamento com cargas de trabalho reais e *benchmarks* SPEC. Os dados de compartilhamento foram extraídos das máquinas virtuais por um módulo de núcleo dedicado, essa escolha possibilita a análise dos *hashes* das páginas sem que haja a necessidade de realização de *dumps* de memória na rede ou em disco, a justificativa do autor é que esses processos são muito custosos, pois além de gerar esse *dump* o cálculo de *hashes* é efetuado posteriormente, o que poderia feito pelo próprio *kernel*. Com o método utilizado, foi possível baixar o tempo para análise de *hashes* em aproximadamente 100 vezes, além disso é possível também armazenar o *hash* de determinada página de memória juntamente com informações sobre a aplicação que utilizou essa página.

O estudo buscou analisar oportunidades de compartilhamento não exploradas em diferentes espaços de memória. O compartilhamento de memória pode ser de vários tipos, tratando de páginas de memória anônimas [4] e também pode ser compartilhado o mapeamento do conteúdo de um arquivo ou dispositivo mapeado diretamente para a memória, chamado de *named*

[4] São mapeamentos em memória não sendo associado com nenhum arquivo ou dispositivo

pages[5]. Esses dois métodos podem ser explorados pelo sistema operacional. Porém o KSM só consegue explorar as páginas anônimas.

Um exemplo do estudo utiliza dois programas distintos que abrem vários arquivos (*inodes* diferentes) com o mesmo conteúdo e mapeiam esses em espaço de memória, o KSM não consegue compartilhar essas páginas, pois essas são páginas mapeadas de um arquivo e não anônimas. Dessa forma o KSM não consegue aproveitar esse compartilhamento.

A tarefa de compilação de *kernel* também é avaliada no estudo e por ser uma operação que gera operações de entrada e saída consideráveis, pois acessam muitos arquivos e executa a compilação gerando os programas binários, esse I/O excessivo acabam sempre reconstruindo o cache dos arquivos, dificultando o compartilhamento de páginas de curta duração, foi constatado que a maioria das páginas poderiam ser compartilhadas entre o sistema convidado e o cache de arquivo do sistema hospedeiro.

Já no caso do SPEC, foi utilizado um *benchmark* que trabalha exclusivamente com dados em memória, nesse caso apesar das grandes oportunidades de compartilhamento, a eficiência do KSM foi de aproximadamente 1/3 das páginas alocadas, constatando que as oportunidades de compartilhamento de curto período não são aproveitadas pelo KSM, apesar de não considerar as *zero pages*, essas de certa forma aumentam o tempo de comparação das páginas, agravando essa questão.

Esse trabalho conclui que muitas possibilidades de compartilhamento são ignoradas. Um dos motivos apontados foi devido ao curta período de duração e pelo KSM não aproveitar efetivamente o compartilhamento de *named memory*.

3.5.7 Share-a-meter: An empirical analysis of KSM based memory sharing in virtualized systems

O artigo, [Rachamalla et al., 2013], apresenta uma análise sobre o mecanismo KSM na sua forma de operação, identificando e explorando as possibilidades de compartilhamento. No estudo, é estimado o impacto das configurações do mecanismo KSM como o intervalo de busca (*sleep time*) e as quantidade de páginas que serão examinadas (*scan rate*), a investigação é feita analisando o impacto dessa operação no uso de recursos como CPU e consumo de memória. É investigado também o impacto das aplicações no processo de compartilhamento e consequemente quais deveriam ser as melhores formas de uso das configurações do KSM, analisando cargas de trabalho que possuem diferentes padrões de uso de memória. Uma das cargas estudadas utilizou três VMs de 1 GB de memória cada, essas VMs são inicializadas juntas, posteriormente ocorre uma carga sintética de 512 MB de memória, em tempos distintos. O compartilhamento é analisado no momento da alocação e sua evolução no tempo, conforme figura 3.8, essa carga analisa como o sistema se comporta gerenciando grande quantidade de páginas da árvore estável do KSM sendo comparadas com as páginas que estão sendo alocadas no momento da carga. É avaliado também a grande quantidade de páginas em transição das árvores instáveis para a estável e vice-versa.

O estudo propõe cenários onde seu uso pode "salvar" a quantidade de memória suficiente para evitar que haja um uso excessivo de memória e não o compartilhamento máximo utilizado.

A forma de avaliação da possibilidade de economia de memória (*savings*) também é por meio de *traces* de memória usando a ferramenta `mtrace`, esse valor é comparado com

[5]Normalmente são compartilhadas juntamente com o cache de páginas de arquivos acessados

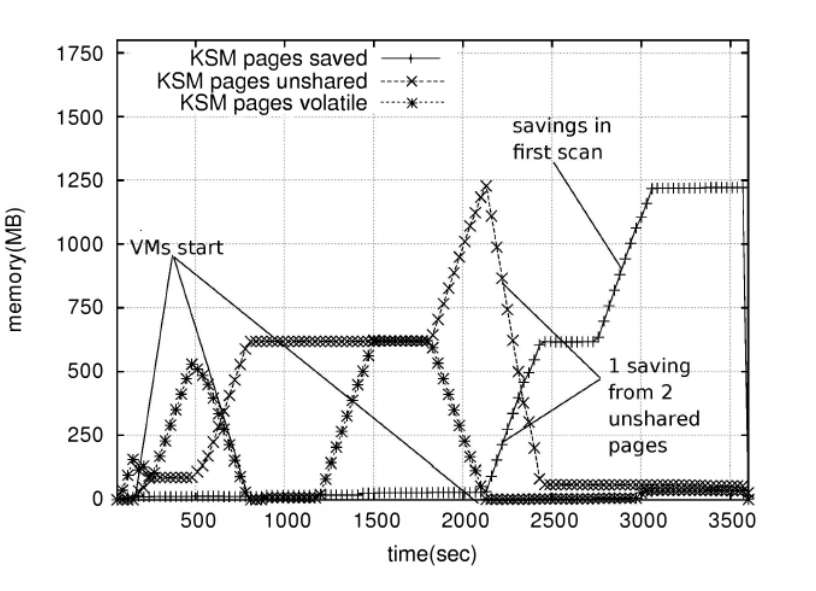

Figura 3.8: Evolução do compartilhamento no KSM [Rachamalla et al., 2013]

o compartilhamento real do KSM, executando cargas homogêneas e heterogêneas entre nas máquinas virtuais. Para realização do estudo é aplicado um *patch* no módulo do KSM para adicionar indicadores na estrutura do KSM.

Na conclusão do estudo é percebido que não há nenhuma configuração universal que maximize a economia de memória com uma sobrecarga mínima para todas as cargas de trabalho. O valor baixo de *scan rate* foi capaz de capturar oportunidades de compartilhamento de longa duração e com baixa atualização, já com o *scan rate* maior foi possível de aproveitar o compartilhamento de páginas com ciclo de vida baixo e com muitas atualizações.

3.5.8 Empirical Study of Memory Sharing in Virtual Machines

O estudo [Barker et al., 2012] estuda os mecanismos de compartilhamentos de memória, em aplicações do mundo real, o objetivo principal desse trabalho é apresentar os fatores que impactam nos potenciais de compartilhamento de memória, estudando as cargas dos programas e os dados armazenado em memória.

A análise dos dados em memória é feita por meio de *traces* ou "rastros" de páginas de memória, capturando todas as páginas com potencial de compartilhamento e comparando essas entre si. Foram utilizados nos testes aproximadamente 50 máquinas entre servidores e estações de trabalho, efetuando *snapshot* da memória de 30 em 30 minutos durante uma semana. As aplicações escolhidas para o ambiente *desktop* foram editores de texto, planilhas eletrônicas e navegador internet, para o ambiente server foram realizadas medições com aplicações LAMP (Linux, Apache, Mysql e PHP).

O mecanismo *self-sharing* ou compartilhamento *intra-VM* representa um percentual maior de possibilidade de compartilhamento, chegando a 80% do total compartilhado, outros dados apresentados no estudo incluem uma média de 14% de compartilhamento envolvendo intra-VM. Os resultados envolvendo *inter-VM sharing* foram baixos em comparação ao intra-VM chegando à uma média de 2% somente, não ultrapassando 6%. Grande parte das páginas

de memória compartilhadas foram obtidas principalmente em aplicações que utilizam interface gráfica do ambiente *desktop*.

O autor ainda argumenta que o compartilhamento *inter-VM* é recomendado principalmente em ambientes com a ocorrência do mesmo sistema operacional em várias máquinas virtuais no mesmo *host* físico, pois nos resultados o compartilhamento em sistemas operacionais diferentes de mesma família, representa um percentual pequeno em comparação ao obtido entre várias máquinas virtuais do mesmo tipo utilizando compartilhamento *intra-VM*. Contrariando artigos apresentados que sugeriram que a maioria do compartilhamento de páginas seria devido a bibliotecas do sistema operacional que são comuns em várias máquinas e aplicações.

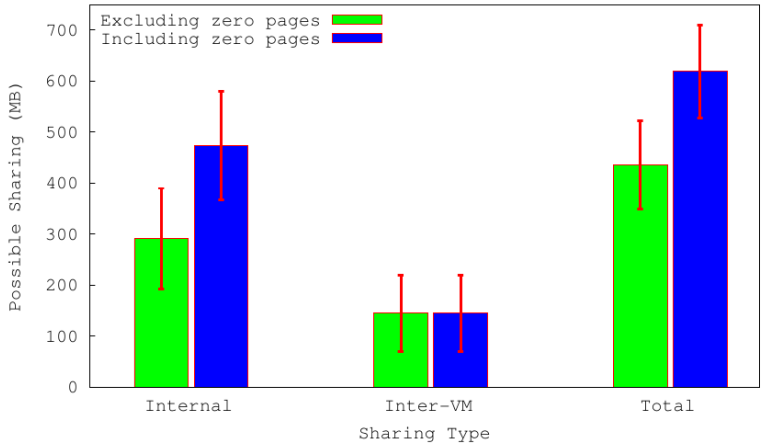

Figura 3.9: Comparativo Inter-VM e Intra-VM [Barker et al., 2012].

Na Figura 3.9 é feita a comparação entre *intra-VM* e *inter-VM* executando uma compilação de *kernel*. O autor inclui os dados relativos a *zero pages* para comparar o impacto dessa no compartilhamento, ao final desse experimento conclui-se que o compartilhamento realizado dentro do próprio sistema operacional é mais eficaz do que por meio de outras VMs e o hipervisor.

As causas apontadas para o elevado *intra-VM* sharing no estudo em ambientes Linux apontam para a maior parte do compartilhamento de memória é realizado por meio de processos que utilizam interface gráfica (GUI), sendo superior a 30% e o compartilhamento dos outros processos não ultrapassa 20%. Portanto o compartilhamento em sistemas *desktop* foi significativamente superior. Com relação ao ambientes com sistemas Windows, aproximadamente 100 MB de diferenças comparando sistemas Windows XP e Windows 7, portanto sistemas de mesma família podem compartilhar várias páginas de memória em aplicações com modo gráfico.

Na conclusão do estudo, é apontado que os sistemas de mesma família possuem um peso maior no compartilhamento, seguido pela aplicação em execução que possui um peso maior que a versão ou a arquitetura do sistema operacional convidado. Também é apontado que a técnica ASLR [6] dificulta o processo de compartilhamento, mas mesmo assim é possível ainda o compartilhamento de páginas.

[6] ASLR busca a proteção de ataques, embaralhando o espaço de memória.

3.6 Conclusão

Ao final desse capítulo, podemos evidenciar que nosso estudo difere de [Kim et al., 2009], pois não propõem avaliação em mecanismos de compartilhamento diferentes de CBPS. Também difere do proposto em [Sindelar et al., 2011] e [Muchalski, 2014] que utilizam o compartilhamento de memória para alocação de máquinas virtuais no ambiente da computação em nuvem e não estudam o compartilhamento em si. O estudo [Wood et al., 2009] e [Miłós et al., 2009] alteram o ambiente de virtualização para otimizar o compartilhamento de memória, a mesma situação é percebida em [Rachamalla et al., 2013] e [Gröninger, 2011]. O presente estudo avalia os mecanismos de compartilhamento, sem alteração da estrutura do sistema convidado. [Rachamalla et al., 2013] apresenta cenários para recuperação de uma quantidade de memória suficiente para evitar o uso excessivo de memória e não o compartilhamento máximo possível, ainda nesse estudo, é avaliado somente o hipervisor KVM e não é feito o comparativo com outros hipervisores, o que também é verificado em [Gröninger, 2011]. Com relação ao estudo [Barker et al., 2012], este estimou apenas o potencial teórico de compartilhamento por meio de imagens de memória das máquinas virtuais, sem avaliar o compartilhamento efetivo alcançado por hipervisores reais. Não é aplicada a abordagem prática que é proposta nesse estudo. [Barker et al., 2012] aponta como sendo a aplicação em execução após o sistema convidado, sendo mais relevante para o compartilhamento do que a família do sistema operacional, essa hipótese será verificada em nosso estudo como forma de poder contribuir com o cálculo de (α) presente no estudo [Muchalski, 2014].

Capítulo 4

Metodologia de Experimentação

Com o objetivo de fornecer os fundamentos necessários para realização da pesquisa e auxiliar no entendimento do problema foram apresentados os capítulos anteriores. Neste Capítulo é apresentada a descrição técnica de como foi desenvolvido o trabalho com a metodologia de avaliação dos mecanismos de compartilhamento de memória.

4.1 Introdução

A presente pesquisa consiste em elaborar um estudo para avaliar e comparar os mecanismos de compartilhamento de memória em hipervisores. Esses mecanismos têm o objetivo de proporcionar economia do recurso de memória RAM, já devidamente identificado como sendo um dos grandes limitadores do número de máquinas virtuais em execução em determinado *host* de virtualização. Neste capítulo é apresentada a metodologia para avaliação desses mecanismos com o ambiente de testes, os cenários utilizados no estudo e as cargas utilizadas na avaliação. Ao final é apresentada uma conclusão dos assuntos tratados neste capítulo.

4.2 Metodologia

Para realização da pesquisa, dois estudos serão efetuados: em um primeiro momento (Seção 5.1), será avaliado o nível real de compartilhamento de memória alcançado por cada hipervisor em um ambiente com várias máquinas virtuais. Esse resultado será comparado com o potencial teórico de compartilhamento em cada situação, obtido usando as técnicas baseadas em *hash* descritas em [Barker et al., 2012].

O segundo estudo, apresentado na Seção 5.2, consiste em observar o comportamento temporal dos níveis de compartilhamento de memória nos hipervisores em várias situações de carga distintas, juntamente com o processamento demandado pelos mesmos ao longo do tempo, em cada situação.

As próximas subseções descrevem com mais detalhes o ambiente de experimentação, os cenários experimentais e as cargas de trabalho consideradas.

4.2.1 Ambiente de Testes

Todos os experimentos foram realizados em um servidor Dell R620 com as seguintes características:

- Processador: Intel Xeon E5-2609 com 8 núcleos, 2,5 GHz, cache 2,5 MB por núcleo;
- Memória: 16 GB de memória DDR3 (1,33 GHz).

Para montagem do ambiente de testes foram considerados hipervisores nativos (*bare metal*) com suporte para sistemas operacionais convidados Debian GNU/Linux, CentOS e Microsoft Windows, tipicamente usados em *datacenters* de nuvens computacionais e na consolidação de servidores. Os hipervisores necessitam dispor dos mecanismos de compartilhamento de memória implementados em suas versões comerciais.

Serão avaliados os hipervidores KVM e VMware e seus mecanismos de compartilhamento de memória KSM e TPS respectivamente. As características específicas de cada hipervisor são apresentadas a seguir:

- VMware ESXi ® 5.5.0 build-2068190
- KVM 3.10.0-229[1]

O hipervisor *Xen* não foi considerado, pois sua implementação de CBPS (chamada *Difference Engine*) é experimental e não está presente nas versões do hipervisor disponíveis ao público [Gupta et al., 2010]. Finalmente, o hipervisor *Hyper-V* da Microsoft não implementa mecanismos de compartilhamento transparente de memória [Microsoft Inc., 2015].

4.2.2 Cenários

Para as máquinas virtuais, foram escolhidos três sistemas operacionais distintos: Windows 2012, Debian GNU/Linux 7.6 e CentOS 7.1 [2], todas em 64 bits e configuração servidor. Dois deles (Debian e CentOS) são propositalmente similares, para avaliar o impacto dessa similaridade de núcleo, bibliotecas e ferramentas nos níveis de compartilhamento. As máquinas virtuais foram instanciadas inicialmente no hipervisor VMware, sendo em seguida copiadas e convertidas por meio do utilitário qemu-img sem alteração da configuração para o KVM. Assim o processo envolvendo os experimentos e a avaliação é realizado em igualdade de condições entre os hipervisores.

Os testes foram organizados em quatro cenários distintos, com as seguintes características:

1. Todos os sistemas operacionais convidados instalados com a mesma versão, para avaliar o funcionamento do compartilhamento com o mesmo *kernel*, bibliotecas e aplicações sendo executadas em convidados com mesmo sistema operacional convidado. Nesse modelo são utilizadas máquinas virtuais com o sistema convidado Debian GNU/Linux; aqui espera-se obter um alto nível de compartilhamento, devido à forte similaridade entre as VMs;

[1] A versão utilizada do QEMU foi a 1.5.3.

[2] O *Kernel* Linux e versão de `libc` no Debian são: 3.2.0 e 2.13-38, no CentOS: 3.10.0-229 e 2.17-78 respectivamente.

2. Idem ao anterior, executando o sistema operacional Windows Server 2012;

3. 50% das máquinas virtuais com Debian GNU/Linux e 50% com CentOS; espera-se que a relativa similaridade entre os sistemas permita alcançar um nível de compartilhamento significativo;

4. 50% das máquinas virtuais com Debian GNU/Linux e 50% com Windows 2012; neste caso, provavelmente a única similaridade entre os dois grupos de máquinas virtuais se dará nas cargas de trabalho que estas executarão. Será analisado como as mesmas aplicações executando em núcleos diferentes com bibliotecas diferentes podem ser compartilhadas. O objetivo é indicar de acordo com as cargas utilizadas nos ambientes distintos, como as aplicações se comportam na questão de compartilhamento de memória em ambientes diferentes.

Tendo em vista a quantidade de memória de máquina disponível (16 GB) e a memória necessária para executar cada máquina virtual sem ocorrência de paginação (*swapping*), decidiu-se lançar 8 máquinas virtuais em cada experimento, com até 2 GB de memória RAM para cada uma. Os sistemas convidados usam somente páginas de 4 KB em todos os experimentos (super-páginas foram desabilitadas). Também foram configuradas 2 CPU virtuais em cada VM.

4.2.3 Cargas de Trabalho

Para os experimentos foram considerados dois tipos de carga de trabalho, ou seja, de aplicações executadas dentro das máquinas virtuais: uma *carga sintética* basicamente uma aplicação que aloca e acessa continuamente uma grande área de memória e uma *carga real* consistindo em uma aplicação real típica de ambiente de nuvem computacional.

A carga sintética consiste em alocar uma grande área de memória (1 GB), preenchê-la com valores predefinidos e reescrever esses mesmos valores periodicamente. A área alocada deve ser alinhada com um início de página (4.096 bytes), para que as páginas da área alocada tenham o mesmo conteúdo em todas as máquinas virtuais. Essa alocação alinhada é obtida através da chamada `memalign` no Linux e `_aligned_malloc` no Windows. A carga sintética usada nos experimentos segue o algoritmo 1.

Foi escolhido o valor de carga sintética de 1 GB, pois os sistemas operacionais possuem na sua estrutura o núcleo, bibliotecas e programas que são carregados em memória além da carga que será executada, portanto não permitindo que a memória disponível seja ultrapassada e consequentemente haja a ocorrência de *swap*.

O programa usado como carga sintética foi escrito na linguagem C e seu conteúdo nos sistemas Windows e Linux pode ser visualizado no apêndice A.1.

Observa-se que todas as páginas da área alocada são idênticas, e que seu conteúdo se mantém constante, pois as escritas nessa área escrevem sempre os mesmos valores nas posições de memória (linhas 4 e 11 do Algoritmo 1). Contudo, a frequência de escritas é variável, sendo definida pelo parâmetro *write_rate* (*wr*). Assim, a carga sintética pode ser configurada para diferentes níveis de *stress* no uso da memória RAM, variando o valor de *wr*. Os seguintes casos são considerados:

- **Melhor caso**: considera $wr = 0$, ou seja, após o preenchimento inicial (linhas 3-5), a área de memória alocada não é mais acessada, pois $wr = 0 \Rightarrow num\_writes = 0$ (linha 7). Neste

Algoritmo 1 Carga de trabalho sintética.

Require: *page_size*: size of each page (4096 bytes)
Require: *write_rate*: % of pages to write in each cycle
1: *buffer_size* = 1 GB
2: *buffer* = `memalign` (*page_size*, *buffer_size*)
3: **for** $i = 1$ **to** *buffer_size* **do**
4: *buffer*[i] = i mod 256
5: **end for**
6: $num\_pages = buffer\_size \div page\_size$
7: $num\_writes = num\_pages \times write\_rate$
8: **loop**
9: **for** $i = 1$ **to** *num_writes* **do**
10: *pos* = random (1...*buffer_size*)
11: *buffer*[*pos*] = *pos* mod 256
12: **end for**
13: sleep (10s)
14: **end loop**

caso, o mecanismo de compartilhamento está totalmente livre para varredura da memória e agrupar páginas replicadas, pois as máquinas virtuais estarão ociosas após a alocação.

- **Pior caso**: com $wr = 1$, cada ciclo de escrita (linhas 9-12) reescreve *num_pages* posições aleatórias de memória na área alocada (em média uma escrita por página). As contínuas escritas na memória, embora não alterem seu conteúdo, podem inibir a atividade do mecanismo de compartilhamento.

- **Casos intermediários**: com $0 < wr < 1$, cada ciclo de escrita reescreve $wr \times num\_pages$ posições aleatórias de memória na área alocada. Após alguns testes, os valores $wr = 1/8$ e $wr = 1/32$ foram escolhidos para este estudo.

Como carga real para os experimentos, foi escolhido um servidor de banco de dados *MySQL* (versão 5.5) e um programa de *benchmark* para o mesmo denominado *MySQLslap* (versão 5.5) [MySQL, 2007]. Essa escolha se justifica por ser uma aplicação frequentemente virtualizada em ambientes de larga escala, por apresentar uma demanda significativa de memória RAM e por estar disponível nos três sistemas operacionais utilizados.

O programa *MySQLslap* cria uma base de dados onde efetua operações de criação de tabelas, consultas, inserção e remoção de dados, para testar o comportamento e o desempenho do serviço MySQL. Após alguns testes, foram adotados os seguintes parâmetros de *benchmark*:

- Nível de concorrência: 400 clientes simulados;

- Número de iterações: 50;

- Número de consultas: 5500 por cliente;

- Motor de SQL: *InnoDB*.

A definição desses parâmetros buscou obter um grande consumo de memória e processador e também evitando a ocorrência de *swapping*. Nos testes realizados, cada máquina virtual executa uma instância do cliente *MySQLslap* e do servidor *MySQL*, para minimizar o tráfego de rede e evitar interferências nos experimentos.

4.2.4 Coleta de dados

Com base nos cenários e cargas foi definido que 40 medições seriam realizadas na tabela 4.1 segue a descrição das coletas efetuadas:

Tabela 4.1: Testes realizados

Hipervisor	Cenários	Carga Sintética	Carga Real	Total
KVM	4	4	1	20
VMware	4	4	1	20

Em cada cenário são executas cinco cargas, sendo quatro sintéticas mais uma real para cada hipervisor. Cada experimento ocorrerá uma vez e serão realizadas medições adicionais para comprovar a confiabilidade dos dados obtidos.

Por se tratar de um estudo experimental, foram coletados dados reais dos *hosts* físicos e dos hipervisores, neles é possível obter as informações desejadas relativas ao uso de memória, consumo de CPU e informações dos mecanismos de compartilhamento. Para os dois hipervisores foram utilizados mecanismos distintos para obtenção desses dados que apresentam informações que serão interpretadas e utilizadas para posterior análise dos resultados obtidos.

No hipervisor VMware, os dados foram gerados por meio de ferramenta de monitamento dos recursos em tempo real `esxtop`, sendo um mecanismo do próprio hipervisor. O `esxtop`, coleta diversos indicadores de funcionamento, para o estudo foram considerados os seguintes indicadores:

- Physical Cpu % Util - Uso de CPU no Sistema
- Kernel MBytes - Memória Utilizada pelo sistema Hospedeiro
- NonKernel - Memória Utilizada pelas máquinas virtuais
- Free MBytes - Memória Disponível Total
- PShare Shared - Memória total compartilhada
- PShare Common - Memória em uso pelo sistema de compartilhamento
- PShare Savings - Memória que não precisou ser alocada (salvo)

No VMware, é possível optar em utilizar páginas de 2 MB de memória para os sistemas convidados, ao invés de páginas de 4 KB, para isso pode ser alterado o parâmetro `Mem.AllocGuestLargePage`, essa configuração influencia diretamente no compartilhamento de memória, segundo [VMware Inc, 2014], a eficiência no compartilhamento de memória com páginas de 4 KB é maior devido ao fato do mecanismo utilizado, realizar as verificações de duplicidade de página no momento que essas são alocadas e o mecanismo não é habilitado imediatamente quando utilizadas páginas de 2 MB, sendo ativado sob demanda.

O KVM foi instalado no sistema Centos 7. Por se tratar de uma distribuição Linux, a coleta dos parâmetros de CPU e memória foram feitas por meio de indicadores do próprio sistema operacional como `vmstat` e `sar`, utilizados em sistema de análise de desempenho e monitamento de servidores. Os dados sobre o desempenho do KVM, foram coletados do diretório `/sys/kernel/mm/ksm/`, da estrutura de diretórios. Nessa estrutura foram coletados dados dos seguintes arquivos dinâmicos:

- pages_volatile - Páginas que estão mudando rapidamente e que não serão compartilhadas, estão na árvore instável do KSM
- pages_unshared - Páginas únicas que não serão compartilhadas
- pages_shared - Páginas que são utilizadas pelo KSM, essas estão em uso e sendo compartilhando para as demais
- pages_sharing - Páginas compartilhadas que as VMs apontam no total.

Apesar dos hipervisores contarem com os mecanismos vistos acima que indicam o compartilhamento de memória, durante a realização dos testes esses não demonstraram ser confiáveis para serem utilizados no estudo. Portanto, para avaliação dos mecanismos será utilizado como indicador o uso de memória dos sistemas nos cenários propostos.

4.3 Conclusão

Este capítulo apresentou os principais detalhes técnicos do ambiente de testes e a metodologia utilizada para obtenção dos dados, também foi apresentada uma justificativa da escolha dos componentes do ambiente técnico e os indicadores coletados.

No próximo capítulo serão apresentados os experimentos realizados e os resultados obtidos com as análises sobre esses resultados.

Capítulo 5

Resultados e Discussões

Nesse capítulo, serão apresentados, os resultados experimentais provenientes das medições realizadas, os resultados estão divididos em duas seções: Compartilhamento potencial e real e compartilhamento ao longo do tempo. Como forma de comprovar e analisar a variação dos valores obtidos, será apresentado também um estudo estatístico com alguns dos casos analisados para verificar possíveis variações nas medições juntamente com a avaliação dos resultados obtidos. Ao final do capítulo, as conclusões e considerações referentes aos resultados.

5.1 Compartilhamento Potencial e Real

Buscou-se avaliar a eficiência dos mecanismos de compartilhamento de memória dos hipervisores em estudo, através da comparação entre os compartilhamentos realizados (medidos pelas ferramentas de monitoração dos hipervisores) e o potencial teórico de compartilhamento, obtido pela análise *offline* das imagens de memória das máquinas virtuais, de forma similar ao realizado em [Barker et al., 2012]. Nesta análise foram consideradas as condições de trabalho dos hipervisores, as cargas de trabalho sintéticas descritas em 4.2.3, e em alguns cenários (Seção 4.2.2).

A fim de determinar o compartilhamento potencial (ou compartilhamento máximo teórico), as máquinas virtuais foram suspensas após 20 minutos de execução e suas imagens de memória foram salvas em disco através de ferramentas específicas. No VMware foi utilizado o utilitário `vmdumper` para suspender a máquina e gerar o *dump* da memória e posteriormente foi executado outro utilitário o `vmss2core`, para gerar um arquivo contendo imagem linear da RAM, *byte a byte*. Da mesma forma foi feito no KVM, sendo utilizados respectivamente o `virsh` e o `lqs2mem`.

Para simplificar a análise, para cada página de RAM das imagens obtidas foi gerado um *hash* SHA1 de seu conteúdo. A comparação de *hashes* revela páginas replicadas na memória das máquinas virtuais, permitindo calcular o conjunto mínimo de páginas necessárias para atender aquelas máquinas virtuais (número de *hashes* distintos), bem como o potencial de compartilhamento de páginas (número de *hashes* replicados). As páginas cujo conteúdo é nulo (zeros) não foram consideradas, pois não são mapeadas em memória física pelos hipervisores.

Cada uma das 8 máquinas virtuais recebe até 2 GB de RAM do hipervisor, o que resulta em 524.288 páginas de 4 KB por VM. O resultado da análise de compartilhamento potencial e real é apresentado na Tabela 5.1.

Tabela 5.1: Compartilhamento potencial e real

Experimento		Potencial teórico			Uso de RAM real	
Hipervisor	**Cenário**	**uso de RAM**	**intra-VM**	**inter-VM**	**com comp.**	**sem comp.**
VMware	I: 8 × Debian	627 MB	8.225 MB	751 MB	1.158 MB	10.079 MB
VMware	II: 8 × Windows	3.296 MB	9.003 MB	2.202 MB	4.708 MB	16.384 MB
KVM	I: 8 × Debian	630 MB	8.241 MB	757 MB	965 MB	10.014 MB
KVM	II: 8 × Windows	3.258 MB	8.489 MB	2.760 MB	4.176 MB	16.384 MB

Nos valores das colunas "Potencial teórico" foram obtidos números de páginas correspondentes e comparadas entre si. Sendo calculado o proporcional das páginas de 4 KB correspondentes em MB. As colunas "uso RAM" representam a quantidade de memória RAM necessária para as máquinas virtuais, tanto no caso teórico (considerando o máximo compartilhamento possível) quanto na prática (o valor observado em cada hipervisor). Nessa tabela, constata-se que a quantidade de memória usada na prática é bem maior que o mínimo teórico; os motivos prováveis para que isso ocorra estão descritos abaixo:

- A igualdade entre algumas páginas pode ser de curta duração não sendo capturada pelos hipervisores;
- Páginas com *hash* correspondente, mas com conteúdo diferente que não puderam ser mescladas;
- A ineficiência do mecanismo implementado em cada hipervisor.

Outra constatação relevante é que a maior parte do potencial de compartilhamento provém de páginas replicadas na mesma máquina virtual (*intra-VM*), corroborando os resultados de [Barker et al., 2012]. Essas páginas poderiam ser mapeadas em uma única página de memória física da máquina virtual pelo próprio sistema operacional convidado, mas não o foram.

Isso significa que a gestão de memória RAM dos sistemas operacionais convidados não está adaptada para funcionar de forma eficiente em ambientes virtualizados. Ela opera baseada no pressuposto que a quantidade de memória RAM é constante e deve ser usada em sua totalidade para melhorar o desempenho do sistema (fazendo cache do sistema de arquivos, por exemplo). Isso não é mais válido em sistemas virtualizados, pois a memória RAM usada como cache em cada máquina virtual poderia ser melhor aproveitada pelo hipervisor.

As duas últimas colunas da Tabela 5.1 apresentam o consumo de memória RAM em cada experimento, com mecanismo de compartilhamento de memória ligado ou desligado. Nos experimentos com o cenário II, esporadicamente algumas máquinas virtuais apresentaram *swapping* quando o mecanismo de compartilhamento estava desligado, indicando que a memória de máquina fora esgotada.

5.2 Comportamento ao Longo do Tempo

Esta seção apresenta resultados de experimentos que avaliam o comportamento de cada hipervisor ao longo do tempo, em função das cargas de trabalho e dos sistemas convidados.

Os experimentos permitem observar como os hipervisores reagem à demanda de uso de memória pelas máquinas virtuais e como os mecanismos baseados em CBPS se comportam

em função da disponibilidade de processador, uma vez que são ativados quando a carga de processamento é baixa.

O primeiro ponto das curvas, em $t = 0$, corresponde ao uso de memória com as máquinas virtuais inicializadas, imediatamente antes de lançar a carga de trabalho. A linha tracejada (*Host Mem*) corresponde à memória usada pelo próprio hipervisor, que não varia de forma perceptível nos dois hipervisores durante a realização dos testes.

As Figuras 5.1 e 5.2 apresentam a evolução temporal do uso de memória de máquina (RAM) no cenário I (8× Debian), com várias cargas de trabalho, usando respectivamente os hipervisores KVM e VMWare. Em todos os experimentos, foram feitas medições a cada 5 minutos, até 60 minutos. Os valores de todas as cargas sintéticas são apresentados até 30 minutos, pois após esse período o compartilhamento permaneceu constante.

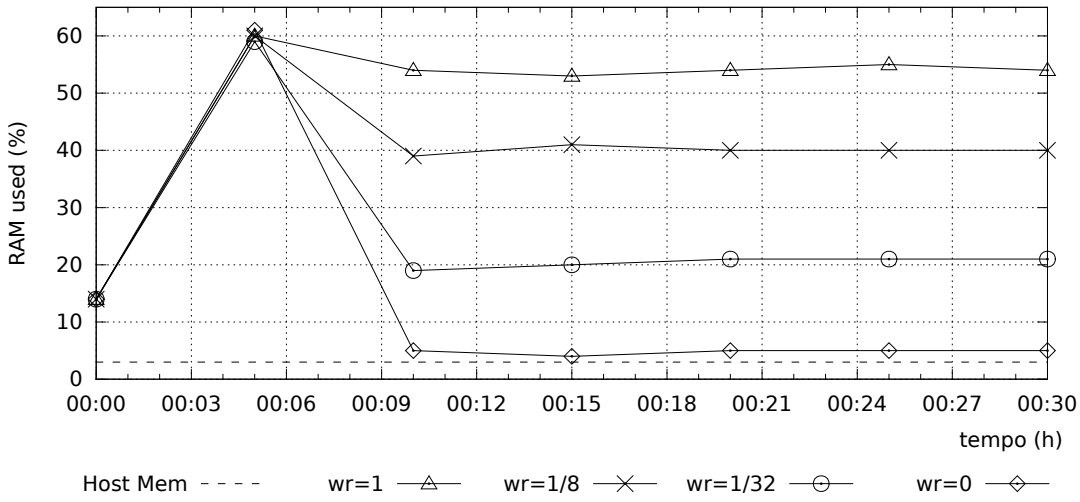

Figura 5.1: Uso de RAM no KVM com 8 VMs Debian (Cenário I).

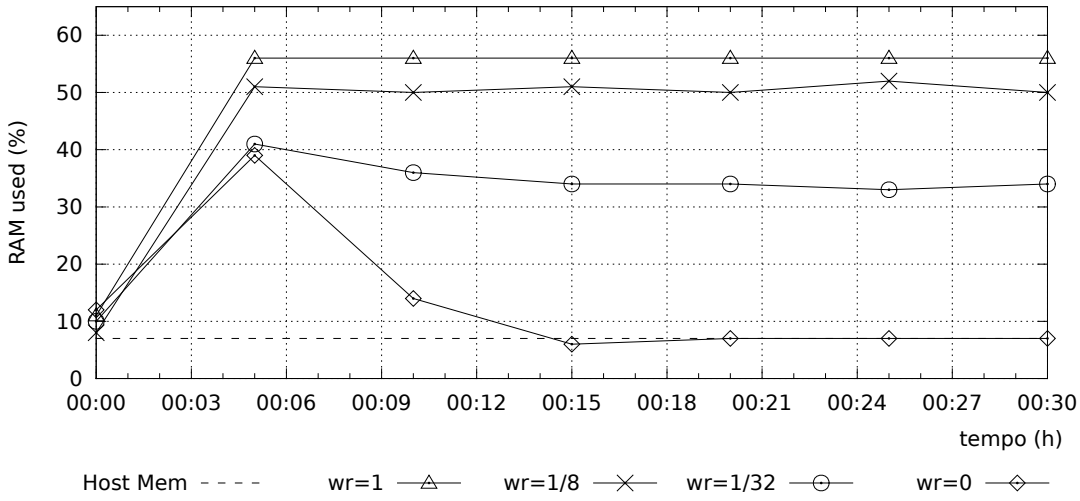

Figura 5.2: Uso de RAM no VMware com 8VMs Debian (Cenário I).

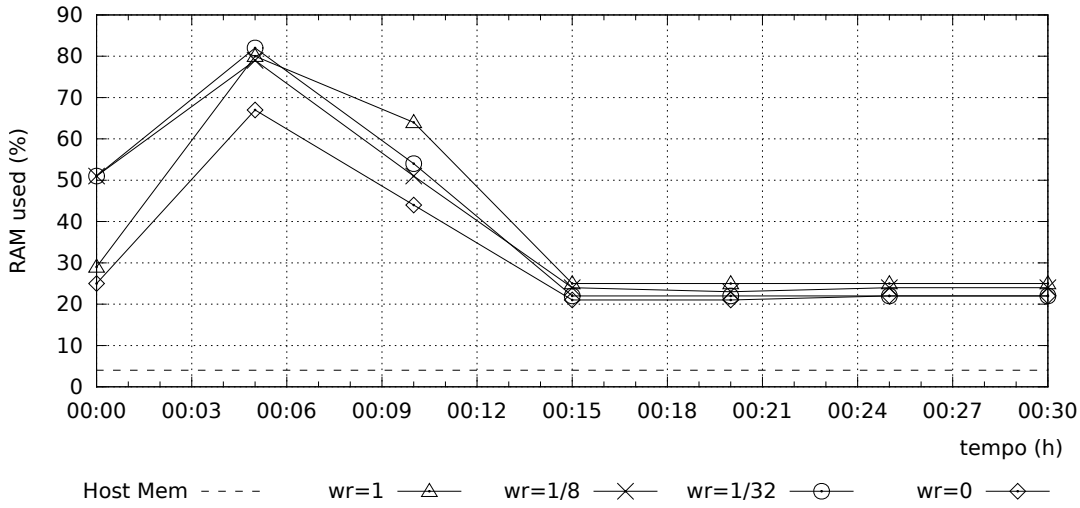

Figura 5.3: Uso de RAM no KVM com 8VMs Windows (Cenário II).

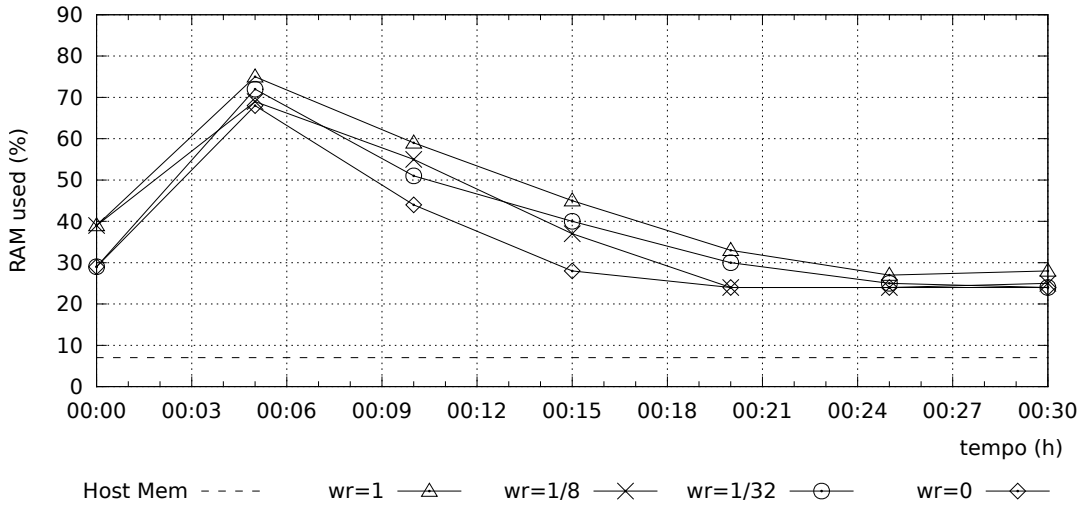

Figura 5.4: Uso de RAM no VMware com 8VMs Windows (Cenário II).

Em todos os gráficos que tratam do consumo de memória percebe-se um pico no uso do memória física no início do experimento, após o lançamento das cargas de trabalho. Esse pico é mais significativo no KVM (Figura 5.1), cujo mecanismo de compartilhamento parece levar mais tempo para iniciar sua operação[1]. Já no VMWare (Figura 5.2) esse pico é menos intenso, sobretudo para as cargas de trabalho com menor taxa de escritas ($wr \rightarrow 0$). Contudo, após o pico inicial, o hipervisor KVM reduz de forma mais significativa o uso de memória física que o VMWare, sobretudo para as taxas de escrita mais baixas. Ambos os hipervisores têm comportamento similar no pior caso ($wr = 1$).

As Figuras 5.1 a 5.4 mostram que o hipervisor KVM teve um melhor aproveitamento das possibilidades de compartilhamento de memória, sobretudo nas cargas mais leves ($wr \rightarrow$

[1]O mecanismo KSM do hipervisor KVM somente funciona quando a quantidade de memória livre cai abaixo de um limite (por default 20%); em nosso testes, esse valor foi alterado para 75%, para poder observar melhor seu comportamento.

0). Observa-se também uma clara diferença nos níveis de compartilhamento obtidos entre os cenários I (8× Debian) e II (8× Windows). Apesar do menor consumo de memória ser atingido no cenário I, este cenário também é o mais sensível à carga de trabalho (*wr*).

Nos cenários III (4× Debian, 4× Centos) e IV (4× Debian, 4× Windows) , Figuras 5.5 a 5.8, os resultados são similares aos anteriormente apresentados, ou seja, o KVM apresentou os melhores indicadores de compartilhamento. Outra questão que merece ser destacada é que por se tratar de um ambiente com sistemas convidados distintos, devido a carga adicional do sistema operacional das 4 VMs, têm-se um valor maior principalmente para o caso $wr = 0$, esse fato pode ser verificado nas Figuras 5.5 e 5.6 em comparação com as Figuras 5.1 e 5.2 onde os sistemas convidados são todos do mesmo tipo.

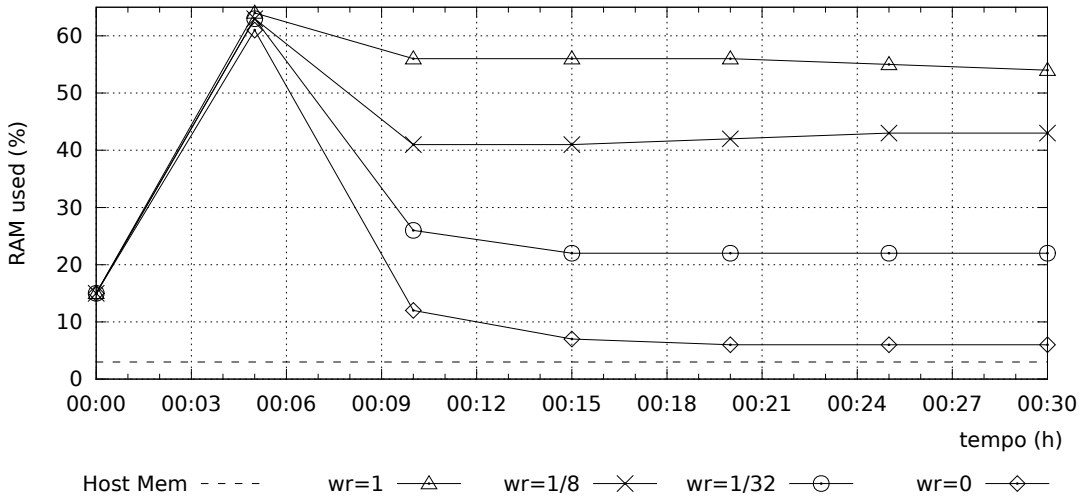

Figura 5.5: Uso de RAM no KVM com 4VMs Debian e 4VMs Centos (Cenário III).

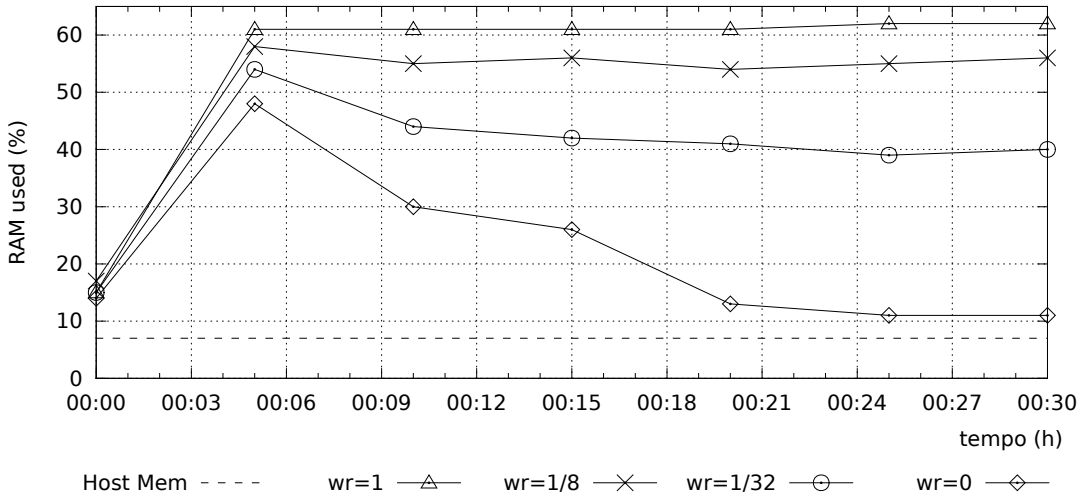

Figura 5.6: Uso de RAM no VMware com 4VMs Debian e 4 VMs Centos (Cenário III).

Essa pequena diferença na carga $wr = 0$ ocorre principalmente pelo fato dos sistemas operacionais serem de mesma família, nas Figuras 5.3 e 5.4 isso não ocorre quando comparado

ao resultado do cenário IV, Figuras 5.7 e 5.8. O primeiro motivo para isso ocorrer é por conta dos sistemas operacionais serem de famílias diferentes e mesmo executando a mesma carga, não foi possível aproveitar o compartilhamento. Outro fator que contribui é que o Windows necessita de mais memória para sua execução em comparação ao Linux, o que foi visto também no estudo teórico na tabela 5.1. Demonstrando que o compartilhamento em sistemas operacionais de diferentes famílias não é eficiente, o que foi constatado também no estudo [Barker et al., 2012].

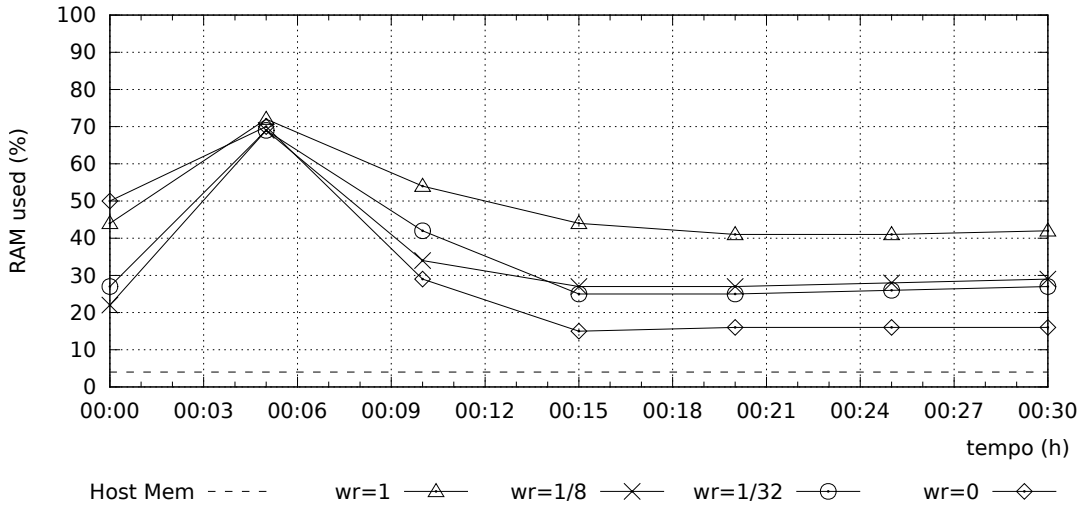

Figura 5.7: Uso de RAM no KVM com 4 VMs Debian e 4VMs Windows (Cenário IV).

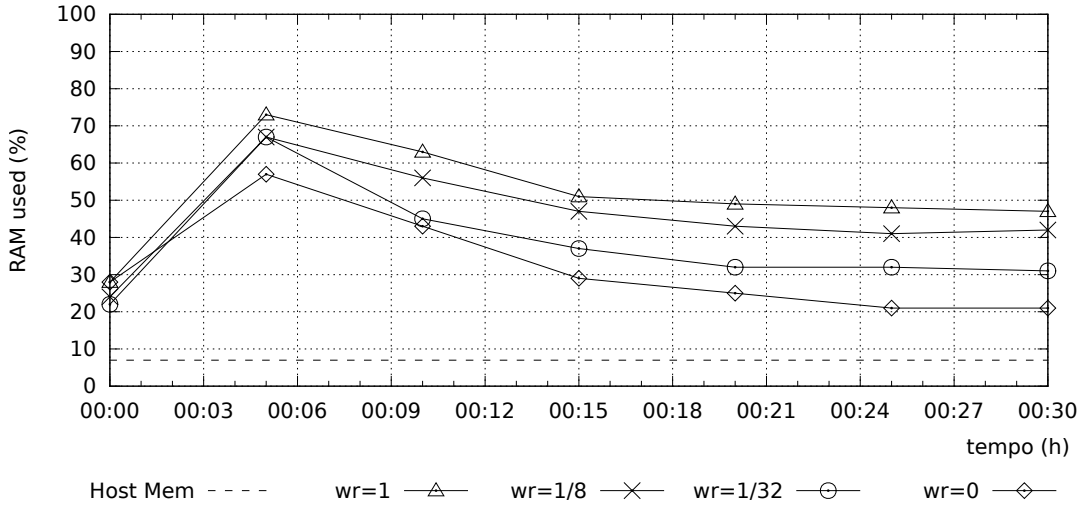

Figura 5.8: Uso de RAM no VMware com 4VMs Debian e 4VMs Windows (Cenário IV).

Para a carga $wr = 1$, percebe-se que nas Figuras 5.7 e 5.8 quando foram adicionados os sistemas com Windows, o consumo de memória baixou de forma significativa. Reforçando a maior sensibilidade dos sistemas Linux na execução dessa carga, o que fica evidenciado em todos os gráficos envolvendo sistemas Linux executando essa carga. Nota-se também que quando são executados os casos intermediários ($wr = 1/8$ e $wr = 1/32$) o consumo diminui proporcionalmente com a quantidade de páginas regravadas nos sistemas Linux, mesmo com os valores

sendo iguais. A influência de mecanismos de segurança como ASLR foi estudada para verificar se devido a sua forma de funcionamento dificultaria o processo de compartilhamento, o que foi investigado também em [Barker et al., 2012]. Para essa carga sintética os valores obtidos desabilitando o ASLR nos sistemas Linux foram bem semelhantes com ele habilitado (situação padrão), descartando essa possibilidade.

Na carga real, da mesma forma que nas cargas sintéticas, foi constatado que o KVM possuiu os melhores indicadores do uso de memória o que pode ser observado nas Figuras 5.9 e 5.10. Apesar dos sistemas que executam Windows exigirem mais memória para o sistema operacional base em comparação ao Linux, o consumo em todos os cenários ficou próximo, indicando que nos sistemas Windows foi possível o compartilhamento de páginas na carga real, principalmente nos minutos finais dos testes.

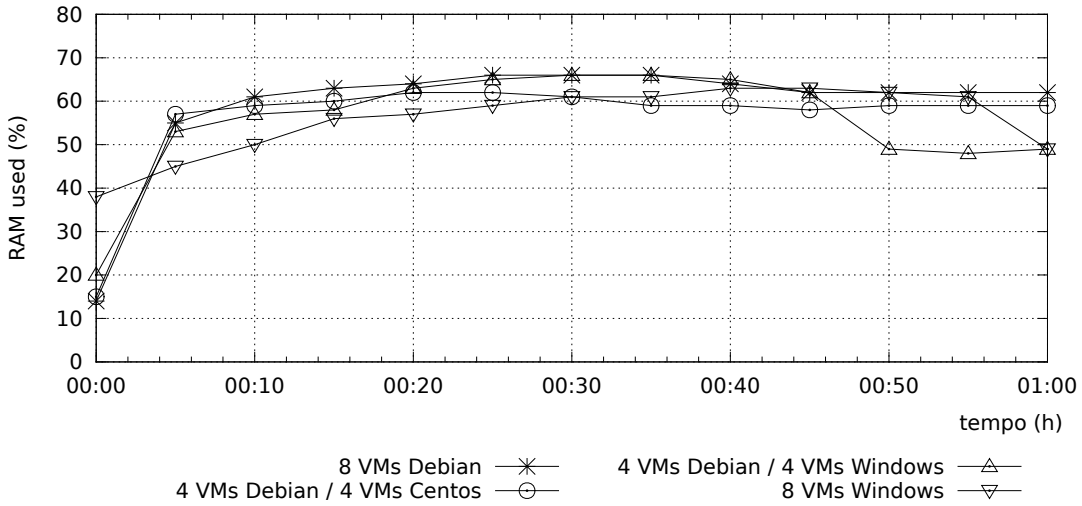

Figura 5.9: Uso de RAM no KVM com a carga real

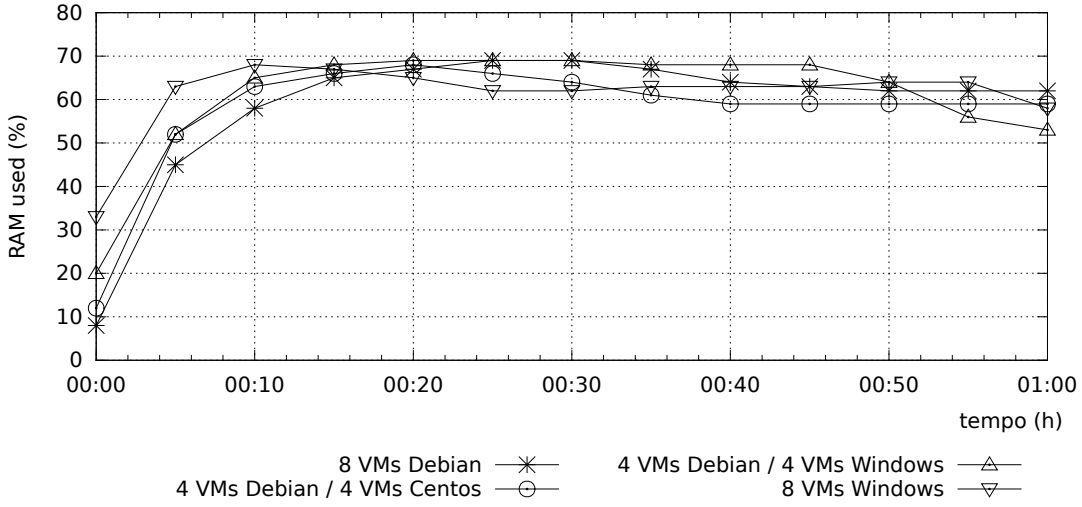

Figura 5.10: Uso de RAM no VMware com a carga real

Por ser uma carga que efetua muitas operações de entrada e saída, as oportunidades de compartilhamento são muito curtas e os dois hipervisores se mostram ineficientes no compartilhamento dessas páginas. Uma anomalia percebida nesses gráficos é com relação ao menor consumo nos testes envolvendo o cenário III em comparação ao cenário I. Apesar de representar uma pequena quantidade de ganho esse fato foi investigado e o fator que contribuiu diretamente para que isso ocorresse foram que a carga `mysqlslap` e o serviço `Mysql` em execução no Centos obteve um consumo de memória um pouco menor do que no Debian mesmo se tratando do mesmo teste.

Conforme discutido no capítulo 3, os mecanismos de compartilhamento de memória podem consumir muito processamento para encontrar páginas de memória replicadas. As Figuras 5.11, 5.12, 5.13 e 5.14 apresentam o uso de processador (no *host*) para os hipervisores estudados em todos os cenários com as cargas sintéticas ($wr = 0$, $wr = 1/32$, $wr = 1/8$ e $wr = 1$). Observa-se um uso maior de processador nos primeiros minutos do experimento, sobretudo para o KVM no cenário II (8× Windows), mas ele sempre permanece abaixo de 30%. Ou seja, haveria processamento disponível para uma política de compartilhamento mais agressiva. Mesmo assim conforme visto anteriormente, uma maior agressividade do mecanismo de compartilhamento provocaria um uso maior de CPU em virtude da maior verificação de páginas duplicadas e mesmo assim não seria garantido que o compartilhamento iria aumentar devido as deficiências dos hipervisores no compartilhamento de memória evidenciados anteriormente na tabela 5.1.

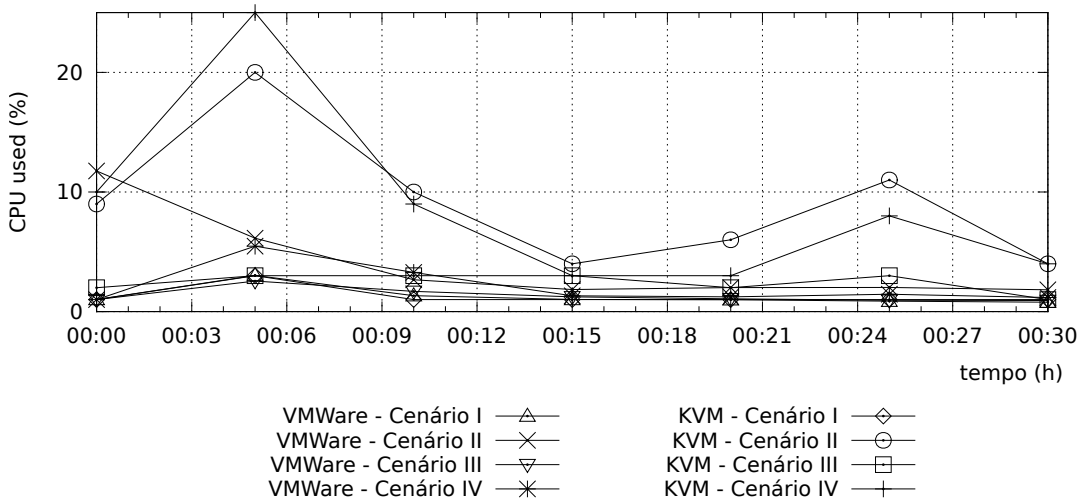

Figura 5.11: Uso de CPU no VMware e KVM com a carga $wr = 0$

Diferentemente do consumo de memória, o VMware apresentou um uso menos intenso de CPU pelos mecanismos de busca de réplicas em páginas de memória em comparação ao KSM. Isso fica bem evidente pelo pico de uso de CPU nos primeiros momentos do teste principalmente no caso dos cenários II e IV quando são utilizadas máquinas com sistema Windows, esses cenários se destacam no total. Os demais em comparação permanecem baixos.

Com relação ao uso de CPU na carga real, representado nas Figuras 5.15 e 5.16, foi percebido que a carga real provocou um grande uso de CPU em comparação a carga sintética, principalmente no hipervisor KVM, onde o uso de processamento em alguns casos quase chegou ao máximo. Analisando a queda ocorrida no consumo de processamento no decorrer dos

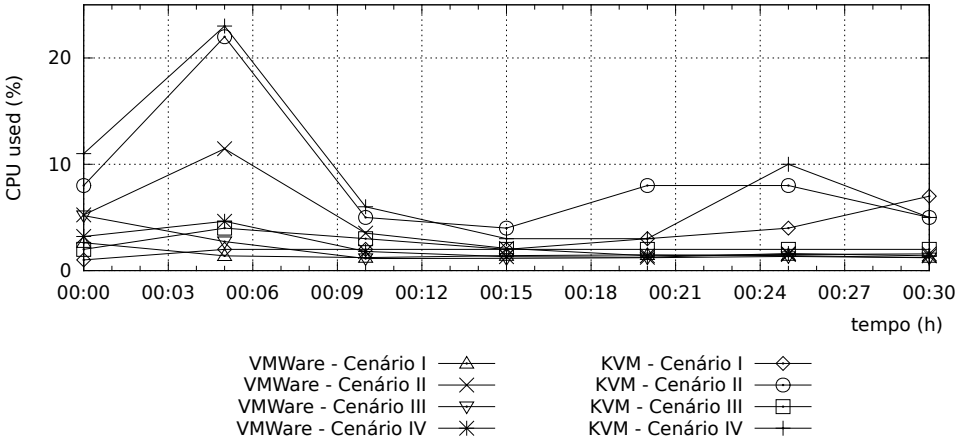

Figura 5.12: Uso de CPU no VMware e KVM com a carga $wr = 1/32$

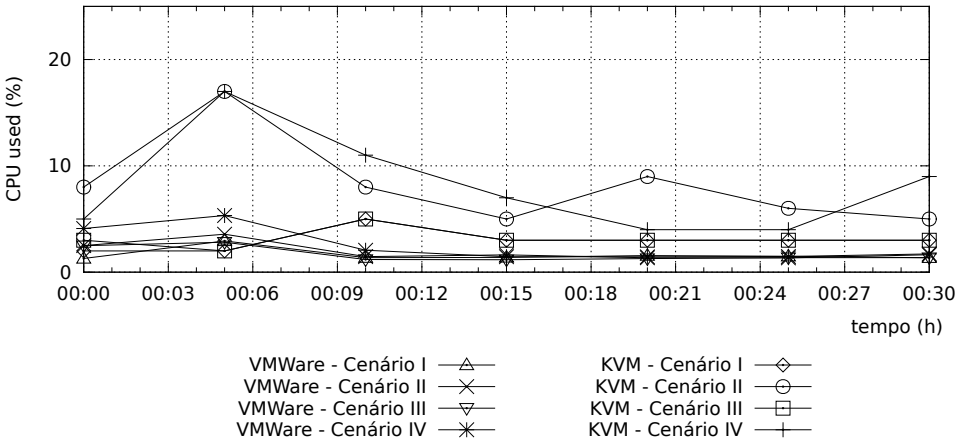

Figura 5.13: Uso de CPU no VMware e KVM com a carga $wr = 1/8$

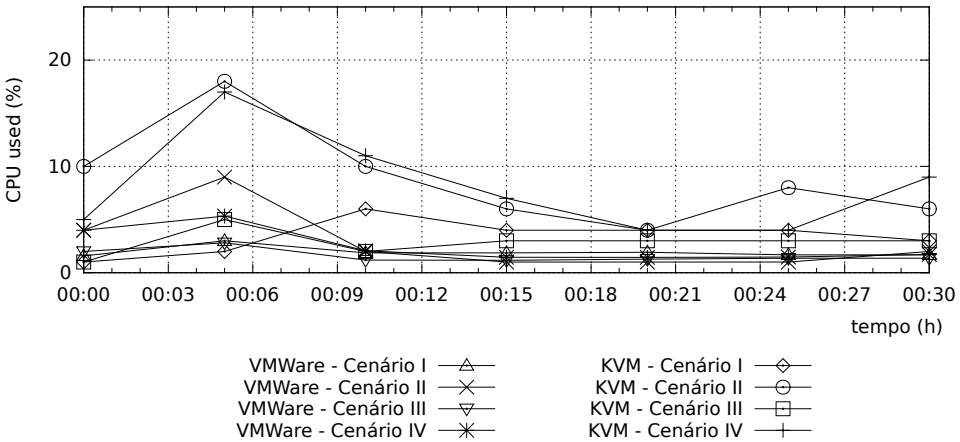

Figura 5.14: Uso de CPU no VMware e KVM com a carga $wr = 1$

testes foi percebido que ao final da carga o processamento baixou consideravelmente, portanto é evidenciado que quando o uso de processador diminui até abaixo de 5% é o momento em que a carga real é finalizada. Os sistemas após execução da carga poderiam potencializar o compartilhamento, o que não ocorreu, pois observando as figuras 5.9 e 5.10 percebe-se que o consumo de memória não baixou, ou baixou muito pouco.

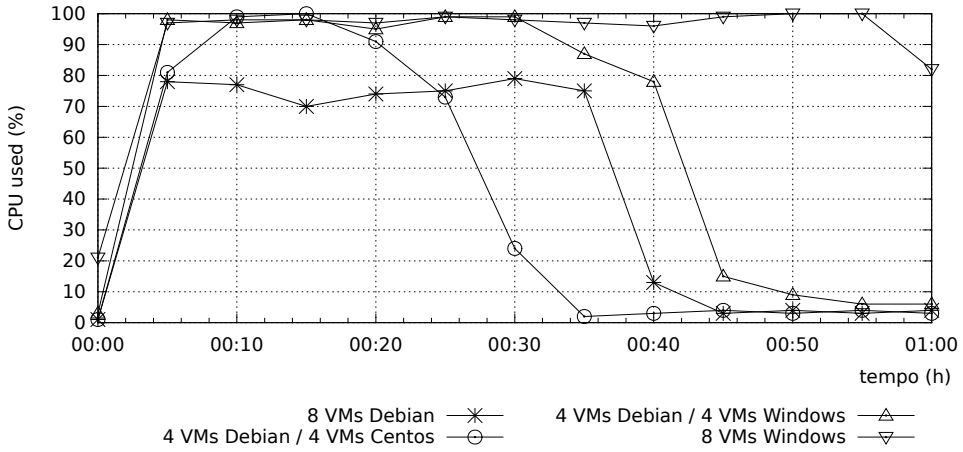

Figura 5.15: Uso de CPU no KVM com a carga real

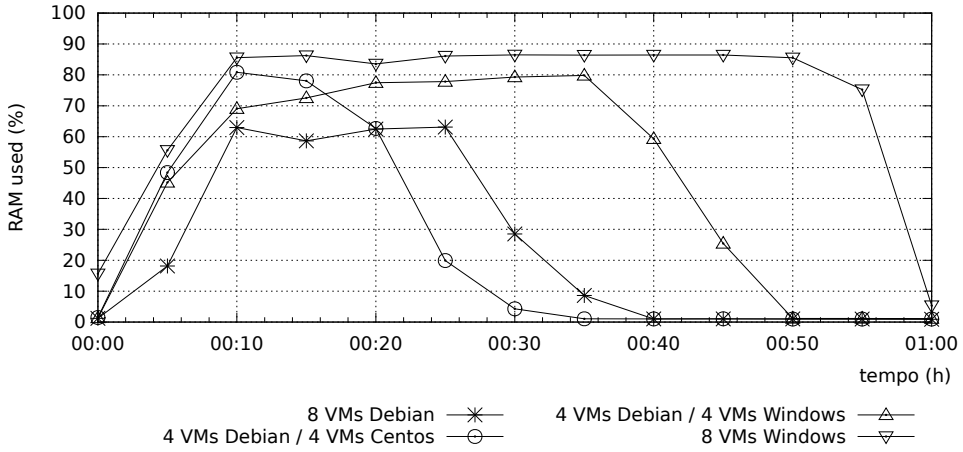

Figura 5.16: Uso de CPU no VMware com a carga real

O consumo de CPU diminui no decorrer dos testes e conforme a evolução da carga. Isso ocorre em alguns casos quando uma ou mais VMs finalizaram a carga enquanto outras ainda estavam executando a carga real. Vale ressaltar que nos cenários II e IV os testes demoraram mais para serem finalizados, ainda no cenário II no VMware, os testes finalizaram apenas entre os minutos 55 e 60 (final do teste) e no KVM eles foram finalizados após o minuto 60. Outro fator observado foi que no cenário III, os testes com carga real foram finalizados antes dos demais nos dois hipervisores, seguido do cenário I, portanto os sistemas Linux demonstraram uma performance maior na execução do teste.

No caso das máquinas executando Windows percebe-se que o KVM conseguiu aproveitar uma oportunidade de compartilhamento de forma mais rápida que o VMware, pois após

a execução das cargas o mecanismo de compartilhamento conseguiu “salvar” algumas páginas de memória de forma mais rápida, o que não ocorreu no VMware após a execução das cargas e antes da finalização dos testes para o cenário IV.

5.3 Conclusão e Avaliação dos Resultados

Ao final dos testes e após análise dos dados provenientes dos experimentos foi verificado que os principais fatores que impactam no compartilhamento de memória podem ser listados abaixo como sendo:

1. Família de sistema operacional;
2. Aplicação em execução;
3. Versão do sistema operacional;

Na figura 5.17 é possível comprovar essa constatação juntamente com os dados de todos os experimentos realizados. Com isso, têm-se que mesmo que os sistemas operacionais de diferentes famílias executem a mesma carga de trabalho, o compartilhamento não será tão efetivo. Esse resultado é bem semelhante ao exposto no estudo [Barker et al., 2012], com a diferença que neste estudo teórico foi analisada a questão da arquitetura (x86 ou x64) do sistema operacional convidado.

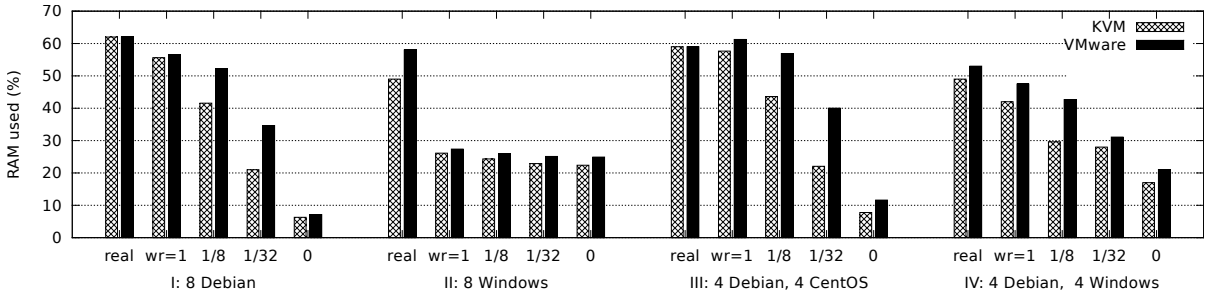

Figura 5.17: Uso de RAM geral

Com relação ao uso de memória na maioria dos casos pesquisados o KVM foi superior ao VMware no compartilhamento e teve um consumo de memória inferior, já com relação ao uso de CPU o VMware apresentou um uso mais conservador e constante, enquanto o KVM chegou a um uso maior e de forma mais rápida, provocando um pico de uso nos primeiros minutos.

Na seção 5.1 foi estudado o compartilhamento máximo teórico e concluído que a maior parte do potencial de compartilhamento é do tipo *intra-VM*, com isso conclui-se que os sistemas operacionais poderiam mapear os dados da carga sintética em uma única página de memória física da máquina virtual.

Outra fator que contribui para não haver um melhor aproveitamento do compartilhamento é que algumas oportunidades de correspondência são de curta duração, como evidenciado em [Rachamalla et al., 2013] e [Gröninger, 2011] os hipervisores não conseguem explorar essas oportunidades.

Para comprovar a confiabilidade dos dados resultantes dos experimentos foi realizado um estudo com 5 experimentos sobre a alternância dos dados e desvio padrão na carga $wr = 0$ nos hipervisores avaliados.

Tabela 5.2: Média e desvio padrão da carga $wr = 0$

Coleta VMware - $wr = 0$							**Coleta KVM -** $wr = 0$						
1	2	3	4	5	**Média**	**Desvio**	1	2	3	4	5	**Média**	**Desvio**
1163	1152	1163	1169	1161	**1161,6**	**5,49**	965	1005	981	1004	995	**990**	**12,02**

Os resultados da tabela 5.2 evidenciam uma pequena mudança nos valores dos experimentos, comprovando a confiabilidade dos dados coletados.

No próximo capítulo são apresentadas as conclusões finais, as contribuições da pesquisa e as sugestões de trabalhos futuros.

Capítulo 6

Conclusão

Conclui-se que o material obtido fundamenta a tecnologia de virtualização e os demais conceitos necessários para compreensão dos temas abordados que fundamentam a pesquisa. Também são abordados os mecanismos de compartilhamento e gerenciamento de memória em ambientes virtualizados. Foram investigados diversos mecanismos baseados em CBPS para tratar especificamente da questão do compartilhamento de memória nos hipervisores.

Além das abordagens existentes foram pesquisados os principais trabalhos na área que aplicam essas técnicas e estudam o potencial de compartilhamento. Ao mesmo tempo é demonstrado que a tecnologia de virtualização possui grande importância para os ambientes computacionais e serve como base para o paradigma de computação em nuvem, uma das tendências da área de computação que já vem sendo explorada.

[Gröninger, 2011] questiona o porquê do ambiente de computação desperdiçar sua memória com as mesmas informações em diferentes sistemas, sendo um desafio cada vez mais importante evitar a duplicação de informações.

Espera-se que esse trabalho contribua com o novo paradigma de computação em nuvem, onde os algoritmos de colocação de máquinas virtuais, possam aproveitar o recurso do compartilhamento de memória para estabelecer em qual *host* físico determinado sistema operacional poderia residir aproveitando melhor o compartilhamento de memória que aquele hipervisor específico pode oferecer. Levando em consideração os outros sistemas operacionais convidados contidos nesse mesmo *host*.

A pesquisa tem seus objetivos inicialmente propostos baseados em um estudo experimental para avaliar a efetividade dos mecanismos de compartilhamento de memória disponíveis nos hipervisores atuais, onde foram estudados por meio de experimentos e medições dos valores referentes ao compartilhamento de memória e seus indicadores como forma de validação e comprovação do estudo. Para isso foram aplicados 3 cenários diferentes de testes envolvendo hipervisores comerciais em sistemas convidados Windows e Linux executando cargas sintéticas e reais para avaliar o compartilhamento de memória *inter-VM* e *intra-VM* com base no consumo de memória.

Observa-se que o compartilhamento de memória é fortemente influenciado pelos sistemas operacionais e pelas aplicações em execução nas máquinas virtuais. Nos cenários analisados, o KVM apresentou melhores níveis de compartilhamento de memória para as cargas sintéticas; por outro lado, o VMware apresentou um uso menos intenso de CPU pelos mecanismos de busca de réplicas em páginas de memória.

Nas cargas reais os resultados relativos ao consumo de memória foram bem semelhantes nos dois hipervisores. Com relação ao consumo de processamento o VMware novamente apresentou um consumo menos agressivo de CPU. O *benchmark* `mysqlslap` apresenta um uso intenso de processamento o que dificulta o processo do compartilhamento, porém, mesmo após a realização do carga os dados relativos ao compartilhamento pouco mudaram com o processador disponível.

No estudo teórico (seção 5.1) foram apontadas deficiências nos sistemas operacionais e nos hipervisores em comparação aos dados obtidos na prática, conforme tabela 5.1. Com base nessa mesma tabela foi constatado que grande parte do compartilhamento de memória é do tipo *intra-VM*. Os sistemas operacionais em execução nos ambientes virtualizados poderiam gerenciar de forma diferenciada as páginas de memória, aproveitando melhor a memória física da máquina virtual, gerando oportunidades de compartilhamento para os hipervisores.

As oportunidades de compartilhamento de curta duração não são aproveitadas pelos hipervisores, esse fato fora verificado pelos estudos [Rachamalla et al., 2013] e [Gröninger, 2011], que tratou diretamente do hipervisor KVM. Em nosso estudo foi evidenciado por meio da carga real 5.9 e 5.10 que o hipervisor VMware também não consegue aproveitar essas oportunidades.

A conclusão desse estuda determina os principais fatores que implicam no compartilhamento de memória analisados na seção 5.2, foram a família do sistema operacional; o programa em execução e a versão do sistema operacional de forma semelhante ao encontrados em [Barker et al., 2012].

Como trabalho futuro, poderia ser encaminhada uma avaliação dos hipervisores usando sistemas operacionais específicos para ambientes de nuvem computacional, como o OSv [Kivity et al., 2014], que possuem um *footprint* de memória menor que os sistemas convencionais. Podem ser também incluídos outros hipervisores como *containers* de virtualização, juntamente com outras cargas de trabalho reais como servidor *web* Apache e aplicações Java utilizadas no ambiente de computação em nuvem.

Referências Bibliográficas

[Amit et al., 2014] Amit, N., Tsafrir, D., and Schuster, A. (2014). Vswapper: A memory swapper for virtualized environments. In *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 349–366, New York, NY, USA. ACM.

[Arcangeli et al., 2009] Arcangeli, A., Eidus, I., and Wright, C. (2009). Increasing memory density by using KSM. In *Ottawa Linux Symposium*, pages 19–28.

[Barham et al., 2003] Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I., and Warfield, A. (2003). Xen and the art of virtualization. In *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*, SOSP '03, pages 164–177, New York, NY, USA. ACM.

[Barker et al., 2012] Barker, S., Wood, T., Shenoy, P., and Sitaraman, R. (2012). An empirical study of memory sharing in virtual machines. In *USENIX Annual Technical Conference*, pages 25–25, Berkeley CA, USA.

[Ben-Yehuda et al., 2010] Ben-Yehuda, M., Day, M. D., Dubitzky, Z., Factor, M., Har'El, N., Gordon, A., Liguori, A., Wasserman, O., and Yassour, B.-A. (2010). The turtles project: Design and implementation of nested virtualization. In *9th USENIX Symposium on Operating Systems Design and Implementation (OSDI 10)*.

[Bugnion et al., 1997] Bugnion, E., Devine, S., Govil, K., and Rosenblum, M. (1997). Disco: Running commodity operating systems on scalable multiprocessors. *ACM Transactions on Computer Systems*, 15(4):412–447.

[Bugnion et al., 2012] Bugnion, E., Devine, S., Rosenblum, M., Sugerman, J., and Wang, E. Y. (2012). Bringing virtualization to the x86 architecture with the original vmware workstation. *ACM Trans. Comput. Syst.*, 30(4):12:1–12:51.

[Chang et al., 2011] Chang, C.-R., Wu, J.-J., and Liu, P. (2011). An empirical study on memory sharing of virtual machines for server consolidation. In *IEEE Intl Symposium on Parallel and Distributed Processing with Applications*, pages 244–249.

[Chen et al., 2014] Chen, L., Wei, Z., Cui, Z., Chen, M., Pan, H., and Bao, Y. (2014). Classification-based memory deduplication through page access characteristics. In *10th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, pages 65–76, New York, NY, USA. ACM.

[Chen and Noble, 2001] Chen, P. M. and Noble, B. D. (2001). When virtual is better than real. In *Proceedings of the Eighth Workshop on Hot Topics in Operating Systems*, HOTOS '01, pages 133–137, Washington, DC, USA. IEEE Computer Society.

[Chiueh and Brook, 2005] Chiueh, S. N. T.-c. and Brook, S. (2005). A survey on virtualization technologies. (Vm):1–42.

[Clark et al., 2005] Clark, C., Fraser, K., Hand, S., Hansen, J. G., Jul, E., Limpach, C., Pratt, I., and Warfield, A. (2005). Live migration of virtual machines. In *Proceedings of the 2Nd Conference on Symposium on Networked Systems Design & Implementation - Volume 2*, NSDI'05, pages 273–286, Berkeley, CA, USA. USENIX Association.

[Cormen et al., 2001] Cormen, T. H., Stein, C., Rivest, R. L., and Leiserson, C. E. (2001). *Introduction to Algorithms*. McGraw-Hill Higher Education, 2nd edition.

[Creasy, 1981] Creasy, R. J. (1981). The origin of the vm/370 time-sharing system. *IBM J. Res. Dev.*, 25(5):483–490.

[Fábrega et al., 1995] Fábrega, F. J. T., Javier, F., and Guttman, J. D. (1995). Copy on write.

[Garfinkel and Warfield, 2007] Garfinkel, T. and Warfield, A. (2007). What virtualization can do for security. *The USENIX Magazine*, 32(6):28–34.

[Gröninger, 2011] Gröninger, T. (2011). Analyzing shared memory opportunities in different workloads. Master's thesis, Karlsruher Institut für Technologie.

[Gupta et al., 2010] Gupta, D., Lee, S., Vrable, M., Savage, S., Snoeren, A. C., Varghese, G., Voelker, G. M., and Vahdat, A. (2010). Difference engine: Harnessing memory redundancy in virtual machines. *Communications of the ACM*, 53(10):85–93.

[IBM Inc, 2012] IBM Inc (2012). Nested virtualization for the next-generation cloud . `http://www.ibm.com/developerworks/cloud/library/cl-nestedvirtualization/`. Acessado em: 10-09-2014.

[Kim et al., 2009] Kim, I., Kim, T., and Eom, Y. I. (2009). Design and implementation of page sharing scheme between guests in virtualization environments. In *International Conference on Hybrid Information Technology*, pages 244–247, New York, NY, USA. ACM.

[King et al., 2003] King, S. T., Dunlap, G. W., and Chen, P. M. (2003). Operating system support for virtual machines. In *Proceedings of the Annual Conference on USENIX Annual Technical Conference*, ATEC '03, pages 6–6, Berkeley, CA, USA.

[Kivity et al., 2014] Kivity, A., Laor, D., Costa, G., Enberg, P., Har'El, N., Marti, D., and Zolotarov, V. (2014). OSv — optimizing the operating system for virtual machines. In *USENIX Annual Technical Conference*, pages 61–72.

[Laureano et al., 2007] Laureano, M., Maziero, C., and Jamhour, E. (2007). Protecting host-based intrusion detectors through virtual machines. *Comput. Netw.*, 51(5):1275 –1283.

[Lguest, 2007] Lguest (2007). The Simple x86 Hypervisor. `http://lguest.ozlabs.org/`. Acessado em: 11-25-2013.

[Madvise, 2015] Madvise (2015). Madvise System Call. https://www.kernel.org/doc/man-pages/online/pages/man2/madvise.2.html. Acessado em: 09-09-2015.

[Microsoft Inc., 2015] Microsoft Inc. (2015). Competitive Advantages of Windows Server Hyper-V over VMware-vSphere. http://download.microsoft.com/download/E/8/E/E8ECBD78-F07A-4A6F-9401-AA1760ED6985/Competitive-Advantages-of-Windows-Server-Hyper-V-over-VMware-vSphere.pdf. Accessed: 06-25-2015.

[Miłós et al., 2009] Miłós, G., Murray, D. G., Hand, S., and Fetterman, M. A. (2009). Satori: Enlightened page sharing. In *Proceedings of the 2009 Conference on USENIX Annual Technical Conference*, USENIX'09, pages 1–1, Berkeley, CA, USA. USENIX Association.

[Muchalski, 2014] Muchalski, F. (2014). Alocação de máquinas virtuais em ambientes de computação em nuvem considerando o compartilhamento de memória. Master's thesis, Programa de Pós-Graduação em Computação Aplicada – UTFPR, Curitiba PR.

[MySQL, 2007] MySQL (2007). MySQLslap – Load Emulation Client. http://dev.mysql.com. Acessado em: 02-12-2014.

[Popek and Goldberg, 1974] Popek, G. J. and Goldberg, R. P. (1974). Formal requirements for virtualizable third generation architectures. *Commun. ACM*, 17(7):412–421.

[Rachamalla et al., 2013] Rachamalla, S., Mishra, D., and Kulkarni, P. (2013). Share-o-meter: An empirical analysis of KSM based memory sharing in virtualized systems. In *Intl Conference on High Performance Computing*, pages 59–68.

[Rosenblum et al., 1995] Rosenblum, M., Herrod, S. A., Witchel, E., and Gupta, A. (1995). Complete computer system simulation: The simos approach. *IEEE Parallel Distrib. Technol.*, 3(4):34–43.

[Rosenblum and Varadarajan, 1994] Rosenblum, M. and Varadarajan, M. (1994). Simos: A fast operating system simulation environment. Technical report, Stanford, CA, USA.

[Sindelar et al., 2011] Sindelar, M., Sitaraman, R. K., and Shenoy, P. (2011). Sharing-aware algorithms for virtual machine colocation. In *ACM Symposium on Parallelism in Algorithms and Architectures*, pages 367–378.

[Vahalia, 1996] Vahalia, U. (1996). *UNIX Internals – The New Frontiers*. Prentice-Hall.

[VMware Inc., 2010] VMware Inc. (2010). Understanding memory resource management in VMware ESX Server. Technical Report EN-000411-00, VMWare Inc., Palo Alto CA, USA.

[VMware Inc, 2014] VMware Inc (2014). VMware Knowledge Base. http://kb.vmware.com/kb/1021095. Acessado em: 11-20-2014.

[Vrable et al., 2005] Vrable, M., Ma, J., Chen, J., Moore, D., Vandekieft, E., Snoeren, A. C., Voelker, G. M., and Savage, S. (2005). Scalability, fidelity, and containment in the Potemkin virtual honeyfarm. *SIGOPS Operating Systems Review*, 39(5):148–162.

[Waldspurger, 2002] Waldspurger, C. (2002). Memory resource management in VMware ESX Server. In *5th Symposium on Operating Systems Design and Implementation*, pages 181–194, New York NY, USA. ACM.

[Wen et al., 2012] Wen, Y., Zhao, J., Zhao, G., Chen, H., and Wang, D. (2012). A survey of virtualization technologies focusing on untrusted code execution. In *Proceedings of the 2012 Sixth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing*, IMIS '12, pages 378–383, Washington, DC, USA. IEEE Computer Society.

[Wood et al., 2009] Wood, T., Tarasuk-Levin, G., Shenoy, P., Desnoyers, P., Cecchet, E., and Corner, M. D. (2009). Memory buddies: Exploiting page sharing for smart colocation in virtualized data centers. In *ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, pages 31–40, New York, NY, USA. ACM.

[Xavier et al., 2013] Xavier, M., Neves, M., Rossi, F., Ferreto, T., Lange, T., and De Rose, C. (2013). Performance evaluation of container-based virtualization for high performance computing environments. In *Parallel, Distributed and Network-Based Processing (PDP), 2013 21st Euromicro International Conference on*, pages 233–240.

[Yun et al., 2014] Yun, J., Lee, C., and Yoo, C. (2014). Providing extra memory for virtual machines by sharing compressed swap pages. In *Consumer Electronics (ICCE), 2014 IEEE International Conference on*, pages 430–431.

[zRAM, 2014] zRAM (2014). Compressed RAM based block devices. `https://www.kernel.org/doc/Documentation/blockdev/zram.txt`. Acessado em: 5-19-2014.

[Zswap, 2013] Zswap (2013). A lightweight compressed cache for swap pages. `https://www.kernel.org/doc/Documentation/vm/zswap.txt`. Accessed: 5-19-2014.

Apêndice A

Cargas Sintéticas

São mostrados os códigos dos programas utilizados para as cargas sintéticas nos sistemas Windows e Linux.

A.1 Programa no Windows

A.1.1 Melhor Caso

```
#include <malloc.h>
#include <stdio.h>
#include <Windows.h>
// Carga Sintética Windows / wr = 0
int main()
{
        void *ptr;
        size_t alignment;
        char *buffer;
        int i, size;
        size=1024*1024*1024; //1024 MB
        alignment = 4096;
        // aloca size bytes de memória alinhada
        ptr = _aligned_malloc(size, alignment);
        // converte variável ptr para tipo char para preenchimento dos dados
        buffer = (char *)ptr;
        // preenche área alocada com valores entre 0 e 255
        for (i=0; i<size; i++)
        {
                buffer[i] = i % 256;
        }
        // pausa por 1 hora para as medições
        Sleep(3600000);
} // finaliza a execução
```

A.1.2 Casos Intermediários e Pior Caso

```
#include <malloc.h>
#include <stdio.h>
#include <Windows.h>
// Define número de páginas que serão gravadas novamente
// Carga Sintética Windows / wr = [1,8,32]
#define NUMPGWR 1 // Alterar valor para casos intermediários
int main()
{
        void     *ptr;
        size_t   alignment;
        char     *buffer;
        int i, size, numpages,numwrites,pos;
        size=1024*1024*1024; //1024 MB
        alignment = 4096;
        numpages = size / alignment ;
        numwrites = numpages / NUMPGWR ;
        // aloca size bytes de memória alinhada
        ptr = _aligned_malloc( size, alignment );
        // converte variável ptr para tipo char para preenchimento dos dados
        buffer = (char *)ptr;
        // preenche área alocada com valores entre 0 e 255
        for (pos=0; pos<size; pos++)
        {
                buffer[pos] = pos % 256 ;
        }
        // preenche novamente a área alocada com os mesmo valores
        while (1)
        {
                for (i=0; i<numwrites; i++)
                {
                        pos=rand( size );
                        buffer[pos] = pos % 256 ;
                }
                // aguarda 10 segundos para preencher novamente
                Sleep (10000) ;
        }
}
```

A.2 Programa no Linux

A.2.1 Melhor Caso

```
#include <malloc.h>
#include <time.h>
#include <unistd.h>
//Carga Sintética Linux / wr = 0
int main ()
```

```
{
        long int i, size=1024*1024*1024; // 1024 MB
        char *buffer ;
        // aloca size bytes de memória alinhada com o tamanho da página
        buffer = memalign (getpagesize(), size) ;
        // preenche a área alocada com valores entre 0 e 255
        for (i=0; i<size; i++)
        {
                buffer[i] = i % 256 ;
        }
        // pausa por 1 hora para as medições
        sleep (3600);
} // finaliza a execução
```

A.2.2 Casos Intermediários e Pior Caso

```
#include <malloc.h>
#include <time.h>
#include <unistd.h>
//Carga Sintética Linux / wr = [1,8,32]
//Define número de páginas que serão gravadas novamente
#define NUMPGWR 1
int main ()
{
        long int i, size, pos, numpages, numwrites;
        size=1024*1024*1024 ; // 1024 MB
        char *buffer ;
        numpages=size/getpagesize();
        numwrites=numpages / NUMPGWR; // 1 escrita por página em todas as páginas
        // aloca size bytes de memória alinhada com o tamanho da página
        buffer = memalign (getpagesize(), size) ;
        // preenche a área alocada com valores entre 0 e 255
        for (pos=0; pos<size; pos++)
        {
                buffer[pos] = pos % 256 ;
        }
        // preenche novamente a área alocada com os mesmos valores
        while(1)
        {
                for (i=0; i<numwrites; i++)
                {
                        pos = random() % size ;
                        buffer[pos] = pos % 256 ;
                }
        // aguarda 10 segundos para preencher novamente
          sleep (10);
        }
}
```