COPPE/UFRJ

AVALIAÇÃO EXPERIMENTAL DE TÉCNICAS DE VIRTUALIZAÇÃO ATRAVÉS DE BALANCEAMENTO DE CARGA EM CLUSTERS DE COMPUTADORES

Almir Dominicini Fernandes

Dissertação de Mestrado apresentada ao Programa de Pós-graduação em Engenharia de Sistemas e Computação, COPPE, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Mestre em Engenharia de Sistemas e Computação.

Orientador: Claudio Luis de Amorim

Rio de Janeiro
Setembro de 2010

AVALIAÇÃO EXPERIMENTAL DE TÉCNICAS DE VIRTUALIZAÇÃO ATRAVÉS DE BALANCEAMENTO DE CARGA EM CLUSTERS DE COMPUTADORES

Almir Dominicini Fernandes

DISSERTAÇÃO SUBMETIDA AO CORPO DOCENTE DO INSTITUTO ALBERTO LUIZ COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE ENGENHARIA (COPPE) DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Examinada por:

Prof. Claudio Luis de Amorim, Ph.D.

Prof. Felipe Maia Galvão França, Ph.D.

Prof. Alexandre Sztajnberg, D.Sc.

RIO DE JANEIRO, RJ – BRASIL
SETEMBRO DE 2010

Fernandes, Almir Dominicini

Avaliação Experimental de Técnicas de Virtualização através de Balanceamento de Carga em Clusters de Computadores/Almir Dominicini Fernandes. – Rio de Janeiro: UFRJ/COPPE, 2010.

XII, 86 p.: il.; 29, 7cm.

Orientador: Claudio Luis de Amorim

Dissertação (mestrado) – UFRJ/COPPE/Programa de Engenharia de Sistemas e Computação, 2010.

Referências Bibliográficas: p. 62 – 67.

1. Virtualization. 2. Load Balance. 3. Migration. I. Amorim, Claudio Luis de. II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia de Sistemas e Computação. III. Título.

Resumo da Dissertação apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.)

AVALIAÇÃO EXPERIMENTAL DE TÉCNICAS DE VIRTUALIZAÇÃO ATRAVÉS DE BALANCEAMENTO DE CARGA EM CLUSTERS DE COMPUTADORES

Almir Dominicini Fernandes

Setembro/2010

Orientador: Claudio Luis de Amorim

Programa: Engenharia de Sistemas e Computação

Apresenta-se, nesta dissertação, a comparação de duas técnicas de virtualização de sistemas. Neste trabalho, é avaliada a eficiência destas duas técnicas de virtualização. Essa avaliação ocorre em um *cluster* de máquinas físicas, onde cada máquina física suporta diversas máquinas virtuais, que por sua vez, podem migrar entre as máquinas físicas, sempre que esta migração for necessária para balancear a carga do sistema. Avalia-se também a eficiência das técnicas em migrar as suas Máquinas virtuais. Para montar o ambiente necessário a esta avaliação, foi preciso desenvolver um gerenciador de máquinas virtuais, que é o responsável por balancear a carga do sistema. A apresentação deste gerenciador também está no escopo da dissertação.

Abstract of Dissertation presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

EXPERIMENTAL EVALUATION OF VIRTUALIZATION TECHNIQUES BY LOAD BALANCING IN A COMPURTER CLUSTER

Almir Dominicini Fernandes

September/2010

Advisor: Claudio Luis de Amorim

Department: Systems Engineering and Computer Science

In this work we present a comparison between two system virtualization techniques. We compare this techniques with respect to their efficiency, in an environment composed by two physical machines hosting several virtual machines, that can be switched among the physical machines each time such migration is needed, due to load balancing constraints. We also compare the efficiency of this two techniques to migrate theirs virtual machines. In order to build our testing environment, we developed a virtual machine manager that acts as a load balancer. The virtual machine manager is also presented in this work.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Listagens

Capítulo 1

Introdução

Nos últimos vinte anos, a tecnologia da informação tem viabilizado soluções disruptivas, que acontecem com velocidades cada vez mais espantosas. A disponibilidade, a rapidez, a maneira de acesso e a forma como são computados os dados e informações, afeta diretamente a operação das empresas, e também, a forma de se comunicar, sociabilizar e trabalhar das pessoas. Assim, a tecnologia da informação serve de base para mudanças de conceitos e quebra de paradigmas que têm, de maneira cada vez mais frequentes, influenciado a vida de mais pessoas. A virtualização está certamente entre as principais tecnologias capaz de proporcionar tais mudanças.

O uso de virtualização em servidores é uma tendência mundial. A explicação desta adoção, está na maneira como a virtualização de servidores resolve alguns dos problemas impostos às empresas, cujo negócio, está diretamente relacionado a tecnologia da informação. Com a virtualização de servidores, pode-se melhor aproveitar o espaço físico, reduzir substancialmente o custo com manutenção, infra estrutura e energia elétrica. E esta última, além de implicar na redução dos custos, condiz com as práticas das empresas que adotam a TI verde [1]. O impacto da virtualização não se restringe a esses aspectos. Combinada com a computação em *cluster*, a virtualização se torna uma das tecnologias que servem como base para o surgimento da *cloud computing*[2], ou computação na nuvem. A computação na nuvem está mudando a maneira das empresas enxergarem as áreas de TI. E por outro lado, as empresas de TI terão que se adaptar para desenvolver novos produtos e serviços que estarão imersos nessa nova realidade.

Usar a tecnologia de virtualização nos *clusters* de computadores, possibilita aumentar a utilização dos *clusters*. A virtualização permite uma utilização mais flexível

[1] TI verde é um conjunto de práticas para tornar mais sustentável e menos prejudicial o uso da tecnologia da informação

[2] *Cloud Computing* é um modelo de computação onde o processamento, o armazenamento e o software, são acessados remotamente e estão em algum lugar não necessariamente conhecido por quem os usa. Apesar desta ser uma das definições, vale lembrar que *cloud computing* é um paradigma ainda em evolução, cujas definições ainda não são muito claras.

das máquinas de um *cluster*. Os utilizadores de um *cluster* têm flexibilidade para escolher seu próprio sistema operacional e seu próprio ambiente de execução, criando seu próprio *cluster* virtual. O *cluster* físico pode ser usado simultaneamente por vários *clusters* virtuais. As máquinas virtuais podem ser balanceadas entre as máquinas do *cluster*, de maneira a melhorar vazão do sistema. As máquinas do *cluster* podem ser ligadas ou desligadas de acordo com a demanda, visando a economia de energia. O isolamento proporcionado pelo uso de máquinas virtuais nos *clusters* permite que um *cluster* seja usado por aplicações com características diversas. Isso implica em dizer que, pessoas ou empresas poderiam usufruir deste sistema, ou também, que pessoas ou empresas poderiam pagar para que, por exemplo, seus servidores de aplicações fossem executados num *cluster* remoto. Ambientes semelhantes a este, como o EC2[55] da Amazon, já são realidades, e sua adoção, por empresas, é crescente.

Todo este cenário serviu como motivação para alcançar os objetivos desta dissertação.

1.1 Objetivos e Contribuições

O objetivo desta dissertação é comparar a eficiência de duas soluções de virtualização que implementam técnicas distintas. Para isso, será analisado o comportamento das técnicas de virtualização, quando as máquinas virtuais executam em um *cluster* de computadores, sendo que, as máquinas virtuais podem migrar entre as máquinas do *cluster*. As migrações ocorrem com a finalidade de balancear a carga entre as máquinas do sistema e são determinadas por um programa que gerencia as MVs.

As constribuições desta dissertação são as seguintes:

Avaliação de duas técnicas de virtulaização Cumprindo os objetivos propostos, foram avaliadas duas técnicas de virtualização de sistemas. Esta dissertação apresenta e compara as características e dificuldades das técnicas de virtualização quando operam sobre cluster com balanceamento de carga.

Gerenciador de máquinas virtuais Para realizar a avaliação dessas duas técnicas, foi composto um sistema formado por um *cluster* de máquinas, máquinas virtuais executando sobre as máquinas, aplicações executando sobre as MVs e um gerenciador de MVs, responsável por manter a carga balanceada entre as máquinas do sistema. Esse gerenciador está habilitado para operar com as duas técnicas de virtualização.

Implementação de um algoritmo de balanceamento de carga Foi adaptado e implementado o algoritmo eficiente [7] de balanceamento de carga

Assign-U estendido [7]. Este algoritmo foi desenvolvido para operar balanceamento de carga de processo. Nesta dissertação, está implementado para balancear MVs.

1.2 Organização da Dissertação

Esta dissertação se divide em seis capítulos. O Capítulo 2 apresenta a fundamentação utilizada neste trabalho. Neste capítulo, são discutidos os conceitos, classes e técnicas de virtualização. O Capítulo 3 descreve detalhes específicos das técnicas de virtualização usadas nos experimentos. O Capítulo 3, também apresenta os *softwares* e os algoritmos implementados para gerenciar a migração de máquinas virtuais.

O Capítulo 4, mostra os detalhes referentes ao ambiente experimental e a análise dos resultados obtidos nos experimentos.

Finalmente, o Capítulo 6 apresenta conclusões a respeito dos rumos e resultados desta dissertação, assim como sugestões de trabalhos futuros.

Capítulo 2

Virtualização

Neste capítulo, são apresentados alguns fundamentos deste trabalho. Estes fundamentos são importantes para o bom entendimento dos objetivos propostos e da análise experimental. O principal tema abordado neste capítulo é a virtualização, contemplando os conceitos, classes de virtualização e identificando o potencial desta tecnologia.

2.1 Introdução

Apesar do crescimento acelerado do uso de virtualização nos últimos anos, é um equívoco pensar que o conceito de virtualização surgiu no início deste período. Na verdade, os primeiros artigos que tratam deste tema na computação, surgiram há mais de 30 anos. Entre o fim da década de 60 e o início de 70, em um contexto tecnológico diferente do que vivemos atualmente, surgiram as primeiras definições [52] e implementações [54] [51] do uso de virtualização na computação. Foi naquela época que surgiram os primeiros mainframes, que por serem computadores grandes e caros, geralmente tinham que ser compartilhados por um grande número de usuários. Neste cenário, surgia um forte incentivo ao uso de virtualização, já que diferentes usuários, desejavam utilizar SOs diferentes. Com o desenvolvimento da tecnologia, o *hardware* passou a ser mais barato e os mainframes foram substituídos por *desktops*. Desta forma, diminuiu o interesse pelo uso de virtualização.

Recentemente, o uso de MVs voltou a se tornar popular. Os grandes e caros mainframes de ontem, são hoje os também grandes e caros *clusters* computacionais que ainda são compartilhados por muitos usuários e também estão habilitados a usar virtualização. Um outro motivo para o "retorno" da virtualização está no grande poder computacional dos *desktop* atuais e na otimização das novas tecnologias de virtualização, que em conjunto, tornam suportável o *overhead* resultante do uso da virtualização, mesmo em computadores pessoais.

Os benefícios que o uso de virtualização pode trazer são muitos, entre eles podemos citar:

- **Portabilidade**: Permite executar um programa binário compilado para uma arquitetura diferente da máquina física que será utilizada na execução.

- **Múltiplos Ambientes Seguros**: Várias máquinas virtuais podem executar, simultaneamente, em uma mesma máquina física. Cada MV tem a ilusão de que a máquina física é exclusivamente sua. Qualquer problema que ocorra dentro de uma MV, seja por executar um programa malicioso ou até mesmo por existir algum *bug* no sistema operacional da MV, é improvável que as outras MVs hospedadas na mesma máquina sejam afetadas. Essa característica permite que máquinas hospedem diferentes serviços em diferentes MVs sem que uma eventual falha de segurança de um serviço venha comprometer a segurança do outro.

- **Múltiplos Sistemas Operacionais**: Em uma única máquinas pode-se ter diferentes MVs com sistemas operacionais diferentes. Permite que o usuário utilize, simultaneamente, na mesma máquina, diferentes programas que executam em SOs distintos. Também é uma boa solução para o problema de usuário da mesma máquina que necessitam de usar SOs diferentes.

- **Monitorar eventos**: Algumas implementações de máquinas virtuais possibilitam ao desenvolvedor monitorar com mais facilidade alguns eventos do sistema. A virtualização pode facilitar a obtenção do *trace* e permite até mesmo ter uma imagem ou um *dump* da máquina virtual, e a partir desta imagem, várias instâncias da MV original podem ser criadas, facilitando ao desenvolvedor encontrar erros no sistema.

- **Migração das Máquinas Virtuais**: A virtualização permite a migração de MVs entre diferentes máquinas de forma transparente para a aplicação, ou seja, as aplicações continuam a execução naturalmente como se ainda estivessem na mesma máquina. Migrar uma MV entre máquinas pode ser muito útil quando se identifica que uma máquina está sobrecarregada ou até mesmo quando algum recurso de *hardware* da máquina está prestes a falhar. A capacidade de migração pode ser considerada uma das mais importantes vantagens do uso de virtualização. Esta característica é muito relevante neste trabalho e será mais explorada neste e nos capítulos seguintes.

2.1.1 Fundamentos

Os computadores modernos estão certamente entre as máquinas mais poderosas e complexas produzidas pelo homem. Esses computadores contêm diversos *chips*, onde cada *chip*, por sua vez, contém centenas de milhares de transistores. Estes pequenos *chips* ficam interligados fisicamente entre si e com outros dispositivos. Se adicionarmos a esta estrutura o SO, aplicações, *drivers* e bibliotecas; temos uma poderosa máquina cujos limites ainda são desconhecidos.

Juntar esses diversos componentes de *hardware* e software para formar um complexo sistema computacional, se torna mais fácil quando este sistema é dividido em camadas de abstração separadas por uma interface bem definida. As camadas de abstração formam uma hierarquia onde as camadas de *hardware* são classificadas como de baixo nível e as de software, camadas de alto nível. Uma vantagem notável do uso de camadas de abstração com interfaces bem definidas, é poder desenvolver uma camada independentemente do desenvolvimento da outra. Comunidades de desenvolvedores ou empresas podem assim, concentrar seus esforços no desenvolvimento de apenas um, ou alguns dos níveis da camada. Determinada equipe de trabalho não precisa conhecer os detalhes das camadas adjacentes, pois a interface entre as camadas já estava definida, e fica como responsabilidade de outra equipe implementar o que foi definido pela interface de um componente da camada adjacente. Um exemplo claro de empresas que trabalham neste contexto são a AMD[31] e a Microsoft[28]. Enquanto a AMD desenvolve microprocessadores com conjunto de instruções Intel IA-32, a Microsoft desenvolve compiladores que mapeiam linguagens de alto nível de abstração para este mesmo conjunto de instruções.

Essa divisão de camadas com interfaces bem definidas não traz somente vantagens. Ter interfaces bem definidas entre as camadas limitam sua portabilidade, pois, os componentes desenvolvidos para uma interface não conseguem operar com outros que implementam uma interface distinta. Existem processadores com diferentes conjuntos de instruções, assim como existem também SOs com arquiteturas muito distintas. Aplicações que já estão em código binário, teoricamente só podem executar, de forma direta, sobre um determinado conjunto de instruções e SO específicos. Resolver este problema talvez seja uma das características mais notável da virtualização. Voltar a atenção para este fato pode ajudar na compreensão do que vem a ser a virtualização.

Definição

A Virtualização foi a maneira encontrada para dar maior portabilidade aos componentes das camadas e subsistemas. Virtualizar um sistema (ou subsistema) é mapear as interfaces e demais recursos, visíveis para outros sistemas que operam so-

bre a camada de virtualização, para as interfaces e recursos existentes sob a camada de virtualização. A camada de virtualização pode informar para esse sistema, que o *hardware* que está sob ela, possui mais, menos, e até recursos diferentes do que este realmente possui. A Figura 2.1, representa a arquitetura de um sistema sem virtualização, já a Figura 2.2, apresenta a arquitetura tradicional de um sistema virtualizado. Na Figura 2.2, temos a camada de virtualização inserida entre o SO e a camada de *hardware*, os traços que separam a camada de SO e de virtualização são diferentes dos que separam a camada de virtualização da camada *hardware*. Esse traços são diferentes, para representar a diferença entre as interfaces, o que significa que o SO não é diretamente dependente do hardware.

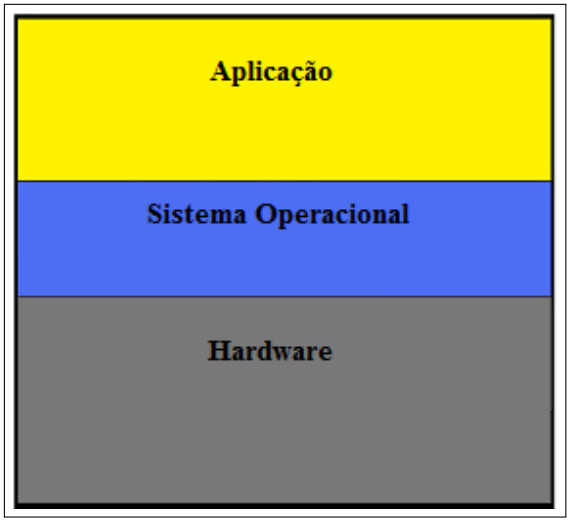

Figura 2.1: Arquitetura de um sistema não virtualizado.

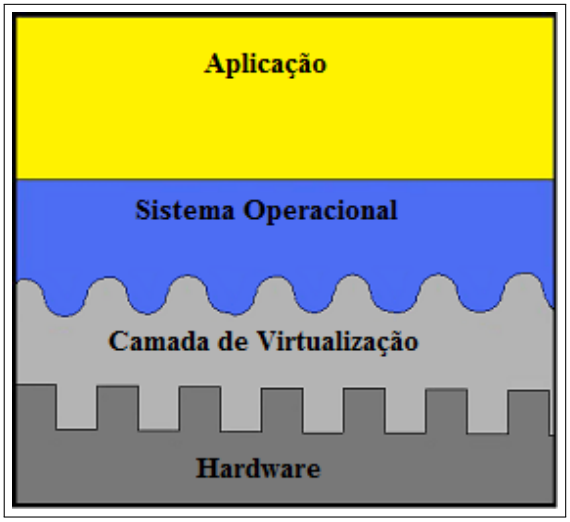

Figura 2.2: Arquitetura de um sistema virtualizado.

Formalmente, pode-se ver o processo de virtualização como um isomorfismo[52]. Como ilustrado pela Figura 2.3, apresentada na publicação [52]. Para uma sequência de operações no sistema virtual, que leva este sistema de *Si* para *Sj*, existe uma sequência de operações *e'* que modifica de forma equivalente o sistema real de *S'j* para *S'j*. De maneira mais restrita, pode-se pensar que as operações no sistema virtual são requisitadas pela camada de SO à camada de virtualização, e as operações no sistema real são as operações requisitadas pela camada de virtualização à camada de *hardware*.

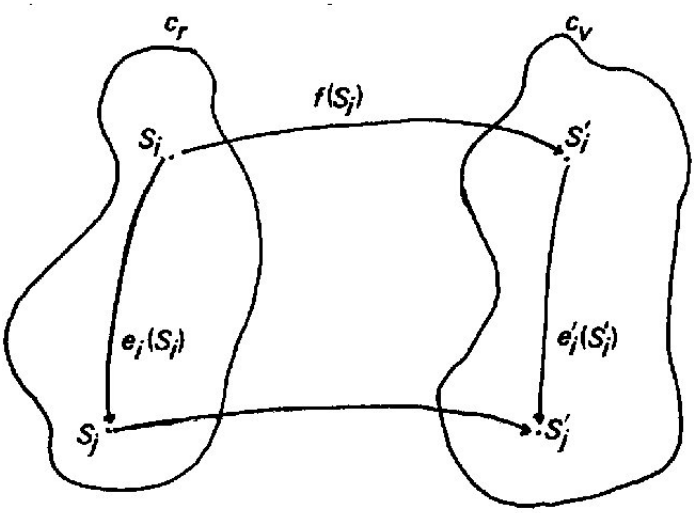

Figura 2.3: Mapeamento do sistema virtual em um sistema real.

2.1.2 Classes de Virtualização

Antes de apresentar as diferentes técnicas de virtualização, é instrutivo, falar sobre o significado de máquina no ponto de vista de um processo de usuário e do sistema operacional.

Na perspectiva de um processo executando um programa de usuário, uma máquina consiste em um espaço de endereçamento lógico onde ficam armazenados seus registradores e instruções, permitindo a execução do código e armazenamento de seu estado. Toda a parte de I/O de processo só pode ser realizada por intermédio do sistema operacional. Desta maneira, pode-se dizer que, para um processo, a máquina é uma combinação de *hardware* com o SO.

Na perspectiva do SO, a máquina subjacente suporta a execução de todo o sistema, onde o sistema é todo o ambiente de execução que suporta os processos de diferentes usuários. O SO é o responsável por compartilhar, entre os processos usuários, todos os recursos de *hardware* disponíveis. Enquanto os processos iniciam, executam e terminam, o SO deve permanecer executando normalmente e em caso de *reboot* ele deve manter seu estado. Pode-se dizer que para o SO, uma máquina é todo o hardware subjacente, com o qual ele se comunica através do ISA[1] [46].

Como caracterizado pelo isomorfismo apresentado anteriormente, o processo de virtualização se divide em duas partes. A primeira consiste no mapeamento dos recursos virtuais como registradores, memória e arquivos em recursos reais da máquina física. A segunda parte é o uso das instruções da máquina real para realizar

[1] *Instruction Set Architecture* Representa os limites entre o software e o hardhare.

as ações solicitadas pela máquina virtual.

Tradicionalmente, as máquinas virtuais são divididas em duas principais classes, são elas: máquinas virtuais de processo e as máquinas virtuais de sistema. Essas classes refletem as duas diferentes conotações de máquina descritas anteriormente, isto é, enquanto a primeira reflete a virtualização da máquina na forma como é vista pelo processo, a segunda representa a virtualização da máquina na visão do SO. Apesar de não ser consenso entre os autores da área [49], a classe de máquina virtual no Nível do SO ou de *Container* é considerada como uma classe intermediária entre classe de MV de processo e a classe de MV de sistema.

Máquinas Virtuais de Processo

As máquinas virtuais de processo são criadas para suportar apenas uma aplicação. A máquina é criada quando a aplicação inicia, e descartada quando o processo termina. O software de virtualização desta classe é comumente chamado de *runtime* e se localiza acima de uma combinação *hardware* e software vista pelo processo. As Figuras 2.4 e 2.5 refletem respectivamente, a arquitetura desta classe e a visão que o processo tem da máquina sobre a qual ele executa. Note que o processo da aplicação só interage com o software de virtualização, isso implica que sua execução independe do tipo de hardware e SO sobre o qual ele executa, pois toda a interação com o hardware e com SO é de responsabilidade da camada de virtualização. Existem várias implementações de MVs de processo, a mais comum, que também pode ser considerado como virtualização de processos, é a multiprogramação, empregada em quase todos os SOs atuais e fornece a um processo a ilusão de ter toda a máquina exclusivamente para ele. Um outro tipo, que se tornou popular com o uso da Java Virtual Machine [53], provê a independência da plataforma utilizada. Esse tipo de MV, no nível de processo, permite a portabilidade da execução do processo, e seu ambiente virtual não necessariamente correspondem ao de alguma plataforma real existente.

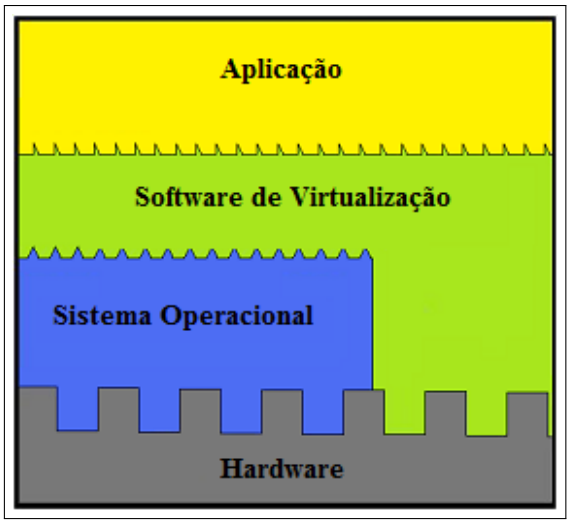

Figura 2.4: Arquitetura das camadas na virtualização de processo.

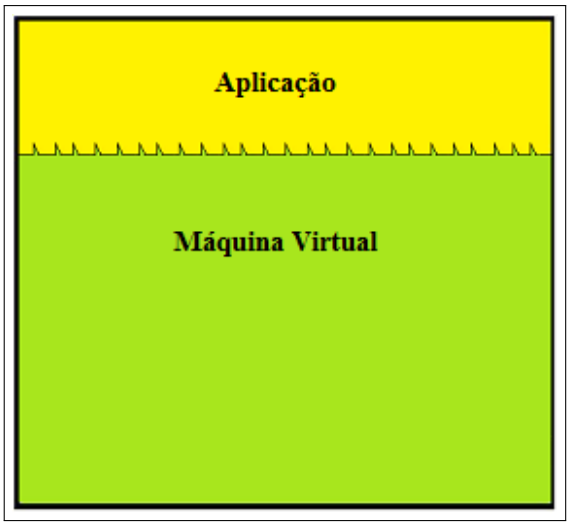

Figura 2.5: Visão da processo, na virtualização de processo.

2.1.3 Máquina Virtual de Sistema

As máquinas virtuais de sistema provêm o ambiente completo do sistema no qual possivelmente coexistem muitos processos de diferentes usuários. Nesta classe, o SO hospedado deve ter acesso aos recursos do hardware subjacente, incluindo os de rede e a interface gráfica. As Figuras 2.6 e 2.7 representam, respectivamente, a arquitetura e a visão do SO sobre a virtualização de sistemas. Na Figura 2.7 nota-se que a execução do SO se dá de maneira independente em relação ao hardware. O SO depende da camada de virtualização, deixando toda interação com o *hardware* por conta da mesma. O software representado na camada de virtualização, responsável pela virtualização do sistema é denominado Monitor de Máquina Virtual(MMV). Também existem diferentes implementações de máquinas virtuais de sistemas, e por serem parte importante deste trabalho, serão melhor exploradas nas seções seguintes.

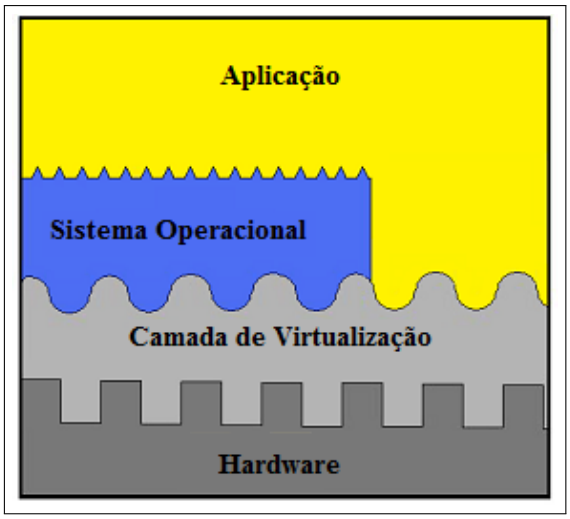

Figura 2.6: Arquitetura das camadas na virtualização de sistema.

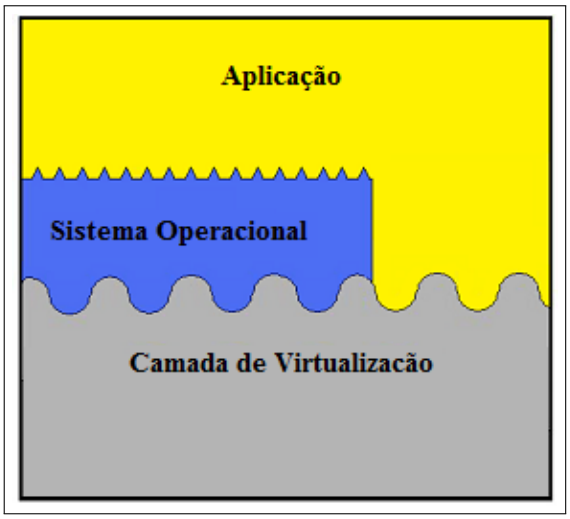

Figura 2.7: Visão da máquina virtual na virtualização de sistema.

Como o foco do trabalho está em comparar alguns técnicas de virtualização de sistemas, não será descrita a virtualização de processos. Porém é importante conhecer mais detalhes das técnicas de virtualização de sistemas, pois a análise dos resultados apresentados neste trabalho dependem do conhecimento de algumas das técnicas de virtualização de sistemas. Na próxima seção serão detalhadas algumas dessas técnicas.

2.2 Técnicas de virtualização de Sistemas

Existem diferentes técnicas de virtualização de sistemas. Antes de escolher uma técnica, ou uma tecnologia de virtualização, deve-se saber se a escolhida é mais adequada para atender as necessidades pré-definidas. Geralmente, as características mais preponderantes na escolha do tipo de virtualização estão diretamente relacionadas ao isolamento e a eficiência proporcionada pelo tipo [52]. O isolamento indica o nível de independência entre as MVs, ou seja, o quanto um erro ou o consumo de recursos em uma MV pode influenciar a execução de outra MV. Já a eficiência está ligada ao nível de *overhead* gerado pela uso da técnica de virtualização, ou seja, quanto mais eficiente menor o *overhead*. Nas próximas subseções são apresentadas conceitos básicos das técnicas consideradas importantes para esta dissertação.

2.2.1 A Virtualização Clássica

A principal característica deste tipo de virtualização é a presença do software de virtualização, conhecido como monitor de máquina virtual (MMV), ou em inglês

virtual machine monitor (VMM), que fica localizado entre o hardware e as MV hospedadas. Para esta técnica, é o MMV quem deve gerenciar o compartilhamento dos recursos de hardware disponíveis para as máquinas virtuais. O MMV detêm os recursos físicos do hardware e pode torná-los disponíveis para as MVs hospedadas. Cada MV possui a ilusão de que tem os recursos de forma exclusiva. Os recursos virtuais vistos pelas MVs podem ou não ter um recurso físico correspondente. Quando o recurso não existir fisicamente, o MMV deve emular as funções deste recurso virtual. Geralmente isso é feito por uma combinação entre software e outro recurso real disponível no sistema. É responsabilidade do MMV agendar e gerenciar a alocação de recursos como registrador de cpu, memória real do sistema e os dispositivos de *IO* disponíveis no sistema. Por essas características, pode-se dizer que a relação entre um SO e as aplicações que nele executam, são semelhantes a relação existente entre o MMV e as máquinas virtuais hospedadas.

Como apresentado nas Figuras 2.8, 2.9 e 2.10, a disposição do MMV pode variar em relação aos outros componentes do sistema. Na Figura 2.8 o MMV é o único componente que executa no nível de privilégio mais alto, MVs executam em modo menos privilegiado. O MMV deve emular o nível de execução dos SOs hospedados para que estes sejam executados como usualmente. O problema desta configuração está na necessidade de remover o SO existente na máquina antes de instalar o MMV. Na Figura 2.9, existe um SO hospedeiro abaixo do MMV, executando em um nível de privilégio maior que este. Neste caso, apesar de não ser preciso retirar o SO para instalar o MMV, esta configuração perde em termos de eficiência, pois insere uma camada de software a mais entre as aplicações e o hardware. Uma forma de aproveitar um pouco da vantagem das duas disposições anteriores, seria permitir que o MMV executasse tanto em modo privilegiado quanto em modo usuário, como apresentado na Figura 2.10, onde o MMV pode aproveitar os benefícios de executar nos dois níveis de privilégio.

Abaixo, é apresentado como é realizada a gerência dos recursos por essas técnicas de virtualização:

Virtualização dos Processadores

Virtualizar um processador implica em executar as instruções de todas as MVs hospedadas no sistema. Essa execução pode proceder de duas maneiras. A primeira, por meio de emulação. Nesta, o MMV deve examinar a instrução e emular essa instrução executando uma, ou um conjunto de instruções para produzir um resultado equivalente ao da instrução emulada. Emulação é a única maneira de virtualizar quando a ISA do processador hospedado é diferente da ISA suportada pela MV hospedada. A segunda forma é a execução direta da instrução no processador. Executar a instrução diretamente é, geralmente, mais rápido do que emular sua

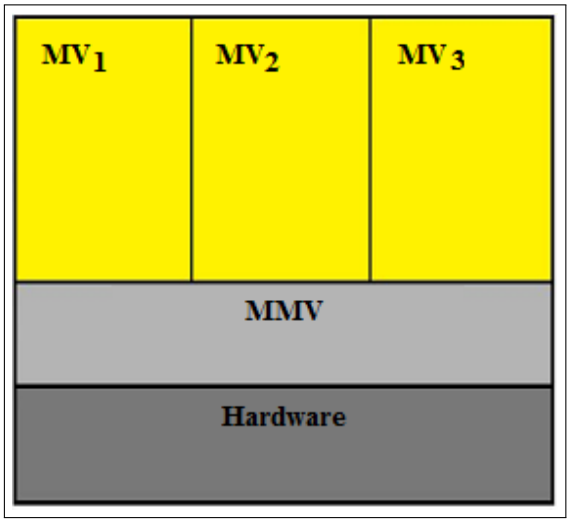

Figura 2.8: Disposição do Monitor de Máquina Virtual nas camadas do sistema virtualizado. O MMV com o maior privilégio do sistema.

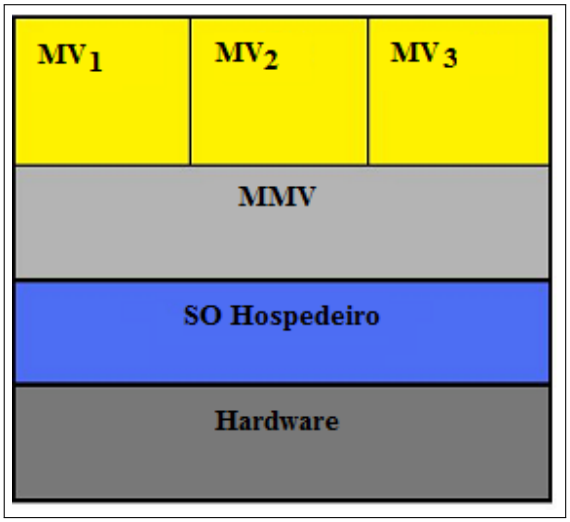

Figura 2.9: Disposição do Monitor de Máquina Virtual nas camadas do sistema virtualizado. O MMV no nível de privilégio de aplicação do SO.

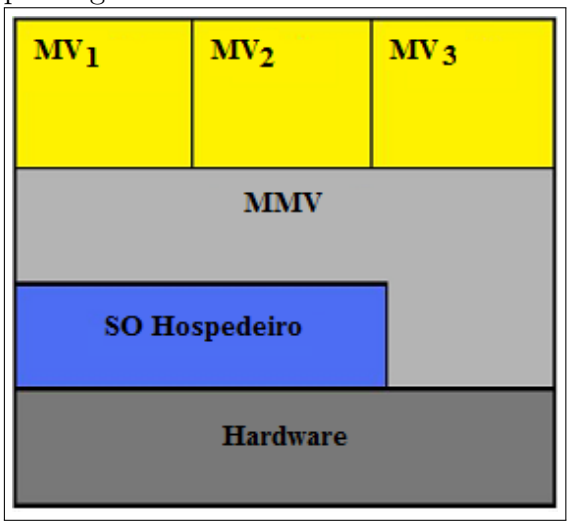

Figura 2.10: Disposição do Monitor de Máquina Virtual nas camadas do sistema virtualizado. O MMV ora no nível de privilégio da aplicação, ora no maior nível de privilégio do sistema.

execução. O problema da segunda forma de execução é que esta só pode ser utilizada quando o processador possui a mesma ISA suportada pela MV hospedada.

Mesmo quando o MMV, executando seguindo o modelo de camadas apresentado na Figura 2.8, e a ISA do processador for igual a ISA suportada pelas MVs, as MVs devem executar sempre em modo menos privilegiado que o MMV, o que é importante para que o MMV mantenha o controle do sistema [52] Mas existem instruções destas MVs que necessitam executar em modo privilegiado para que procedam corretamente. O MMV deve então emular estas instruções para que as MVs executem como se detivesse o modo privilegiado do sistema. Essa emulação gera *overhead*. Existem também instruções não privilegiadas que não podem executar diretamente sem a intervenção do MMV. Essas instruções são críticas para o MMV, pois diferentemente das instruções privilegiadas elas não geram interrupção, dificultando a tarefa do MMV. A arquitetura Intel IA-32 por exemplo, contém muitas instruções desta categoria. Por este motivo o IA-32 é considerada uma arquitetura que não é eficientemente virtualizável [46]. Quanto mais execução direta, menos emulação de instrução, mais eficiente a virtualização se torna. Desta forma, a eficiência da virtualização está diretamente relacionada com a forma que a MMV lida com essas instruções privilegiadas [45]. A maneira como o MMV identifica e lida com estas instruções é outro fator influente na eficiência desta virtualização.

A tecnologia de virtualização Xen[29] modifica o SO hospedado a fim de contornar as dificuldades inerentes a virtualização de processadores com arquitetura IA-32. Este tipo de virtualização que modifica o SO hospedado é chamada de Paravirtualização. Devido a essas modificações, a interface da máquina virtual provida pelo Xen é ligeiramente diferente da arquitetura real do processador. Apesar de invasiva, essa modificação é justificada pelo ganho de desempenho gerado pela diminuição no *overhead* da virtualização do processador. Já o VMWare[30], outra tecnologia de virtualização da classe de virtualização de sistemas, aplica a técnica de Virtualização Total. Esta técnica utiliza o MMV, mas não faz nenhuma modificação no SO hospedeiro. O MMV do VMware é obrigado a examinar sequencialmente os blocos de instruções, antes da execução, em busca de instruções críticas. Caso encontre uma instrução crítica, esta será substituída por uma instrução de interrupção para o MMV. Quando esta interrupção for executada, o MMV assume o controle e emula a execução da instrução crítica. Não modificar o SO traz a vantagem de não precisar alterar as camadas de software adjacentes. Mas o grande problema de não faze-la é o *overhead* necessário para virtualizar a arquitetura IA-32. Esse *overead* pode ser inaceitável em algumas situações.

Virtualização da Memória

Pode-se considerar que a virtualização de sistema é uma generalização dos conceitos utilizados na memória virtual, atualmente implementado nos SOs [43] . Assim como uma MV hospedada vê os recursos do sistema de maneira diferente do que realmente é, a memória que o SO aloca a um processo também é diferente da memória física que é suportada. A memória virtual dos OSs comuns, sem virtualização, necessita de uma tabela de páginas para fazer o mapeamento da endereço virtual de um processo para a correspondente memória física deste. Para utilizar a memória na virtualização do sistema, é geralmente, necessário incluir mais uma camada entre a memória física e a memória virtual vista pelo processo. Isso é feito adicionando mais uma tabela, para cada MV hospedada, que mapeia o endereço real em endereço físico. Observe que neste caso, o endereço real não corresponde diretamente ao endereço físico. O MMV passa à MV hospedada a ilusão de que o endereço real corresponde ao endereço físico, mas na verdade, ainda é preciso utilizar uma tabela para traduzir o endereço real para o endereço físico.

Se a virtualização fosse utilizada simplesmente como apresentado até aqui, seria necessário fazer duas traduções de endereço para, a partir do endereço virtual, chegar ao endereço físico. A fim de diminuir este *overhead*, o MMV mantém, além das páginas de tradução virtual-real e real-físico, é criada uma página sombra que faz a tradução direta de endereço virtual para endereço físico. Para utilizar essa tradução direta, também é preciso que o MMV virtualize o registrador que contém o ponteiro para a tabela de página dos processos da MV, fazendo que ele aponte para a tabela de página sombra. Apesar de diminuir o *overhead*, a utilização da tabela de páginas sombra não soluciona todos os problemas relacionados com eficiência da virtualização da memória pelo MMV. Quando ocorre uma operação de E/S(Entrada e Saída) com os endereços reais (como é comum nos SOs), o MMV precisa utilizar a tabela que contém a tradução real-físico para converter os endereços. O que pode degradar o sistema neste caso, é o fato das páginas reais contíguas utilizadas no E/S não necessariamente corresponderem a páginas físicas contínuas, o que pode prejudicar o mecanismo de cache do sistema [44].

Para tecnologias que utilizam virtualização total, como é o caso do VMware, se faz necessário o uso de páginas sombra como descrito acima [1]. Para o Xen, uma tecnologia que usa paravirtualização, cada SO mantém sua tabela para a tradução direta de endereço virtual para endereço real. Para garantir o isolamento, as MVs têm apenas direito de leitura nas tabelas de página. Quando um SO de uma VM hospedada deseja alterar essa tabela, ocorre então uma interrupção para o MMV, para que este verifique se a operação do SO é permitida, executando-a em caso positivo [40]. Tanto o Xen quanto o VMware deixam as decisões de alocação e

transferência entre o disco e a memória, *swap* a cargo dos SOs hospedados, isso impede que decisão conflitantes entre o SO e o MMV degradem a eficiência do sistema.

Virtualização de E/S

A virtualização dos dispositivos de entrada e saída constitui uma das partes mais complexas da implementação de um sistema de máquinas virtuais. Além de existir uma grande e crescente quantidade de tipos de dispositivos de IO, cada um destes tipos possuem características peculiares, o que implica na necessidade de *drives* específicos para cada tipo de dispositivo. Este fato dificulta a implementação da virtualização destes dispositivos. Abaixo, seguem as três estratégias mais utilizadas na implementação da virtualização de dispositivos:

E/S Direta Na primeira estratégia, a de E/S direta, o dispositivo alvo é acessado diretamente pelo *driver* do dispositivo instalado na máquina virtual. Neste caso não existe intervenção direta do MMV na comunicação entre a aplicação da MV e o dispositivo. Se por um lado esse fato pode tornar a virtualização do dispositivo eficiente, por retirar um intermediário na comunicação entre a aplicação e o dispositivo, por outro lado, a ausência do MMV nesta comunicação dificulta o compartilhamento do dispositivo em questão.

E/S Virtual, com transação emulada Na transação emulada, somente uma MV especial terá acesso direto ao dispositivo. Todos as outras MVs do sistema que desejarem utilizar o dispositivo, devem interagir com a MV especial para que esta interaja com o dispositivo. Esta interação ocorre da seguinte forma: primeiramente um controlador virtual de uma MV recebe, de uma aplicação desta MV, um pedido de E/S para algum dispositivo, então o controlador virtual repassa o pedido, através do MMV, para um controlador nativo da máquina especial. Após receber o pedido, o controlador nativo da MV especial interage diretamente com o dispositivo a fim de atendê-lo. Devido a necessidade da inclusão dos controladores especiais nos SOs das MVs, a estratégia de transação emulada é utilizada em sistemas paravirtualizados. O Xen utiliza esta estratégia para virtualizar os dispositivos do sistema e para agilizar a transferência dos dados do E/S para os domínios utilizam um esquema de anel de E/S,conforme descrito em [40].

E/S usando emulação de dispositivo Na última estratégia, o MMV possui a tarefa de emular o dispositivo para que a MV, juntamente com o seu software de controle do dispositivo, tenham a ilusão de estar interagindo diretamente com o dispositivo emulado. O MMV intercepta as iterações da MV com o

dispositivo, emula e retorna o que foi pedido pela MV ao dispositivo emulado. Esta estratégia é menos invasiva e mais limitada do que a anterior. Menos invasiva pois não se faz necessário modificações no SO e controlador de dispositivo. Mais limitada porque a capacidade de emulação do MMV está limitado aos recursos que este possui. Além disso, essa estratégia não é eficiente, pois a emulação do dispositivo por parte do MMV adiciona ainda mais *overhead* a virtualização. Por suas características, esta técnica é utilizada na Virtualização Total. A fim de evitar as limitações e o *overhead* acima abordados, uma das versões do VMware, VMWare Workstation, aproveita o fato do MMV como um processo usuário no SO hospedeiro, como na Figura 2.9 para efetuar a virtualização de seus dispositivos [39]. A virtualização ocorre da seguinte maneira, sempre que alguma aplicação do SO de alguma MV fizer uma requisição de IO, a parte do MMV acoplada ao hardware converte esta requisição para uma aplicação do SO hospedeiro. Assim, a requisição pode ser atendida por um software controlador instalado no SO hospedeiro. Uma vantagem desta abordagem é que não é necessário colocar softwares controladores no MMV. Por outro lado, muitos pedidos de IO podem sobrecarregar o SO hospedeiro.

2.2.2 Virtualização no Nível do Sistema Operacional

Esta classe de virtualização, também é conhecida como virtualização por Containers, e se apresenta como uma alternativa à virtualização clássica. Existem cenários que requerem sistemas de virtualização mais eficientes e não requerem diferentes sistemas operacionais. Para estes casos, a virtualização por containers pode ser mais adequada. Nesta técnica, o *kernel* do sistema operacional principal é compartilhado entre todas as máquinas virtuais do sistema. O fato de compartilhar o *kernel*, evita que mais uma camada de software seja adicionada ao sistema, o que torna mais eficiente as técnica desta classe. Este mesmo fato torna as máquinas virtuais menos isoladas em relação a técnicas da virtualização clássicas, já que um problema no *kernel* pode afetar todas as máquinas. A Figura 2.11 representa bem o que é esta técnica. Os processos das MVs são segregados em conteiners distintos. Assim, no ponto de vista do *kernel*, as MVs são um conjunto de processos. Já para a MV, ela tem a impressão de que está executando de forma exclusiva no *kernel*, ela não consegue "ver"os processos que estão em outros containers. Também é mostrado a separação entre plataforma virtual e a plataforma hospedeira. A plataforma virtual representa a visão do sistema pelas MVs. A plataforma hospedeira, representa a visão do *kernel* compartilhado.

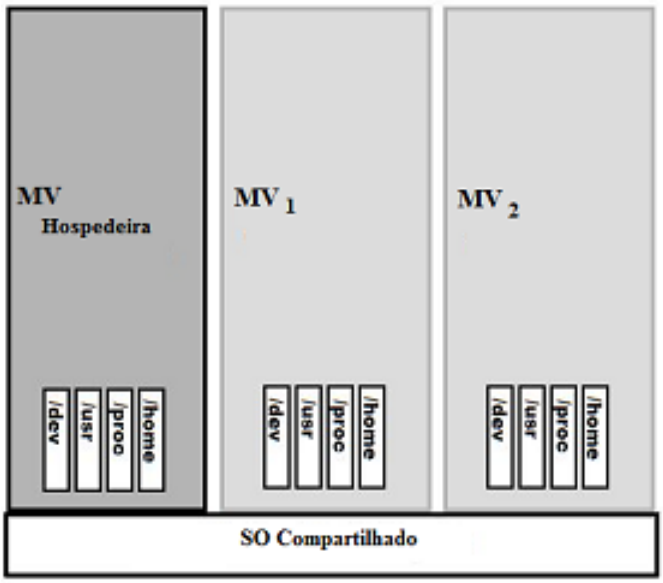

Figura 2.11: Virtualização no nível do SO.

Existem diversas tecnologias pertencente a esta classe de virtualização, entre eles o OpenVZ [33], Virtuozzo [37], Linux-VServer [36], Solaris Zones [35] and FreeBSD Jails [34]. A diferença básica entre eles está no SO hospedeiro e nas técnicas empregadas no Kernel do SO para isolar as MVs.

ssaryname=OpenVZ, description=OpenVZ é uma tecnologia de virtualização em nível de sistema operacional baseada no sistema operacional e núcleo Linux. O OpenVZ é uma base do Parallels Virtuozzo Containers, um software proprietário fornecido pela Parallels. O OpenVZ é licenciado sob a versão 2 da GPL.

As técnicas básicas de segurança nos sistemas de virtualização por Containers são:

- Espaço de endereçamento separados para cada MV, ou seja, contexto diferente para containers diferentes.
- Controle de acesso por meio de filtros; aqui os filtros têm um papel semelhante ao que os MMV tem na classe MV de sistemas. Os filtros controlam o acesso aos objetos do *kernel*, verificando em tempo real se a execução do container está de acordo com suas permissões de acesso.

Virtualização do Processador

Diferente da virtualização clássica, esta técnica, não emprega nenhuma emulação do processador. As aplicações das MVs executam da mesma maneira que executariam em SO sem virtualização. Elas veem o processador da mesma maneira que viriam se executadas em um SO comum. A diferença para os SOs sem virtualização é que

aqui os PID dos processos são virtualizados; cada Container conhece apenas os seus processos. Outra diferença é que também é necessário escalonar o processador entre as máquinas virtuais.

Virtualização da Memória

Não difere muito da forma empregada no SOs comuns. O fato de possuir os filtros e colocar cada MV em espaços de endereçamento distintos, permite que esta técnica use a memória virtual como nos SOs comuns.

Virtualização de IO

Parte dos problemas enfrentados na Virtualização Total e na Paravirtualização com os *drives* dos dispositivos, não existem na virtualização por container. Como na Virtualização por Containers se tem apenas um *kernel* para todas as MVs, o problema dos *drivers* para técnica, se resume a forma como este recurso será compartilhado entre as MVs. Para um dos mais importantes recursos de IO, a interface de rede, a forma de virtualização empregada pelos produtos desta classe buscam em comum as seguintes características:

1. Cada container possui seu próprio IP;
2. O tráfego dos container são isolados;
3. Cada container deve ter suas próprias regras de *firewall*;

Cada produto aplica técnicas diferentes com finalidades parecidas. No Solaris é criado um *Switch* virtual e também as *VNICs* (*virtual network interface card*), interfaces de virtuais de rede para cada container. As *VNICs* permitem que cada container possua seu endereço de IP, enquanto o switch direciona os dados da interface física para a interface virtual de cada container, permitindo o isolamento.

2.3 Migração de Máquinas Virtuais

Um benefício direto da virtualização de sistemas, é poder realizar balanceamento de carga e prevenção de falhas migrando as MVs. A migração de MVs soluciona muitos dos problemas que se tem na migração de processos, como os apresentados nos trabalhos [41],[47] e [48]. A migração de MVs se beneficia da capacidade que possuem os sisteama virtualizados de migrar instâncias de SOs entre máquinas físicas distintas. Existem diversas maneiras de fazer essa migração, e em algumas dessas maneiras, a migração pode ocorrer de forma transparente, tanto para as aplicações executadas na MV migrada, quanto para o usuário das aplicações. Assim, uma

MV pode migrar de uma máquina física para outra, sem que o usuário saiba que a MV mudou de máquina física. Possivelmente, essa é uma das características mais notáveis da virtualização e parte do sucesso atual da virtualização em *clusters e data centers* deve-se a essa potencialidade. A migração de MVs tem sido utilizada em *clusters* e *data centers*, principalmente por permitir economia de energia, otimização do recursos e prevenção contra falhas [26]. Para economizar energia, sempre que os recursos de um *cluster* ou de um *data center* estiverem subutilizados, pode-se migrar todas as MVs para um conjunto de máquinas físicas e desligar as outras máquinas físicas que não fazem parte deste conjunto. Para otimizar os recursos, pode-se fazer o balanceamento da carga das máquinas físicas. Neste caso, deve-se migrar as MVs das máquina sobrecarregadas para as máquina subutilizadas. Uma forma de se prevenir contra falha de hardware seria migrar as MVs de uma determinada máquina, assim que esta apresentasse algum indício de falha.

Por questões de eficiência, todas as formas de migração devem buscar diminuir tanto o *downtime* [2] quanto o tempo total de migração[3]. Dependendo das aplicações, o impacto de um, pode ser mais importante que o outro. Por exemplo, se o que motivou a migração de uma MV foi uma falha na máquina física, é necessário que o tempo total de migração seja pequeno. Mas no caso da migração de alguma MV que execute aplicações interativas, um pequeno *downtime* passa a ser de maior importância. Abaixo, são apresentadas as principais formas de migração utilizadas pelas técnicas de virtualização de sistemas.

Pára e copia Esta é a forma mais simples de migração e favorece o baixo tempo de migração total da MV, já que este tempo irá depender basicamente do tamanho da memória total da MV. Por outro lado, esta é a forma que apresenta o pior *downtime*, pois a MV migrada não executará durante todo o período de migração. Para migrar uma MV utilizando este método, deve-se primeiramente parar a MV. A partir deste momento a MV e todas suas aplicações não estão mais executando. O próximo passo então, é transferir toda instância do SO de uma máquina física para outra. Após a transferência, a MV pode começar a executar na máquina de destino.

Cópia sobre demanda Nesta forma o *downtime* é sempre muito baixo. Já o tempo total de migração é alto. Aqui, o primeiro passo é uma pequena parada na aplicação para a transferência das estruturas essenciais do *kernel*. Após a transferência, a MV começa a executar na máquina de destino. As partes restantes, que ainda ficaram na MV de origem, são enviadas para o destino no

[2]Downtime é o tempo em que a MV fica totalmente parada devido a algum passo da migração, neste instante nenhuma das aplicações desta MV estará executando.

[3]O tempo total de migração é o tempo que decorrente desde o início da migração até o seu final.

momento do primeiro uso no destino. Note que enviar uma página somente no momento do seu uso degrada a aplicação, pois é necessário esperar a página migrar da origem até o destino. Outro fato importante é o tempo total de migração que pode ser excessivamente longo, pois a migração só será concluída quando toda, ou quase toda, a memória da MV for utilizada na máquina de destino.

Pré-cópia A Pré-cópia pode ser vista como um balanço entre as outras duas formas de migração. Pelas suas vantagens, a pré-cópia é utilizada pelas tecnologias de virtualização como a principal forma de migração de suas MVs. A pré-cópia utiliza fases iterativas onde a MV continua executando na origem, e somente no último passo iterativo é que a MV para de executar na origem e passa a executar no destino. Na primeira iteração, a máquina continua executando e toda a memória é copiada para a máquina de destino. Na iteração N só são copiadas as partes de memória modificadas após a iteração N-1. No último passo, após ser interrompida a execução da MV, as páginas modificadas na iteração anterior são copiadas para a máquina de destino, após a cópia a MV inicia a execução na máquina de destino encerrando assim a migração.

As diferentes técnicas de virtualização exigem formas diferentes para migrar suas MVs. As características de cada técnica de virtualização, e principalmente, a forma como a memória de cada MV é vista pela técnica, faz com que cada técnica aborde a migração da MV de maneira diferente. O fato de diferentes técnicas procederem a migração de suas MVs de forma distinta, está entre as principais motivações para esta dissertação.

Este capítulo apresentou os fundamentos de virtualização, enfatizando a virtualização de sistemas. Foram descritas as duas principais classes de virtualização de sistemas, são elas, virtualização clássica e virtualização no nível do SO. Foram apresentadas também, as características das técnicas de virtualização pertencentes a estas duas classes. A maneira como uma técnica gerencia os recursos do sistema e a forma que esta efetua migração de suas MVs, são fundamentos bastante explorados nos próximos capítulos.

Capítulo 3

Virtualização e Balanceamento de Carga

Neste capítulo, são apresentados os sistemas que servirão de bases para o ambiente experimental explorado no capítulo seguinte. Começando pelas técnicas de virtualização, cuja apresentação recorre aos conceitos contidos no capítulo anterior, acrescidos de características mais específicas das técnicas de virtualização explorados. O capítulo é concluído com a explicação do sistema de balanceamento de carga implementado e utilizado nos experimentos deste trabalho.

Existem trabalhos científicos [25],[24],[23],[22],[21],[19] que comparam o desempenho de diferentes tecnologias, representantes de cada uma das classes de virtualização apresentadas no capítulo anterior. Os aspectos mais relevantes nessas comparações são, certamente, a eficiência e o isolamento das máquinas virtuais [52]. Quando se compara o isolamento, é considerada a probabilidade de *bug* no software que gerência os recursos das máquinas, e também o quanto uma MV executando aplicações maliciosas ou apenas sedentas por recursos, influencia a execução de outra MV. Para comparar a eficiência, são medidos os tempos de execução de algumas aplicações quando executadas em cada tecnologia, a quantidade de memória usada por cada MV, a escalabilidade, entre outros. Em alguns casos a comparação das tecnologias podem envolver também a eficiência da migração.

O objetivo deste trabalho é comparar a eficiência de duas tecnologias que implementam técnicas de virtualização distintas. Diferente do que foi realizado em outros trabalhos, será comparado o tempo total de execução de aplicações que estão executando sobre MVs, sendo que estas MVs podem migrar entre as máquinas físicas de um *cluster*. Estas migrações ocorrem com a finalidade de balancear a carga entre as máquinas físicas do sistema e são determinadas por um algoritmo de balanceamento.

Para realizar essas avaliações, foi construído um sistema formado por um *cluster* de máquinas físicas, máquinas virtuais executando sobre as máquinas físicas, aplicações executando sobre as MVs e um gerenciador de MV. Este último, que para tentar

manter a carga das máquinas físicas balanceadas, irá realizar migrações de máquinas virtuais entre as máquinas físicas do sistema. Esta subseção se dedica a explicar a parte do sistema que gerência as MVs, com destaque para o balanceamento de carga por ele este realizado.

3.1 Tecnologias de Paravirtualização e Containers

As duas tecnologias escolhidas para realização dos testes são o Xen e o OpenVZ. O Xen implementa a técnica de Paravirtualização, representando a classe de virtualização de sistemas. O OpenVZ implementa a técnica de virtualização por Containers, representando a classe de virtualização no nível de SO. O Xen e o OpenVZ foram escolhidos por estarem entre os representantes gratuitos mais eficientes de sua classe e também por serem tecnologias de código aberto. Nas subseções seguintes são comparados a eficiência, o isolamento e a gerência de recursos do OpenVZ e do Xen. Esses aspectos são importantes na análise dos resultados presentes na próxima seção.

3.1.1 Eficiência

O trabalho [22] mostra que operações básicas como criação de um processo, troca de contexto e acesso a memória, são mais custosas no Xen, enquanto no OpenVZ os tempos ficam próximos ao de um sistema sem virtualização.

O trabalho [21] conclui que o OpenVZ apresenta alto desempenho no acesso ao disco e na execução de aplicações científicas. Já o Xen demonstra excelente largura de banda, mas alta latência no uso da rede.

Além destes fatos, também podemos considerar que o aproveitamento da memória é mais eficiente no OpenVZ, já que, cada MV no Xen deve executar seu próprio *kernel*. Por isso, e pelos resultados de [19], o número de MVs que podem executar, simultaneamente é maior no OpenVZ e apresenta melhor escalabilidade do que o Xen.

3.1.2 Isolamento

Apesar de não ser consenso [19], por não compartilhar o mesmo *kernel*, o Xen proporciona maior isolamento entre as MVs. No OpenVZ, todas as máquinas virtuais estão dependentes do complexo *kernel* do Linux. Um erro neste *kernel*, pode afetar a execução de todas as MV. Já as MVs do Xen são dependentes do monitor de máquina virtual, cuja complexidade é menor do que a do *kernel* do sistema operacional usado no OpenVZ. Mas não são somente erros no *Kernel* ou no MMV que podem provocar falhas de isolamento, o gerenciamento dos recursos da MF também podem provocar

falhas no isolamento. É falha de isolamento uma MV usar o processador por mais tempo, em detrimento de outras MVs com a mesma prioridade. Testes realizados em [25] mostram que no OpenVZ, a execução de algumas aplicações sedentas por recursos em uma MV causavam impacto em outras MVs, caracterizando falha no isolamento. Já no Xen, sob o mesmo cenário de execução, a falha de isolamento não ocorreu.

3.1.3 Gerencia de Recursos

Gerencia de Processamento

OpenVZ O OpenVZ usa, em dois níveis, o algoritmo de escalonamento Fair-Share para decidir como o processador será alocado. No primeiro nível, o escalonador é usado para decidir qual MV terá o processador. No segundo nível, será decidido qual processo da MV escolhida fará uso do processador.

Existem diferentes implementações do algoritmo Fair-Share, mas a base de todas elas é a atribuição de pesos para as MVs e para os processos. No OpenVZ, o tempo de alocação do processador para as MVs é proporcional ao peso atribuído a uma MV. Assim, se uma MV **A** receber peso 1000 e uma MV **B** receber peso 500, **A** executará duas vezes mais do que B quando houver concorrência pelo recurso CPU.

Xen No Xen, existem disponíveis dois tipos de escalonadores de CPU; são eles: SEDF e o CreditScheduler [27]. O SEDF é um escalonador preemptivo que utiliza algoritmos de tempo real para garantir tempos de execução previamente determinados. O Credit Sheduler, escalonador padrão do Xen 3.0, é baseado em créditos e segue o modelo *Fair-Share*. No Credit Scheduler, cada MV recebe um crédito ou um peso. O crédito atribuído a uma MV, determina quanto uma MV terá de tempo de CPU quando houver concorrência por esse recurso. Assim, se uma MV **A** recebe 50 e uma MV **B** recebe 100 de crédito, em um período de concorrência por CPU, B irá executar 2 vezes mais do que A. Cada vez que uma VCPU executa, parte de sua quantidade de créditos é recalculada. Se a quantidade de créditos de uma VCPU for negativa, ela será classificada como *over*, quando os créditos são positivos, essa VCPU é classificada como *under*. Cada processador tem uma fila de VCPUs para gerenciar, essa fila é ordenada pela classificação da VCPU, VCPUS com classificação *under* sempre estará a frente dos VCPUS com classificação *over*. Sempre que uma VCPU chegar a fila de um processador, ela será colocada atrás das VCPUs que possuem classificação igual a sua.

Gerência de Memória

OpenVZ A alocação de memória para as MVs do OpenVZ se dá de forma flexível. O sistema pode compartilhar parte da memória usada pelas MVs. Existem dois parâmetros importantes para a alocação de memória de uma MV no OpenVZ. Um deles, o *privvmpages*, determina o limite máximo de memória que pode ser alocado para a MV, o outro, *vmguarpages*, diz qual é a quantidade garantida de memória que a MV tem a sua disposição. Desta forma, uma MV pode usar uma quantidade de memória maior do que a memória reservada para ele, desde que essa quantidade seja menor do que o valor estabelecido em *privvmpages*. Conforme foi notado nos experimentos, essa abordagem otimiza o uso deste recurso.

Xen Se por um lado, a alocação de memória para as MVs do Xen é menos flexível, as MVs do Xen têm autonomia de gerenciar a memória que foi alocada a ela. Quando o monitor de máquina virtual cria o ambiente da MV, é como se a MV estivesse recebido parte da RAM da MF. A máquina virtual tem que usar essa memória para alocar seu SO e suas aplicações. Em momento algum, a MV vai poder utilizar mais memória do que foi previamente configurado, a não ser que faça *swap*. Mesmo que uma MV não esteja ocupando toda memória alocada, nenhuma outra MV poderá utilizar essa memória.

Gerência Entrada e Saída

OpenVZ A alocação deste recurso ocorre em dois níveis, de maneira semelhante a alocação da CPU. A diferença é que no segundo nível é usado o escalonador CFQ (*Completely Fair Queuing*). Cada MV possui uma prioridade, e o escalonador distribui a banda de *IO* de acordo com as prioridades das MVs.

Xen O Xen dispões de MV especial com mais privilégios que as outras máquinas virtuais. Esta MV é chamada de domínio 0 e todas as requisições de E/S são centralizadas nela. Como a gerência dos dispositivos de E/S é feita por uma camada de software que fica entre estes dispositivos, as MVs e o Domínio 0, para realizar operações de E/S deve ocorrer transferência vertical de dados, ou seja, transferência de dados entre as camadas. Buscando uma transferência eficiente, com baixo *overhead*, o Xen implementa o mecanismo de anel de descritores de arquivo, cuja estrutura, é mostrada na Figura 3.1. O anel consiste em uma fila circular de descritores alocados pela MV e acessados por meio do Xen. Os descritores não contêm diretamente os dados de E/S, mas são referências indiretas para *buffers* que contêm os dados de E/S, e estes, por sua vez, são alocados de maneira separada pela SO da MV em questão. O acesso a

cada anel, se baseia em dois pares de ponteiros produtor/consumidor. Sempre que uma MV faz uma requisição, ela requisição é gravada no anel e o ponteiro produtor de requisição é incrementado, o Xen então, remove uma requisição quando incrementa o ponteiro consumidor de requisição deste anel. As respostas às requisições, são colocadas no anel de maneira semelhante, só que neste caso, o Xen passa a ser o produtor, enquanto a MV será a consumidora. Este mecanismo de anel é suficientemente genérico para suportar os mecanimos de E/S dos dispositivos.

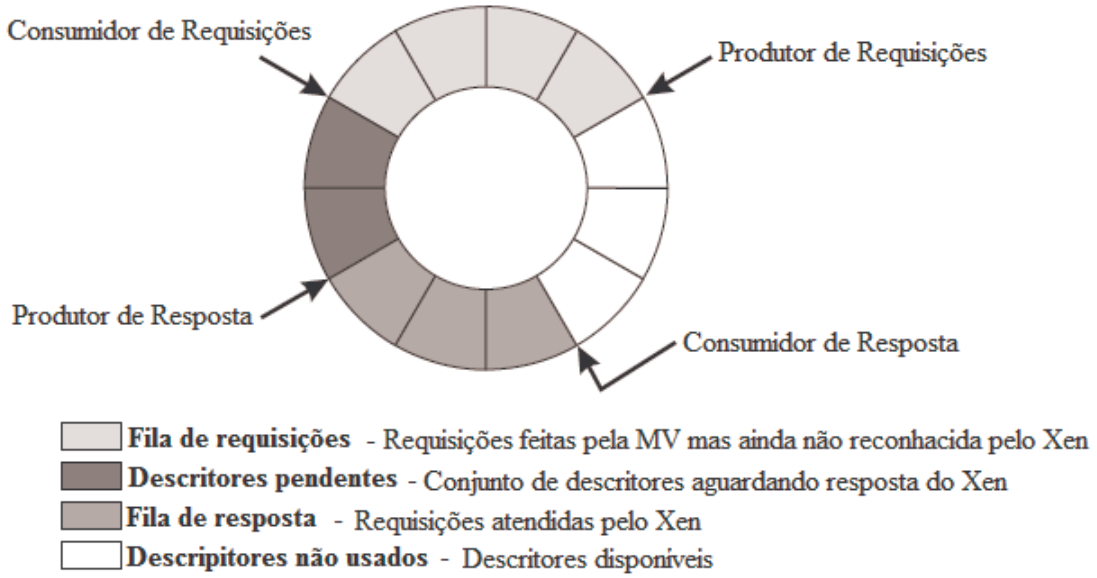

Figura 3.1: Estrutura do anel de E/S, o qual é usado para transferir dados entre o Xen e as máquinas virtuais hospedadas.

3.1.4 Migração

Por utilizarem técnicas bastante distintas na virtualização, enquanto o Xen utiliza paravirtualização e o OpenVZ usa containers, a migração das MVs pelo Xen também ocorre de maneira bastante distinta da migração utilizada pelo OpenVZ. No capítulo passado, que aborda os fundamentos de virtualização, foram apresentados alguns métodos genéricos de migração. Abaixo, é apresentado de maneira mais específica, como o OpenVZ e o Xen implementam a migração *online* [1] de MV.

[1] A migração de uma MV é chamada de *online* quando o *downtime* não é percebido pela aplicação, ou seja, o *downtime* estando na casa dos milisegundos.

Migração no Xen

A forma de migração utilizada pelo Xen é a pré-cópia. Como também já apresentado, nesta forma de migração o tempo downtime é baixo mas o tempo total de migração é alto. Neste modelo de migração, a MV pode ser vista como área de memória e isso possibilita o uso da pré-cópia como forma de migração. No artigo [18] é relatado o uso bem sucedido deste modelo para a migração de VMs executando aplicações de alta interatividade, no caso, um jogo interativo.

Migração no OpenVZ

A migração no OpenVZ ocorre na forma pára e copia. E como já apresentado, é uma forma simples de migração que resulta em um baixo tempo de migração e alto *downtime*.

Em pesquisa que antecede esta trabalho, foi avaliado a migração de MVs do OpenVZ, esta migração acorreu quando a MV executava no servidor de um jogo interativo; os resultados deste experimentos nos levou a avaliar como ocorria a migração e se era possível melhorá-la. Os resultados desta pesquisa podem ser conferido no apêndice A desta dissertação. A principal conclusão desta pesquisa foi que o alto tempo de *downtime* na migração das MVs do OpenVZ foi inaceitável para este tipo de aplicação. Quanto ao esforço em melhorar a migração no OpenVZ, diminuindo o *downtime*, esbarra na estrutura da técnica de virtualização utilizada. Na virtualização por containers, a MV não pode ser vista apenas como uma região de memória. Os processos da MV fazem parte de um *kernel* que não vai ser migrado, após a migração da MV, os processos devem ser recriados no *kernel* da MF de destino. Aplicar a pre-cópia no OpenVZ é inviável para a estrutura atual da técnica de virtualização, pois uma modificação na máquina física de origem pode implicar em algumas modificações na estrutura do *Kernel*. Isso impossibilita o uso de fases iterativas de migração como é utilizada na forma de migração por pré-cópia. Dessa forma, melhorar a migração no OpenVZ pode implicar em alterar também sua forma de virtualização. Um ponto positivo na migração do OpenVZ está na sua simplicidade, fazendo com que sua execução necessite de pouco processamento, como foi observado nos experimentos realizados no trabalho apresentado no apêndice A.

Enquanto a migração do Xen é mais complexa, apresentando baixo *downtime* e alto tempo de migração total, a migração do OpenVZ é mais simples, com alto *downtime* e baixo tempo total de migração. Este é um dos fatores que estimula os testes feitos nesta trabalho.

Para aplicações não interativas, como é a grande maioria das aplicações que utilizam *clusters*, pretende-se comparar o *overhead* gerado pela virtualização. E parte deste *overhead* está relacionado a migração das MVs. Em um cenário onde temos

um *cluster* de máquinas físicas contendo MVs, e onde estas MVs podem migrar, realizando balanceamento de carga entre as máquinas físicas, o *downtime*, o tempo de migração e a complexidade da migração, podem influenciar no tempo total de execução das aplicações das MVs do sistema. Enquanto o *dowtime* afeta diretamente a execução da MV que está sendo migrada, o tempo total de migração e a complexidade da migração podem exigir computação extra, afetando o desempenho de outras MVs. Este é um fator importante que é considerado nos testes aqui realizados.

3.2 Gerenciador de Máquinas Virtuais

Para executar os experimentos é necessário um gerenciador de máquinas virtuais. Este gerenciador tem, por função, manter a carga gerada pelas MVs balanceada entre as máquinas físicas do sistema. Para cumprir sua função, o gerenciador deve coletar informações sobre a carga de cpu das máquinas físicas e virtuais que compõem o ambiente experimental, e baseado no conjunto de informações que recebeu, decidir se devem haver migrações de MVs para que o sistema seja balanceado. Caso exista a necessidade de migrar MVs, o gerenciador realiza a migração das MVs, provavelmente, tornando o sistema mais balanceado do que estava antes da migração. Existem balanceadores que tratam a qualidade de serviço das aplicações, como a avaliação da qualidade do balanceamento não está no escopo desta dissertação essa funcionalidade não será implementada no balanceador.

Como na época não havia nenhum balanceador gratuito, capaz de atender as necessidades acima descritas para o Xen e para o OpenVZ, foi desenvolvido, nesta dissertação, um gerenciador de MVs baseado no Usher [15], [17]. O Usher foi escolhido por ser de código aberto, modular, e por já possuir algumas funcionalidades que seriam básicas para efetuar o gerenciamento das MVs nos exeperimentos. Outro fato influente, é que o Usher está totalmente implementado em Python, que é uma linguagem orientada a objeto e já conhecida. Dentre as principais modificações feitas, pode-se citar a nova capacidade de gerenciar MVs do OpenVZ, a codificação de um algoritmo responsável por balancear a carga do sistema e a adequação para que o gerenciador funcionasse de forma automática, não necessitando de intervenção humana para realizar a migração. O gerenciador criado herdou todas essas características, inclusive também está totalmente escrito em Python. A arquitetura do balanceador é mostrada na Figura 3.2.

Abaixo, são apresentados os módulos do balanceador utilizado.

Controlador Local Em cada máquina física existe um controlador local. É sua função, fornecer ao controlador central as informações de sua máquina física e também das MVs que no momento estejam executando sobre sua máquina

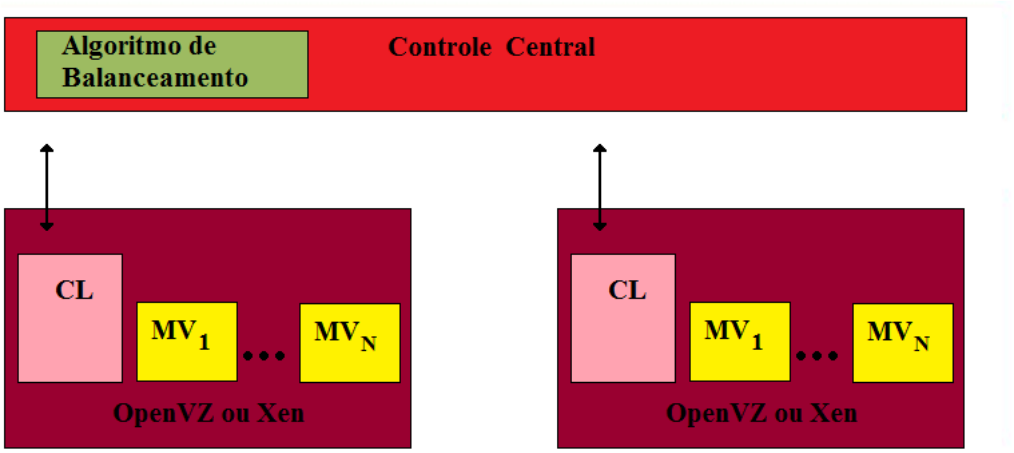

Figura 3.2: Arquitetura do balanceador de carga do sistema. CL representa o Controle Local.

física. Os controladores locais também executam a migração de suas MVs quando ordenados pelo controlador central.

Controlador Central O controlador central é o responsável por receber as informações dos controladores locais e informá-los sobre uma possível necessidade de migração. As informações recebidas dos controladores locais são direcionadas para o balanceador. Caso exista alguma necessidade de migração, o balanceador informa ao controlador central, e este por sua vez, informa ao controlador local qual a MV que deve ser migrada e qual é a máquina física de destino. É necessário somente um controlador para todo o sistema.

Balanceador É o modulo que contém o algoritmo de balanceamento de carga que analisa se o sistema está desbalanceado. Sempre que é aferido desbalanceamento no sistema, ele decide qual MV deve ser migrada e para qual máquina física esta MV deve ir. O balanceador tem como tarefa, realizar a interpretação de todos os dados coletados, e de acordo com essa interpretação, ordenar ou não, alguma ação com a finalidade de balancear a carga entre as MF do sistema. Na próxima seção é apresentado, com mais detalhes, o algoritmo de balanceamento usado.

O gerenciador tem potencial para fornecer as seguintes informações ao algoritmo de balanceamento: quantidade de memória usada por cada MV, porcentagem de CPU usada pelas MVs e pela máquina física, quantidade de dados trafegados por cada MV na interface de rede e a localização atual de cada uma MV. Mas devido a restrição na análise feita pelo algoritmo de balanceamento empregado, este só recebe informações sobre o uso de CPU das máquinas físicas e virtuais e a localização das

MVs. Portanto, as decisões de migração das MVs são decididas a partir da avaliação da carga de CPU das máquinas virtuais e físicas e da localização das MVs.

O apêndice B, apresenta detalhes de implementação e instruções de instalação e uso do gerenciador de máquina virtual implementado nesta dissertação.

3.3 Algoritmo de Balanceamento de Carga

O balanceamento de carga é uma técnica aplicada para distribuir a carga de trabalho entre dois ou mais servidores, enlaces de rede, CPU, ou outros recursos; a fim de otimizar a utilização destes recursos, maximizar o desempenho e evitar sobrecarga. Em geral o balanceamento de carga consiste em três fases [14]. A primeira, consiste na coleta de informações. A segunda, busca determinar qual seria a distribuição ótima para o estado em que o sistema se encontra. É na terceira fase que a ação de balanceamento é executada. O balanceador de carga pode ter uma das seguintes classificações:

Estático A regra de balanceamento é definida uma única vez, baseada em informações estáticas do sistema ou aplicações.

Dinâmico A regra de balanceamento é modificada em resposta ao estado atual do sistema ou das aplicações. O estado do sistema será constantemente atualizado e as decisões tomadas são baseadas nas informações atuais e possivelmente, dos estados anteriores do sistema.

Ao balancear a carga, tenta-se evitar que no sistema, existam simultaneamente máquinas com recursos subutilizados e máquinas com recursos superutilizados. Na maioria dos casos, realizar balanceamento de carga ótimo é impraticável, devido a complexidade computacional para resolver o problema. Conforme [4], o problema de balanceamento de carga é similar a problemas de alta complexidade computacional, que não podem ser resolvido de maneira exata em tempo polinomial. Para contornar esta limitação, heurísticas [2] ou algoritmos de aproximação [3], são usados para apresentarem soluções aproximadas do balanceamento de carga ótimo.

Antes de apresentar o algoritmo escolhido, cabe explicar sua relevância neste trabalho e o motivo de sua escolha de qual seria o algoritmo implementado. Como o algoritmo de balanceamento é o mesmo na realização de todos os testes, este não deve influenciar significativamente os resultados relativos, do Xen com os do OpenVZ. De fato, diferentes balanceadores, ou até mesmo uma modificação nos parâmetros de

[2] Algoritmos que produzem respostas aproximadas para algum problema.

[3] Algoritmos que produzem resposta aproximadas para problemas, sendo que a diferença entre a resposta do algoritmo e a resposta ótima, é sempre menor que um limite definido.

um balanceador, podem modificar, por exemplo, o número de migrações realizadas. E como a migração pode ser mais eficiente para uma das técnicas, o número de migrações pode influenciar no tempo de execução das aplicações.

Diante destas considerações, foram estudados os seguintes algoritmos de balanceamento de carga antes da escolha de qual seria implementado. Todos eles pertencem a classe de balanceadores dinâmicos. Os algoritmos estudados estão:

Algoritmo apresentado em [12] As decisões sobre a migração de MVs são tomadas localmente por cada máquina física. Todas as máquinas físicas têm informações sobre todas as máquinas do sistema. A avaliação da carga de cada máquina é baseada no tamanho da fila da CPU, da utilização da CPU, da utilização da memória e da rede.

Autonomous Learning [11] Neste algoritmo, a decisão de migração é baseada em um limiar dinâmico. Com o decorrer das migrações, o algoritmo vai aprendendo e variando o limiar de acordo com o que foi observado por ele no histórico da sua execução. A avaliação da carga de cada máquina física só considera a utilização de CPU.

Self-Training [10] É um algoritmo centralizado, onde as decisões de migração são tomadas por um controlador central e este pondera o uso de CPU, rede e memória. Ele se baseia no resultado desta ponderação para tomar suas decisões sobre migração das MVs. Um fator também determinante na escolha da migração é o SLA [4] das MVs.

Balack-box and Gray-box [3] Wood e outros [3], desenvolveram dois algoritmos para resolver o problema de balanceamento. Um deles, o Black-Box, estima a carga da máquina, baseado no carga de CPU, rede e quantidade de *swap* realizado na máquina. Já o outro algoritmo, o Gray-Box, se baseia em mais parâmetros além da carga da CPU, para avaliar a carga da máquina. Os parâmetros adicionais são específicos de aplicações que executam nas MVs. Segundo os autores, o com estes parâmetros adicionais, pode-se garantir qualidade de serviço as aplicações. Neste caso, o motivo das migrações não estaria somente em balancear a carga do sistema, mas para que o SLA de uma determinada aplicação não fosse violado.

Assign-U Estendido [7] O Algoritmo escolhido, apresentado em [7], foi baseado no Assign-U. O Assign-U foi criado para direcionar tarefas para máquinas, e estas tarefas permaneciam nestas mesmas máquinas até o fim de sua execução. A extensão do Assign-U, prevê a realocação das tarefas durante suas execuções.

[4]SLA(Service Level Agreement), em português, acordo de nível de serviço, é um acordo que estabelece um nível mínimo de qualidade para determinado serviço.

Todas as ação deste algoritmo são baseadas em uma função não linear, que determina quando deve haver migração de tarefas, qual tarefa deve migrar e para onde deve migrar.

3.3.1 O Algoritmo de Balanceamento Implementado

O algoritmo que foi implementado e é apresentado no trabalho [7], é visto como uma versão mais elaborada do algoritmo Assign-U [6], pois aquele, promove remanejamento das tarefas, podendo migra-las após o início de sua execução. O algoritmo apresentado em [7] é um algoritmo de aproximação eficiente [9], criado para migrar tarefas, portanto, será adaptado para migrar MVs. O algoritmo recebe como entrada o percentual utilizado de CPU de cada máquina física, o percentual que cada MV usa de sua máquina física atual e qual é essa máquina física. Como saída, ele diz se o sistema está balanceado ou não. Quando o sistema está desbalanceado o algoritmo reporta a MV que deve ser migrada, a máquina física de origem e de destino, para que ocorra sua migração. A escolha deste algoritmo se justifica por ser de simples implementação, ter complexidade computacional polinomial e por ter uso facilmente adaptado para virtualização de sistemas, cujo elemento base de migração é uma MV.

A equação e a inequação, apresentadas a seguir, são determinantes para as ações tomadas por este algoritmo guloso [5].

- v - Variável que representa uma das máquinas virtuais do sistema.
- i e j - Variáveis que representam uma das máquinas físicas do sistema. Sempre, $\boldsymbol{i \neq j}$.
- $h_i(v)$ - Carga atual de CPU sobre a máquina $\boldsymbol{i}$, sem contar com a carga gerada pela MV $\boldsymbol{v}$.
- $p_i(v)$ - Carga que a MV $\boldsymbol{v}$ gera sobre a máquina $\boldsymbol{i}$.
- $l_i(v)$ - Carga atual de CPU sobre a máquina $\boldsymbol{i}$.

(i) Condição de equilíbrio:

$$a^{h_i(v)+p_i(v)} - a^{h_i(v)} \leq 2(a^{l_j(v)+p_j(v)} - a^{lj(v)})$$

(ii) Custo marginal:

$$H_i(v) = a^{l_i(v)+p_i(v)} - a^{l_i(v)}$$

[5] Algoritmo guloso, ou ganancioso, é uma técnica de algoritmos para resolver problemas de otimização, sempre realizando a escolha que parece ser a melhor no momento; fazendo uma escolha ótima local, na esperança de que esta escolha leve até a solução ótima global.

A inequação, apresentada em **(i)**, reflete a condição de equilíbrio que determina se o sistema está ou não balanceado. A máquina $\boldsymbol{i}$, representada no lado esquerdo da inequação, é a máquina que contém atualmente a MV $\boldsymbol{v}$. No lado direito, temos a máquina $\boldsymbol{j}$ com potencial de receber a MV $\boldsymbol{v}$. Enquanto a inequação é satisfeita, significa que o sistema se mantém balanceado. Caso a inequação não esteja satisfeita, implica que a máquina $\boldsymbol{i}$ está em desequilíbrio em relação a máquina j. Para equilibrá-la novamente, a MV $\boldsymbol{v}$, deve migrar para alguma máquina a ser determinada pela equação **(ii)**.

A equação, apresentada em **(ii)**, representa o custo marginal, que constitui a base de escolha da máquina de destino. A máquina de destino a ser escolhida, será a que possuir o menor custo marginal.

De maneira intuitiva, pode-se ver o lado esquerdo da inequação como uma quantificação da carga sobre o máquina $\boldsymbol{i}$, e o lado direito, a tentativa de prever como seria se a MV $\boldsymbol{v}$ estivesse executando na máquina $\boldsymbol{j}$. Se a carga total gerada em $\boldsymbol{j}$ for sensivelmente menor que a gerada em $\boldsymbol{i}$, indica que o sistema está em desequilíbrio e que para equilibrá-lo, a MV $\boldsymbol{v}$, que executa em $\boldsymbol{i}$, deve migrar para outra máquina.

O pseudo algoritmo, originado da condição de equilíbrio e do custo marginal, é apresentado no apêndice B.

Os produtos de virtualização, juntamente com o sistema de balanceamento apresentado neste capítulo, serão a base do ambiente experimental discutido no próximo capítulo.

Capítulo 4

Avaliação Experimental

Considerando que, toda avaliação proposta neste trabalho, se baseia em experimentos reais e não simulados, a etapa descrita neste capítulo foi a mais longa e trabalhosa desta dissertação. O ambiente experimental, formado pela combinação de diversos, e muitas vezes complexos *softwares*, que deveriam interagir entre si, fez com que a conclusão desta etapa se tornasse um desafio.

Outro fator desafiador, foi a construção da metodologia a ser utilizada. Existem trabalhos, como [10],[3],[2] e [42], que fazem comparações entre algoritmos de balanceamento. Existem outros, como [5],[20],[22] e [18], que comparam a migração de técnicas de virtualização. Nos experimentos apresentados neste capítulo, busca-se comparar a eficiência das técnicas de virtualização, quando suas MVs podem ser migradas, conforme determinado por um algoritmo de balanceamento. O caminho seguido foi baseado nas metodologias usadas nestes trabalhos, utilizando diferentes aplicações, variando as execuções; ora com sistema de balanceamento, ora sem sistema de balanceamento; ora com carga balanceada, ora com carga desbalanceada.

O OpenVZ e o Xen serão utilizados nos experimentos como representantes das técnicas de virtualização por container e para-virtualização, respectivamente. Mais detalhes sobre os objetivos dos experimentos, assim como, o ambiente experimental e os resultados, podem ser conferidos neste capítulo.

4.1 Objetivos dos Experimentos

O objetivo dos experimentos é comparar a influência do *overhead* de virtualização do sistema e do *overhead* de migração para duas técnicas de virtualização de sistemas. As comparações dos resultados dos experimentos, devem expor, além do *overhead* de virtualização, o custo envolvido na migração das MVs. Este custo da migração é composto pelo *overhead* de migração, que definimos como o tempo de CPU ocupado para migrar uma MV, e o *downtime* de migração, que é o tempo de inatividade da MV durante a migração. Enquanto o *downtime* influencia diretamente a máquina

migrada, o *overhead* de migração pode influenciar todas as MVs localizadas nas MFs de origem e destino da migração. O tempo de execução das aplicações deve refletir todos esses *overheads*. Os experimentos, apresentados neste capítulo, tem por objetivo expor esses *overheads* e a influência da migração no balanceamento de carga do sistema. Estes experimentos

4.2 Ambiente Experimental

Os testes foram realizados no LCP[1] e todos os softwares usados nos experimentos são gratuitos, de código aberto e sempre citados como referência em trabalhos científicos da área. Como o foco dos experimentos é comparar os resultados de duas técnicas de virtualização, procurou-se usar softwares o mais semelhante possível para as duas tecnologias de virtualização. As características dos softwares são representadas nas duas Tabelas a seguir, 4.1 e 4.2. A primeira Tabela, 4.1, mostra os softwares e suas versões, que foram utilizados no experimento realizados com o Openvz como tecnologia de virtualização.

Ambiente OpenVZ	
SO Hospedeiro	CentOS 5.2
Kernel Hospedeiro	2.6.18-128.1.1.el5.028stab062.3PAE
SO Hospedados (MVs)	CentOS 5.2
Kernel Hospedado	2.6.18-128.1.1.el5.028stab062.3PAE
Versão do OpenVZ	2.6.18
Versão do NFS	NFSv4

Tabela 4.1: Especificação de Software - OpenVZ.

A Tabela 4.2, apresenta a configuração dos softwares, quando a tecnologia de virtualização usada nos experimentos foi Xen.

Ambiente Xen	
Hospedeiro	CentOS 5.2
Kernel Hospedeiro	2.6.18-92.1.13.el5xen
SO Hospedados (MVs)	CentOS 5.2
Kernel Hospedado (MVs)	2.6.18-92.el5xen
Versão do Xen	3.2.1
Versão do NFS	NFSv4

Tabela 4.2: Especificação de Software - Xen.

A Tabela abaixo 4.3 apresenta as especificações de hardware usados nos experimentos.

[1]Laborotório de Computação Paralela da COPPE/UFRJ. Para mais detalhes sobre o LCP, veja no site www.lcp.coppe.ufrj.br.

Hardwares	
Processador	Intel(R) Core(TM)2 Quad CPU Q6600 @ 2.40GHz
Memória RAM	2GB
Rede (MVs)	100 Mbit/s
Disco	160 GB 7.200 RPM SATA

Tabela 4.3: Especificação do Hardware.

A Figura 4.1, mostra a arquitetura do ambiente experimental. Todas as máquinas estão interconectadas por meio de um switch, formando uma rede com as características apresentada Tabela 4.3. A máquina superior é usada somente para a gerência das MVs hospedadas nas outras máquinas físicas. Esta máquina executa o módulo CC (Controle Central) do gerenciador apresentado no capítulo anterior. As MFs (Máquinas físicas), localizadas na parte inferior da Figura, são as máquinas que hospedam todas as MVs envolvidas nos testes. Todas essas máquinas físicas possuem o módulo CL (Controle Local) do gerenciador. Todas as MFs com módulo CL são clientes e servidores NFS (Network file System) [2]. A instalação das MVs envolvidas nos testes são igualmente distribuídas pelas MFs e toda parte persistente em disco do SO das MVs é dividida entre as máquinas físicas. Por exemplo, se tivermos três MFs e nove MVs, cada MF receberá os arquivos persistentes de três MVs. A MF exporta, via NFS, toda área de disco de suas MVs, por outro lado, todas as outras MFs montam todos os diretórios exportados. Desta maneira, todas as MFs têm acesso aos arquivos de todas as MVs do sistema. O sistema foi configurado desta maneira para permitir que as MVs possam migrar de maneira mais eficiente, sem necessidade de levar para a máquina de destino toda a instalação do seu SO e todos os seus arquivos. Assim, uma MV pode estar executando na MF1 e estar acessando seus arquivos localizados fisicamente na MF2. O Xen já estava habilitado a operar a migração dessa maneira, mas o OpenVz teve seu *script* de migração levemente modificado para funcionar desta forma.

Conforme comentado no Capítulo 3, a gerência de memória se dá de maneira distinta para cada uma das técnicas de virtualização experimentadas. Por ser um quesito importante na comparação das técnicas, as configurações que definem a quantidade de memória usada pelas MVs tenta aproximar as condições das duas técnicas. Para as duas primeiras aplicações, foi configurado 100MB e para a a terceira 200MB. O tamanho da memória da MV é importante por influenciar diretamente na quantidade de dados trafegados na migração, e consequentemente, no tempo total de migração.

O intervalo entre as coletas de dados por parte do controle central é de 10 segundos. O intervalo é configurável no gerenciador. O tempo de 10 segundo foi escolhido

[2]Sistema de arquivo distribuído usado para compartilhar arquivos e diretórios entre sistemas interconectados, criando assim um diretório virtual

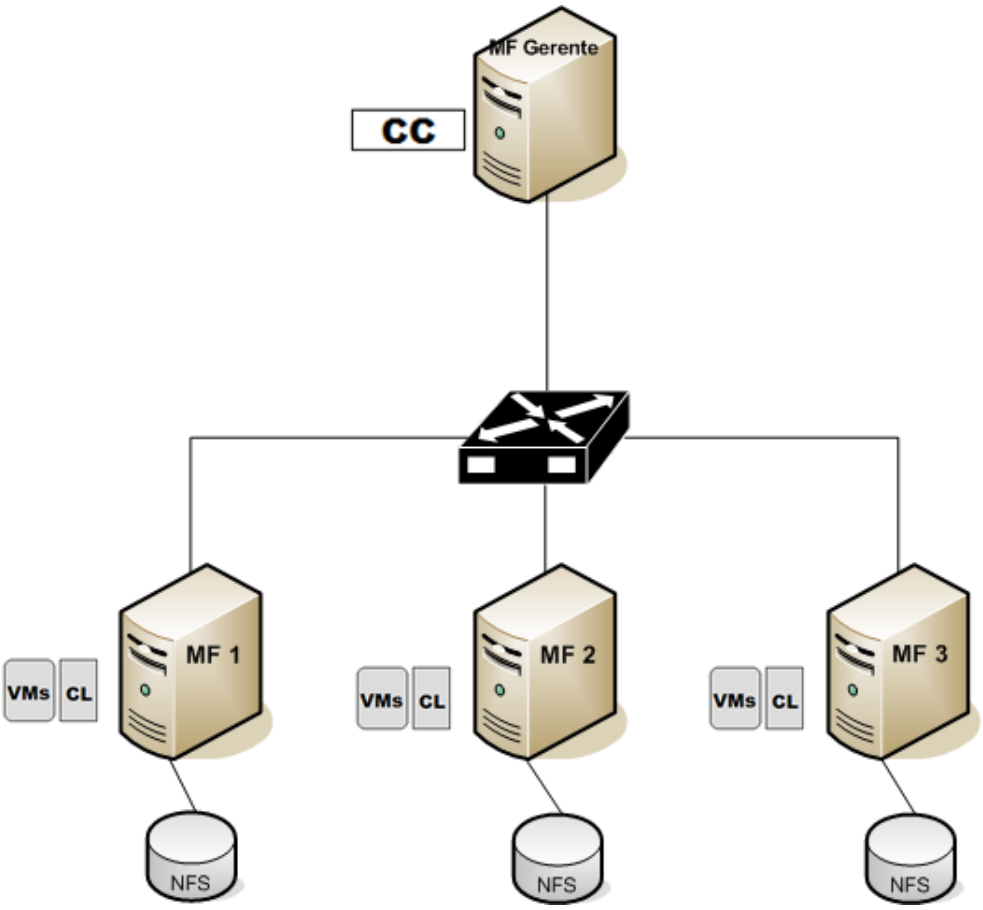

Figura 4.1: Sistema Experimental.

baseado no tempo total de migração das MVs. Para cada experimento, as aplicações são iniciadas simultaneamente.

4.3 Organização dos Experimentos

A apresentação dos experimentos e seus resultados está dividida em três seções. Cada seção apresenta os experimentos realizados com uma aplicação diferente. No Experimento1 e no Experimento2, é usada aplicação sintética e no Experimento3, uma aplicação real. Os testes realizados nos experimentos contemplam igualmente as duas tecnologias de virtualização em questão. Para cada aplicação, são avaliadas algumas variações quanto a distribuição do *workload* sobre as MF e quanto possibilidade de ativação do balanceador de carga do sistema. Em especial, para o experimento referente a compilação do *kernel*, são variados parâmetros de compilação da aplicação e capacidade da rede. Em todos os experimentos são usadas nove máquinas virtuais.

Enquanto as variações realizadas no Experimento3 são explicadas na Seção 4.8, que apresenta o Experimento3; as possíveis variações sobre a distribuição inicial do *workload* e a ativação do balanceamento de carga, são descritas abaixo:

Balanceado - Sem Migração Conforme a Figura 4.2, as MVs são igualmente distribuídas sobre as MF, cada MF recebendo 3 MV. Esta configuração permanece durante toda execução, pois não é realizada migração para balancear a carga do sistema.

Balanceado - Com Migração Nesta variação, as MVs iniciarão o experimento igualmente distribuídas sobre as MFs do sistema, conforme a Figura 4.2. Como é permitido a migração, ao final da execução, a distribuição das MVs em relação as máquinas físicas pode estar totalmente alterada.

Desbalanceado - Sem Migração Neste caso, o sistema fica desbalanceado, a distribuição das MVs ocorre como representado na Figura 4.3 por toda a execução do experimento. A MF1 recebe cerca de 78% do *workload*, o que corresponde a execução de 7 MVs, e a MF2 e MF3 recebem 11% cada uma, correspondendo a uma MV para a MF2 e uma para a MF3.

Desbalanceado - Com Migração Neste experimento, o balanceador é ativado e a disposição inicial das MVs também se dá conforme a Figura 4.3. Ao balancear a carga do sistema, o balanceador migra MVs de uma MF para outra, alterando a disposição inicial das MVs. Balanceando a carga, espera-se que melhore o tempo médio de execução da aplicação Sintética que executa nas MVs.

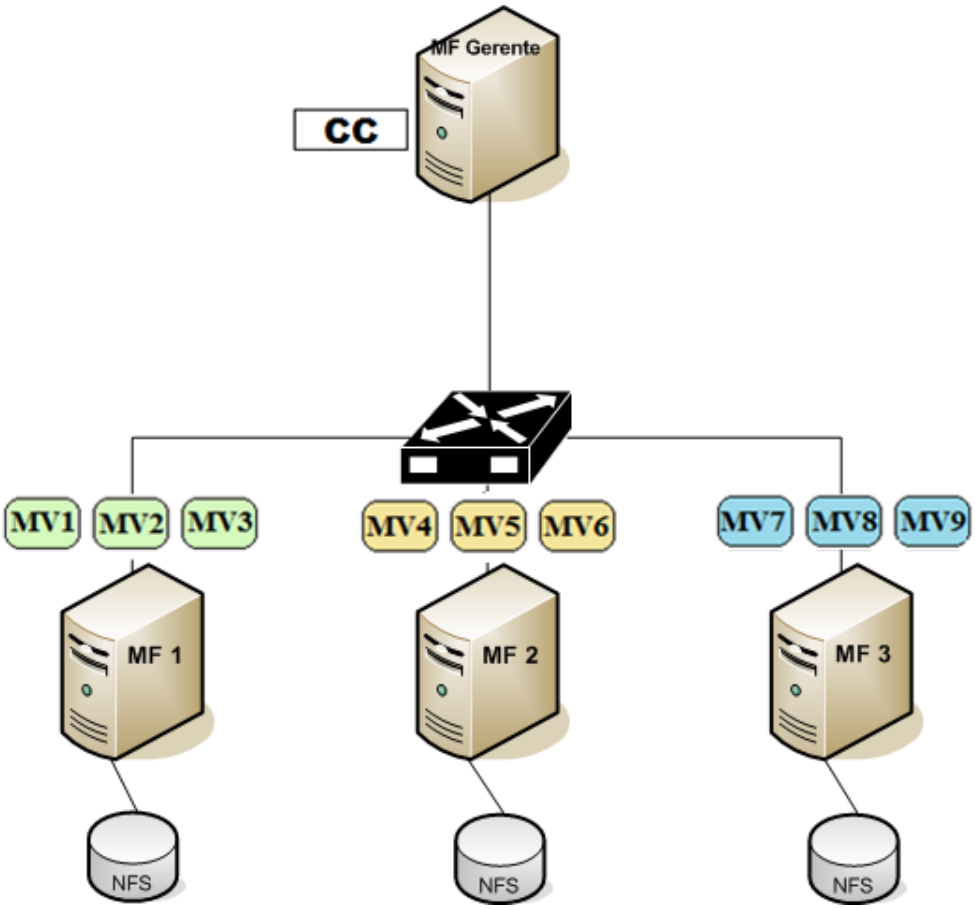

Figura 4.2: Configuração Balanceada

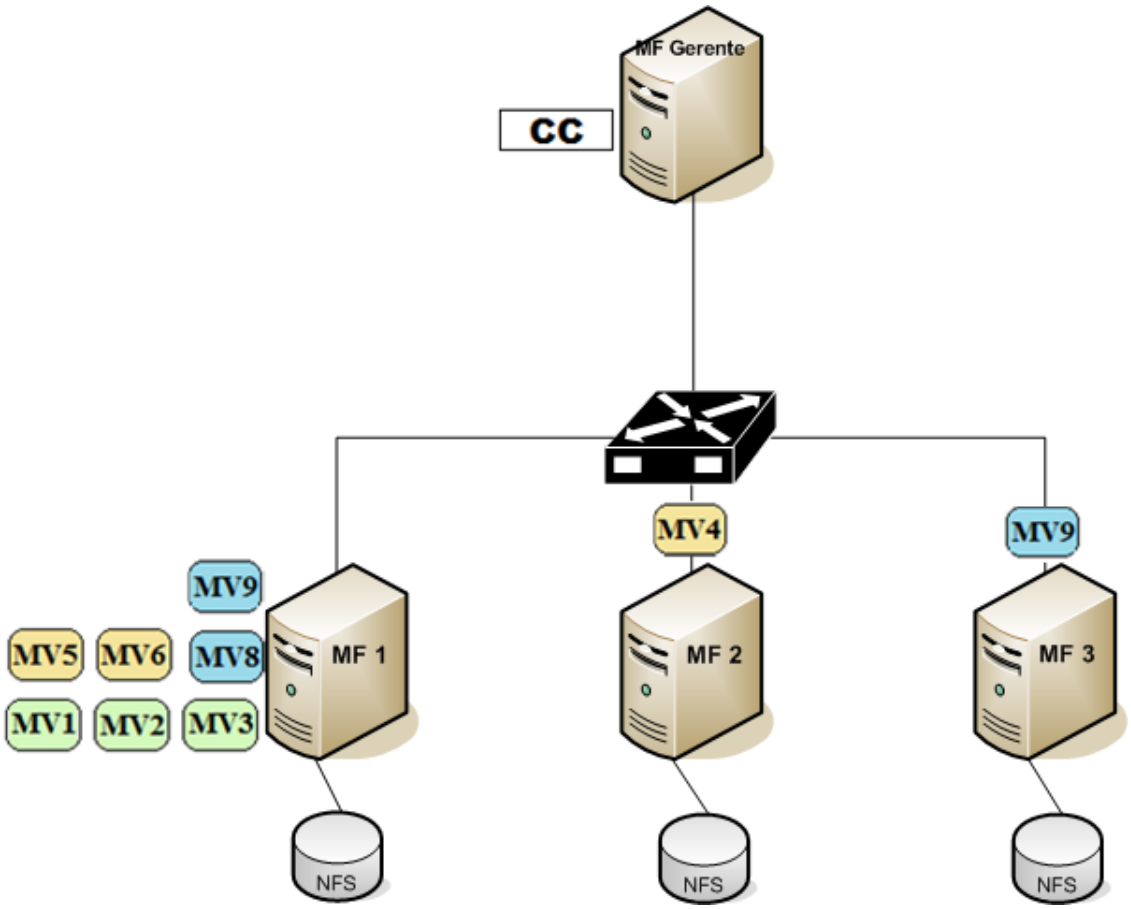

Figura 4.3: Configuração Desbalanceada.

4.4 Aplicações Utilizadas nos Experimentos

Todas as aplicações usadas nos experimentos têm perfil *CPU-Bound*, ou seja, seu desempenho está diretamente relacionado ao tempo de CPU alocado para a aplicação e a capacidade de processamento desta CPU. Em termos de desempenho, o processador deve ser visto como gargalo para quase todos os experimentos. A motivação de se usar aplicações *CPU-Bound* se deve ao objetivo de expor, além do *downtime*, o *overhead* de migração das técnicas. Ao executar aplicações *CPU-Bound* em todas as MVs, espera-se que tanto o *downtime* quanto o *overhead* de migração influenciem no tempo de execução desta MV. O mecanismo de migração e as aplicações das MVs estarão competindo pelo recurso CPU das MF.

Todas as variações realizadas nos experimentos, são executadas no mínimo sete vezes. O erro apresentado nos gráficos, reflete o desvio padrão destas execuções. Foram utilizadas nos experimentos, duas aplicações sintéticas e uma aplicação real, e por isso, as aplicações foram denominamos de Sintética1, Sintética-Randômica e *Kernel*. Nas subseções a seguir, são apresentadas mais características das aplicações usados nos experimentos:

4.4.1 Sintética

Baseado no projeto Stress [32], a função apresentada abaixo calcula por diversas vezes a raiz quadrada de um número randômico. As instruções desta função serão puramente matemáticas, requerendo uso intenso de processador. Esta aplicação foi utilizada no Experimento1.

```
int hogcpu (void)
{
 int a=0,c=0;
      while (1){
            if (a == 19000){
                 usleep(10);
                 a = 0;
                 c++;
                 if (c==18000)
                    return 0;
            }
      a++;
      sqrt (rand ());
      }
}
```

Listagem 4.1: Sintética1.c

4.4.2 Sintética-Randômica

A função apresentada na listagem 4.2, escrita na linguagem Python, executa com alguma probabilidade um determinado número de instâncias da aplicação Sintética. O *loop* principal é executado cinco vezes, e em cada iteração, um dos casos seguintes irá ocorrer: Com probabilidade de 0,2 é executado uma instância da aplicação Sintética. Com probabilidade 0,2 são executadas duas instâncias da aplicação Sintética. Com probabilidade 0,1 são executadas quatro instâncias da aplicação Sintética. Com probabilidade 0,5 a MV fica "dormindo"por 30 segundos. Diferente do que ocorre com a função Sintética pura, a função Sintética-Randômica apresenta uso não regular de CPU. Esta aplicação foi utilizada no Experimento2.

```
import sys
import os, random, time

seed =  sys.argv[1]
random.seed(seed)
for nexec in range(5):
            a = random.randint(0, 100)
            if a < 20:
                        os.system("sintetica1 --nthread 1");
                        continue;
            if a < 40:
                        os.system("sintetica1 --nthread 2");
                        continue;
            if a < 50:
                        os.system("sintetica1 --nthread 4");
                        continue;
            time.sleep(30);
            continue;
```

Listagem 4.2: Sintética-radomica.py

4.4.3 Kernel

Esta aplicação se refere a compilação do *kernel* do linux. É uma aplicação real, geralmente executada por desenvolvedores do *kernel*, utilizada em comparações de

modelos de virtualização [24] e faz parte de *benchmarks*. A compilação do *kernel* requer, além do uso intenso de CPU, muito acesso ao disco rígido. A versão do *kernel* compilada no experimento, foi a linux-2.6.12, assim como no trabalho [24], usando o comando "make -j8 bzImage". O parâmetro *-j8* indica a quantidade de *jobs* simultâneos que usados na compilação do *kernel*, recomenda-se 2 *jobs* para cada processador da máquina física, conforme [56]. Como cada MF tinha 4 processadores, foi usado o parâmetro *j8* no comando de compilação. Esta aplicação foi utilizada no Experimento3.

4.5 Validação Experimental do Gerenciador de Máquinas Virtuais

Antes de começar os experimentos propostos, foram executados testes experimentais de validação do gerenciador de máquinas virtuais implementado. Nesta seção é apresentado um dos testes de verificação. Neste teste, pode-se avaliar tanto o funcionamento do gerenciador de máquina virtual, quanto o algoritmo de balanceamento e toda interação dos softwares do sistema. Antes de chegar a versão final, esse teste foi útil na depuração de todos os *softwares* e sua integração. Na execução deste teste, foram usadas 4 MV e 3 MF. Cada máquina virtual executa uma aplicação, cuja execução está basicamente associada ao uso de CPU. Propositalmente, as aplicações começam a execução em tempos diferentes e podem variar a carga de CPU durante a execução.

A Figura 4.4, apresenta como foi o andamento do teste realizado. Esta Figura contêm três gráficos alinhados pelo eixo x. Cada gráfico está relacionado com uma MF. Em cada um destes gráficos, o eixo y representa o percentual do consumo de CPU da máquina física. No eixo x, tem-se o tempo decorrido em segundos. O consumo de CPU total da MF, é a soma do consumo de CPU atribuído as MVs que estão nesta MF. O consumo de cada MV é representado na Figura por uma cor, independente da MF na qual ela esteja, e mesmo quando a MV migra de MF, a cor correspondente ao consumo desta MV na MF de destino, será a mesma que era na origem.

De posse dessas informações, podemos analisar os gráficos da Figura 4.4. Nos primeiros 25 segundos, nenhuma MV executa alguma aplicação. No segundo 25, duas MVs, representadas pela cor roxa e verde, iniciam execução consumindo quase 20% de CPU de suas MFs. Neste momento, tem-se um desbalanceamento do sistema, mas de nada adiantaria migrar qualquer uma das MVs, pois o desbalanceamento iria permanecer. O balanceador "entende"este fato, e não executa nenhuma migração. No segundo 120, uma nova MV, representada pela cor amarela, inicia execução,

elevando o percentual de uso de CPU da sua MF para 40%. Imediatamente, o gerenciador nota o desbalanceamento e migra a MV amarela, rebalanceando o sistema. No segundo 225, a última MV, representada pela cor cinza, começa a execução que vai aumentando gradualmente o percentual de uso de CPU de sua MF. O gerenciador só realiza uma migração em resposta ao desbalanceamento causado por essa MV cinza, quando nota que pode balancear melhor o sistema migrando a MV verde, deixando o sistema melhor balanceado.

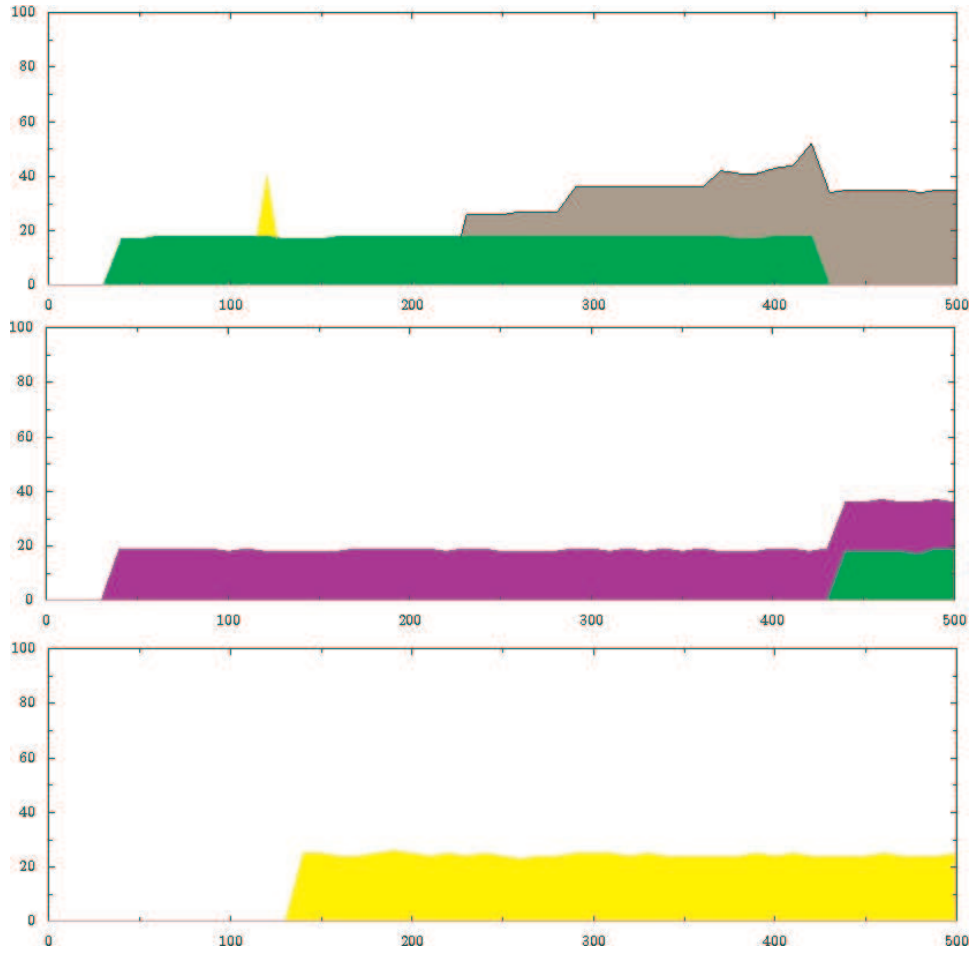

Figura 4.4: Gráfico do consumo total de CPU de três Máquinas Físicas. Cada cor representa carga gerada por uma máquina virtual.

4.6 Experimento1

Este experimento se baseia na execução da aplicação Sintética com uma variação na distribuição da carga inicial sobre as MFs do sistema. Para este experimento, todas as MVs do sistema executarão o mesmo trabalho, que consiste em duas instâncias da Aplicação Sintética1. Essas instâncias são executadas, simultaneamente em todas as MVs. Com as configurações utilizadas para este experimento, quando

duas instâncias da aplicação Sintética1 é executada em uma MF com a CPU não sobrecarregada, a aplicação ocupa cerca de 48% da CPU desta MF.

No experimento são usadas 3 MFs, e sobre elas, são distribuídas 9 MVs que executarão instâncias da aplicação Sintética. As variações executadas neste experimento, são: Balanceada - Sem Migração, Desbalanceada - Sem Migração e Desbalanceada - Com migração. As três subseções seguintes apresentam os resultados das variações deste experimento para os dois modelos de virtualização:

4.6.1 OpenVZ

A Figura 4.5, mostra os resultados obtidos para a execução da aplicação Sintética1, nos diferentes cenários, usando o OpenVZ como técnica de virtualização. Conforme indicado na legenda da Figura 4.5, o eixo x, identifica as MVs, o eixo y, se refere ao tempo de execução da aplicação que é executada na MV e a cor identifica a forma de execução. Por exemplo, para a execução Balanceada, a aplicação da máquina virtual 1 demorou cerca de 113 segundos.

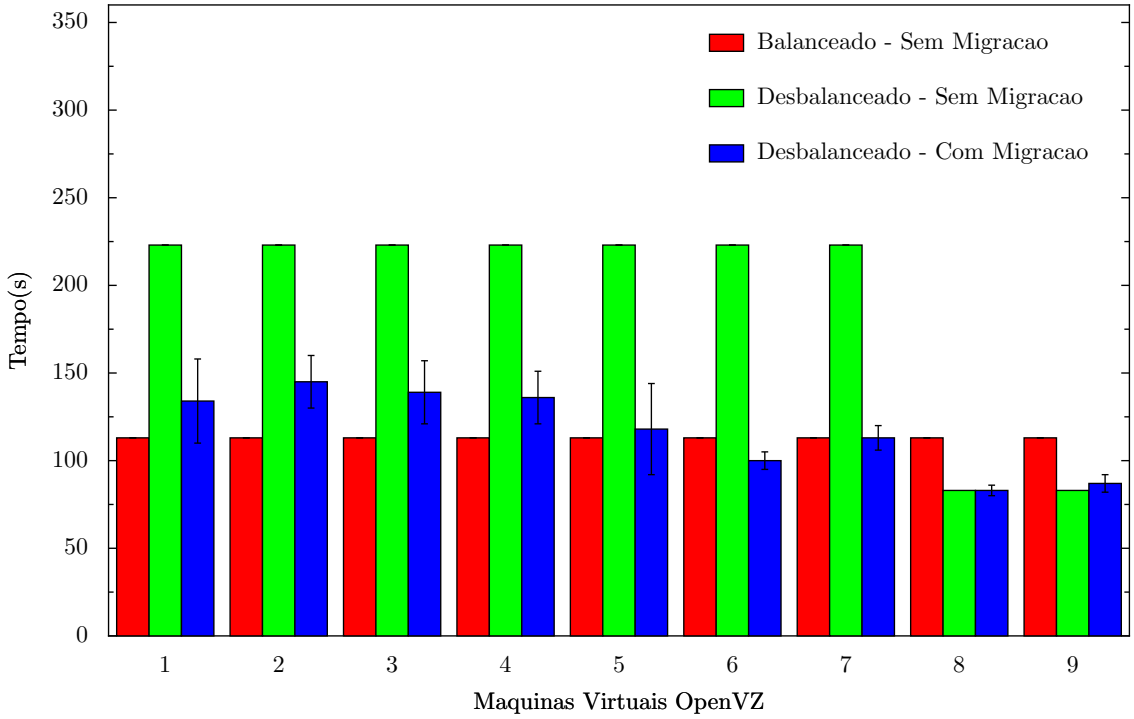

Figura 4.5: Experimento1, OpenVZ. Tempo de execução das MVs.

Quando a configuração inicial do sistema é desbalanceado, representado nos gráficos de resultados pelas cores vermelha e verde, as MVs que apresentam maiores tempos de execução, as MVs 1 a 7, são que inicialmente estavam localizadas na MF1, conforme indicado na Figura 4.3. Este tempo de execução é explicado pela sobrecarga do recurso CPU, provocada pelo grande número de MVs. Quando a

configuração inicial é Balanceada, representado pela cor azul, as MVs, e consequentemente, o *workload*, está igualmente distribuído sobre as MF do sistema. Como era de se esperar, o balanceamento da carga melhorou o tempo de execução das aplicações.

4.6.2 Xen

A Figura 4.6, mostra os resultados obtidos na execução da aplicação Sintética1, nos diferentes cenários, usando o Xen como técnica de virtualização. A apresentação gráfica ocorre da mesma maneira que na Figura anterior 4.5.

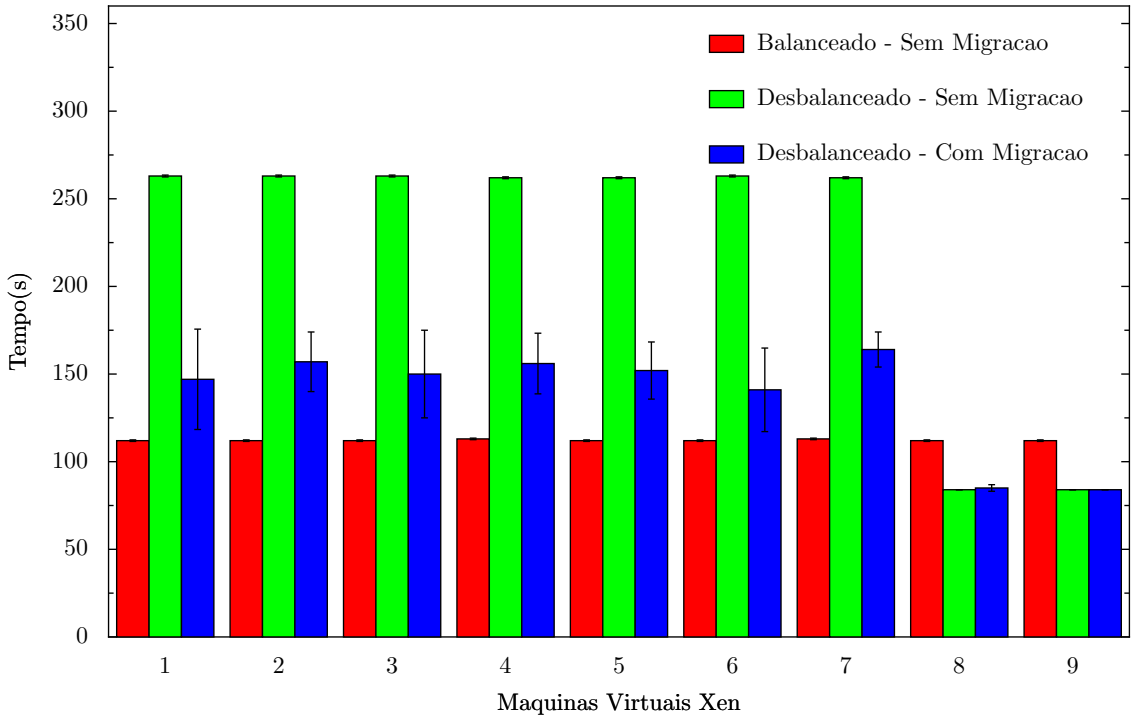

Figura 4.6: Experimento1, XM. Tempo de execução das MVs.

4.6.3 Comparação OpenVZ x Xen

A Figura 4.7, relaciona os resultados do OpenVZ e do Xen quando a configuração do sistema é Desbalanceado - Com Migração. Conforme os resultados apresentados na Figura 4.7, em geral, as MVs do OpenVZ concluíram a execução antes das MVs do Xen. Esses resultados podem ser atribuídos a dois fatores principais. O primeiro, pode ser notado comparando o tempo de execução das MVs, quando estas executavam sobre alguma MF sobrecarregada. Conforme mostrado na Figura 4.7, o Xen não teve boa escalabilidade para executar aplicações puramente *CPU-Bound*. Quando o sistema estava desbalanceado e havia sete MVs executando aplicações

CPU-Bond sobre uma MF, o tempo de execução das MVs do OpenVZ foi menor que o tempo de execução das MVs do Xen. O outro fator, está relacionado com o tempo de migração das MVs. Quanto menor o tempo de migração das máquinas virtuais, mais rápido será o balanceamento de carga, e quanto mais balanceada a carga, melhor será o *througput* do sistema. A velocidade da migração refletiu nos resultados apresentados. Note que para a MV 9 da Figura 4.5, para o sistema desbalanceado (cores verde e vermelho), o tempo de execução foi maior quando houve balanceamento de carga. Isso representa o impacto da primeira MV que foi migrada para balancear a carga, pois, esta passou a disputar recursos de CPU com a MV1. Como pode ser observado na Figura 4.6, o impacto equivalente é quase insignificante para o Xen.

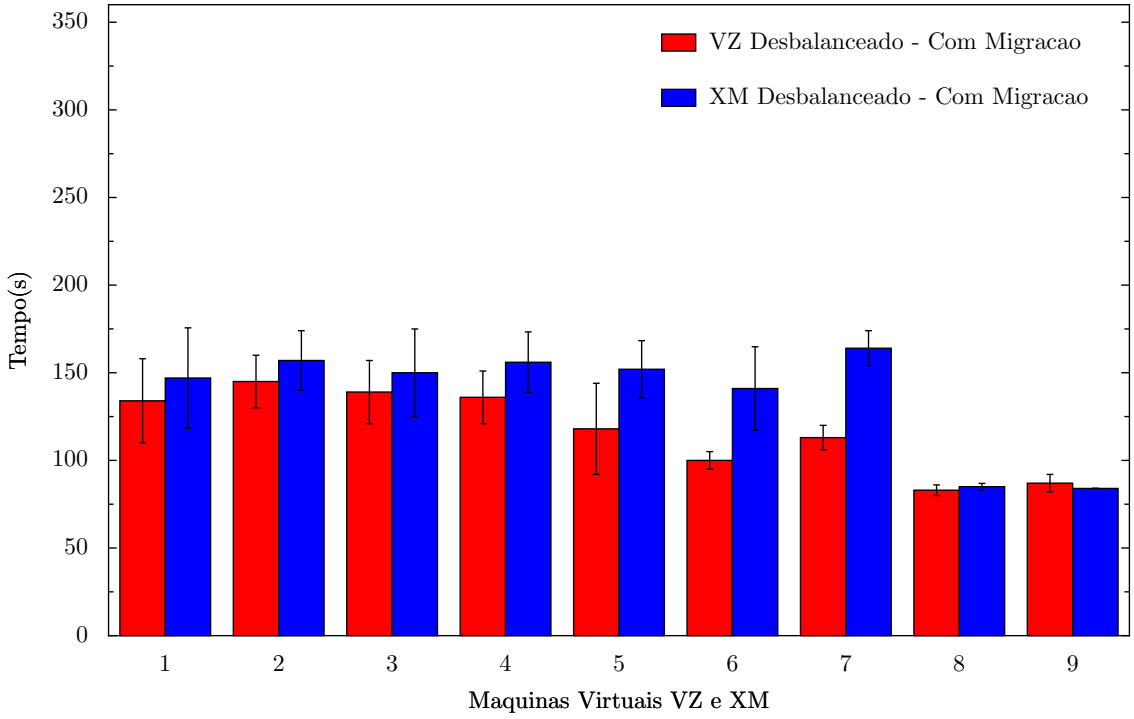

Figura 4.7: Experimento1, OpenVz x Xen. Tempo de execução das MV.

A Figura 4.8, mostra a média do tempo de execução das aplicações das MVs, isto é, o tempo médio que as MVs do sistema demoraram para realizar sua execução. O eixo y corresponde ao tempo médio de execução, já o eixo x, a tecnologia de virtualização avaliada. Com o sistema desbalanceado, realizar o balanceamento de carga melhorou igualmente o tempo de execução das MVs. Tanto para o OpenVZ, quanto para o Xen, o tempo melhorou em 39%.

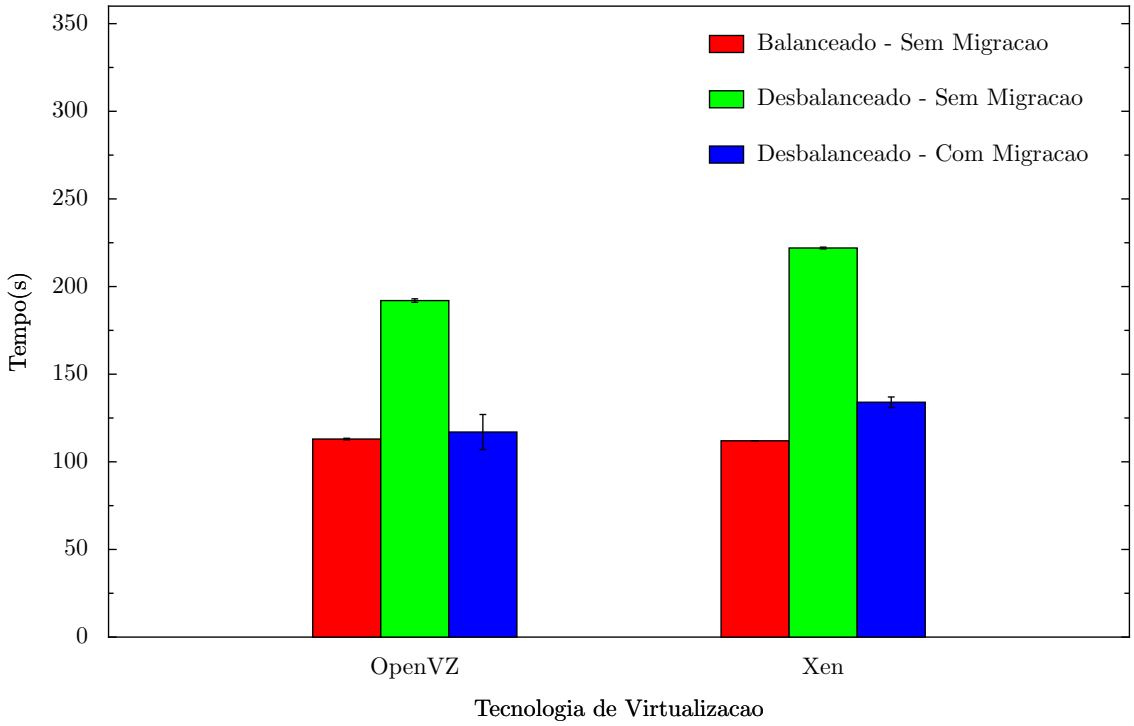

Figura 4.8: Experimento1, OpenVz x Xen. Tempo de execução médio das MV.

4.7 Experimento2

No Experimento2, a aplicação utilizada como base é a Sintética-Randômica. Neste experimento, são usadas 3 MFs e sobre elas são distribuídas 9 MVs executando a aplicação Sintética-Randômica. Todas as 9 MVs do sistema iniciarão simultaneamente a execução desta aplicação, mas isso não significa que o trabalho gerado na execução da aplicação estará igualmente distribuído sobre as MVs. As variações neste experimento se restringem a: Balanceado - Sem Migração e Balanceado com migração. As características desta aplicação dificultam o trabalho do balanceador. Durante o experimento, ocorrem mudanças aleatórias da carga gerada por uma MV, o que pode influenciar o balanceador a realizar migrações desnecessárias. Neste experimento é avaliado se essas migrações, geralmente desnecessárias, irão prejudicar o tempo de execução total das MVs.

As duas subseções seguintes apresentam os resultados das variações deste experimento para os dois modelos de virtualização.

4.7.1 OpenVZ

A Figura 4.9 mostra que o balanceamento de carga praticamente não alterou os tempos execução. Isso indica que mesmo com algumas migrações desnecessárias, o sistema não teve seu tempo de execução prejudicado. As migrações não prejudica-

ram o tempo de execução das aplicações justamente pela característica da aplicação. Por ser puramente *CPU-Bound* e por não ter sobrecarregado as MFs, a aplicação não prejudicou nem o *downtime*, nem o tempo total de migração. Como resultado, as migrações não prejudicaram o tempo de execução das MVs.

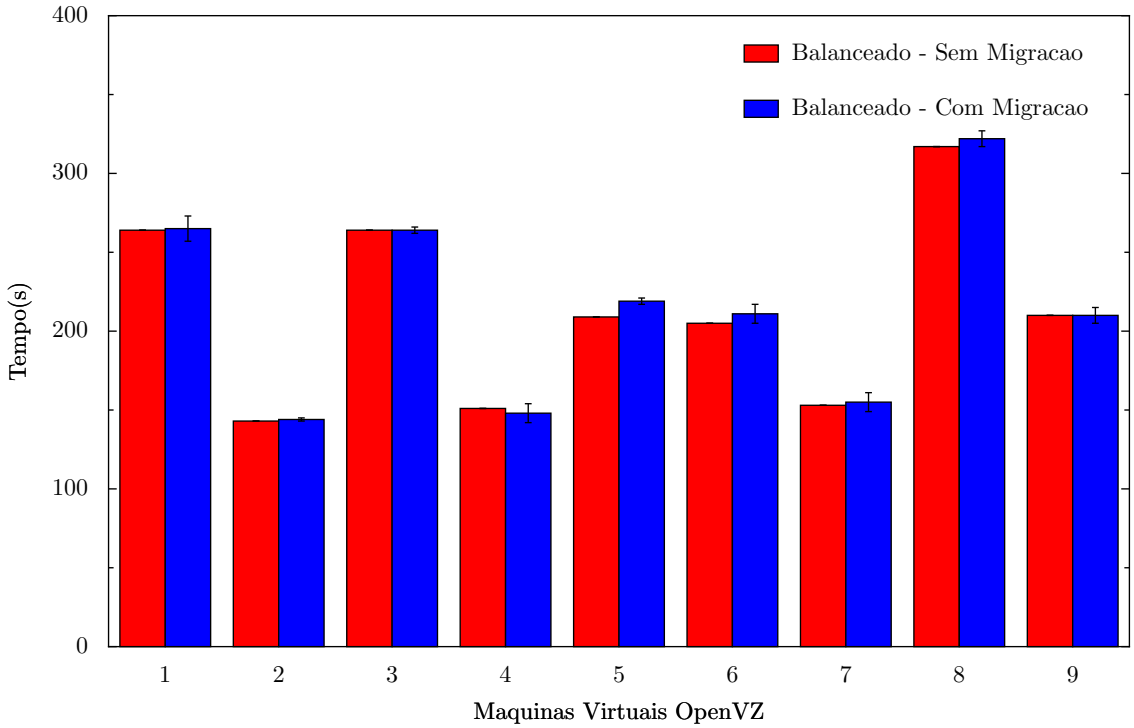

Figura 4.9: Experimento2, OpenVZ. Tempo de execução das MVs.

4.7.2 Xen

A Figura 4.10 também mostra que mesmo com migrações desnecessárias, ocorridas com o Xen, não prejudicou o tempo de execução dos MVs.

4.7.3 OpenVZ x Xen

A comparação entre os tempos do OpenVZ e do Xen, apresentadas nas Figuras 4.12 e 4.11, indicam que as duas técnicas tiveram resultados muito parecidos quando experimentados com a aplicação Sintética-Randômica. Isso mostra que, ou os *overheads* das migrações e execuções foram insignificantes, ou que foram apenas muito parecidos.

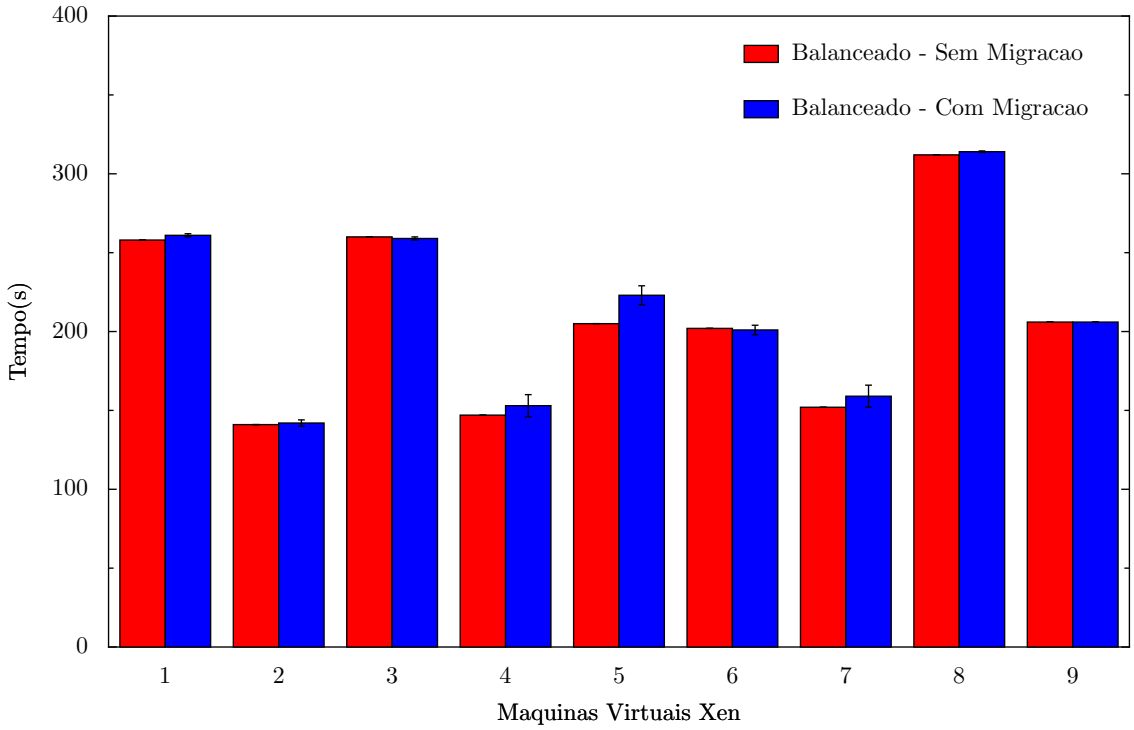

Figura 4.10: Experimento2, XM. Tempo de execução das MVs.

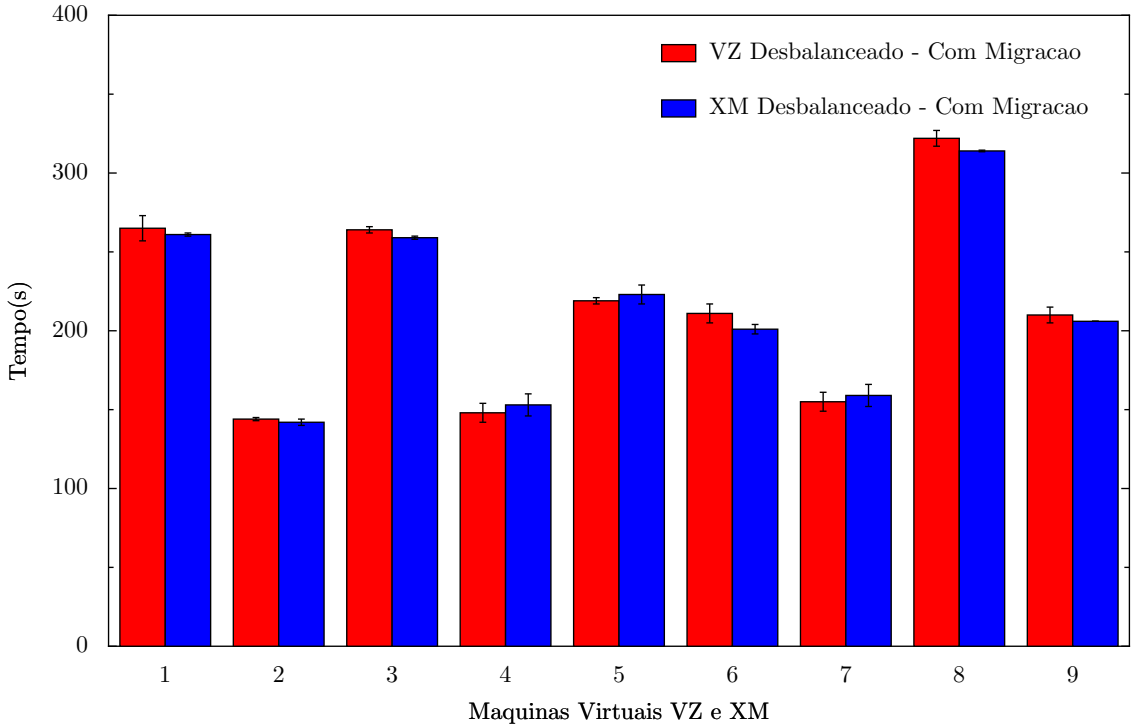

Figura 4.11: Experimento2, OpenVz x Xen. Tempo de execução das MV.

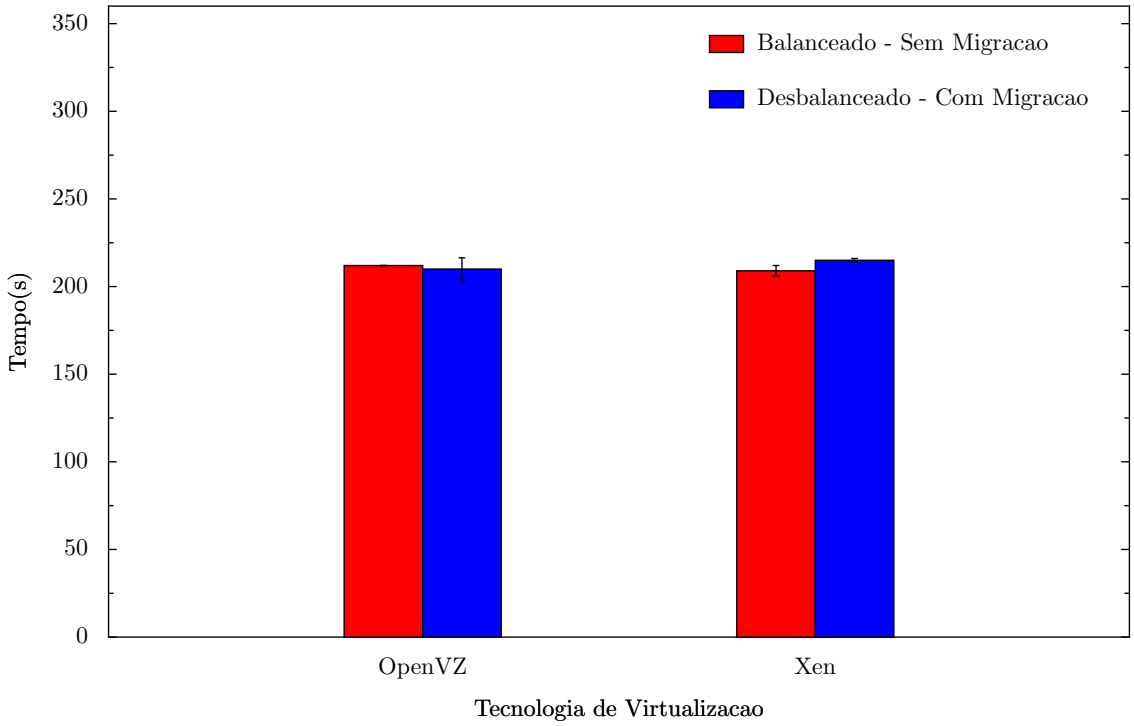

Figura 4.12: Experimento2, OpenVz x Xen. Tempo de execução médio das MV.

4.8 Experimento3

A aplicação, usada neste experimento, é a compilação do *kernel* do linux-1.6.12 com o comando: "make -j8 bzImage". Neste experimento, também são usadas três MFs, essas MFs recebem 9 MVs, sobre as quais é igualmente distribuído o trabalho de compilação realizado pelo sistema. A compilação é iniciada ao mesmo tempo em todas as MVs. Com as configurações utilizadas nos experimentos, e quando a compilação do *kernel* é executada em uma MF com a CPU não sobrecarregada, a aplicação ocupa cerca de 80% da CPU.

Para este experimento, foi necessário considerar as quantidades de memória alocada para cada MV. Se inicialmente foi alocado pouca memória, por conta da baixa demanda das aplicações experimentadas, a quantidade de memória requerida para a compilação do kernel deve ser o dobro do que vinha sendo usada, passando de 100MB para 200MB. No OpenVz, pelas características de uso de memória, conforme explicado no Capítulo 3, não houve problemas. Mas essa mudança teve bastante impacto para o Xen. Foi necessário descobrir o mínimo de memória adequada para que a aplicação não realizasse *swap*. Essa necessidade surge da escassez de memória da MF que deve suportar mais de 7 MVs, e conforme explicado no capítulo anterior, no Xen, a quantidade de memória reservada para as MV deve ser menor que a quantidade total de memória da MF.

As subseções seguintes, apresentam os resultados das variações deste experimento

para os dois modelos de virtualização.

4.8.1 OpenVZ

Como já era esperado, a Figura 4.13 mostra como o balanceamento de carga melhorou o tempo de execução das aplicações das MVs. O destaque apresentado nesta Figura, são os tempos de execução das MVs 1, 2, 3, 4, 5, 6 e 7 para a configuração Desbalanceado - Sem migração, representado pela cor verde. O tempo de execução das MVs 1, 2, 3, e 4, é cerca de 40% maior que o tempo das MVs 5, 6 e 7, mas essas MVs executam o mesmo *workload* na mesma MF conforme mostrado na Figura 4.3. Na verdade, esse resultado aponta a influência do disco na compilação do *kernel*. O tempo das MVs 1, 2, 3 e 4 é maior por que seus arquivos devem ser acessados remotamente, diferente do que ocorre com as MVs 5,6 e 7 cujos arquivos são locais, trazendo mais velocidade no acesso ao disco. Como a compilação exige muita escrita em arquivo e realizar a escrita nos arquivos remotos piora o tempo de execução da aplicação, o tempo de execução foi maior nas aplicações cujos arquivos estavam remotos.

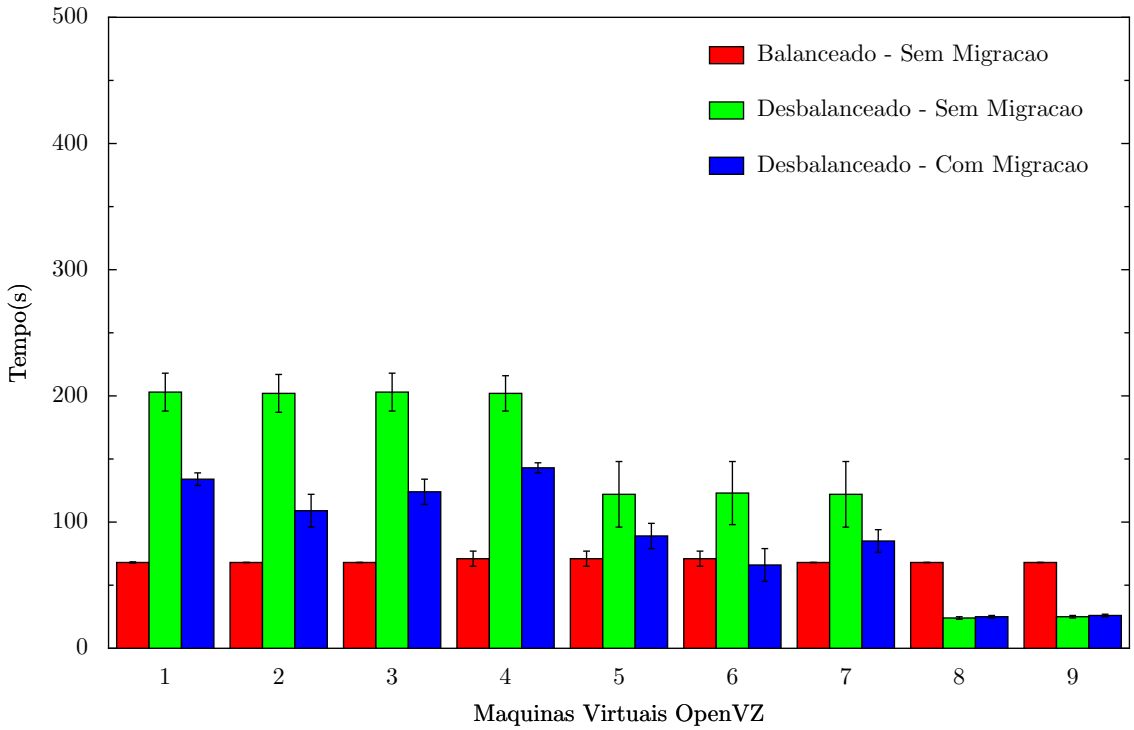

Figura 4.13: Experimento3, OpenVZ. Tempo de execução das MVs.

4.8.2 Xen

Dois fatos se destacam na Figura 4.14. O primeiro, é que o balanceamento de carga não melhorou o tempo de execução das MVs. O segundo, se refere a influência do disco, conforme comentado na subseção 4.8.1. O uso do balanceador não melhorou o tempo de execução, devido a complexidade e ao tempo necessário para migrar uma MV que estava compilando a aplicação Kernel. A grande quantidade de escrita em memória complica a migração da MV, quando esta é realizada com técnica de pré-cópia, como é o caso do Xen. A longa demora para migrar apenas uma máquina virtual prejudicou o balanceamento, e consequentemente a eficiência de execução do sistema, resultando em alto tempo de execução das aplicações, mesmo habilitando o balanceamento de carga. O tempo semelhante de execução para as sete primeiras MVs do Xen, na configuração Desbalanceado - Sem Migração (representado pela cor verde), difere do equivalente, apresentado para o OpenVZ. Este fato está relacionado com o algoritmo de escalonamento de CPU usado nos duas técnicas. As condições são iguais para os dois, mas o OpenVZ favorece as MVs com disco local, em detrimento das MVs com disco remoto.

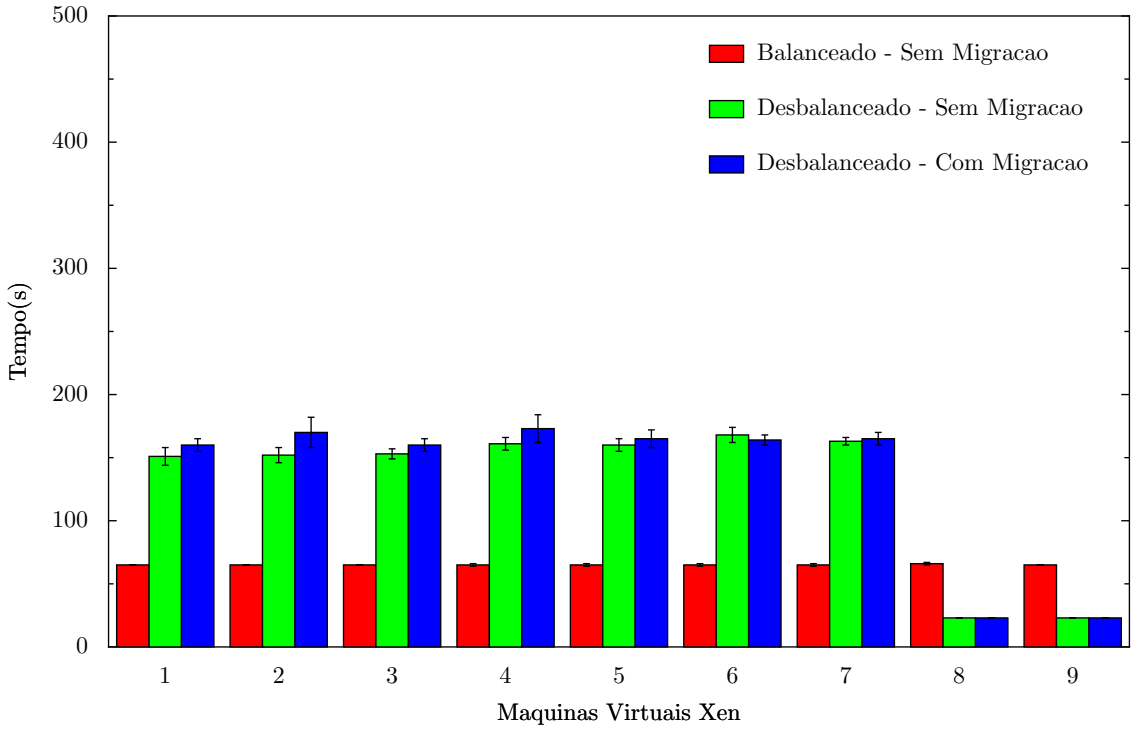

Figura 4.14: Experimento3, XM. Tempo de execução das MVs.

4.8.3 OpenVZ x Xen

Conforme discutido na subseção 4.8.1, ao observar os resultados do OpenVZ para a configuração Desbalanceado -Sem Migração, as MVs cujo sistema de arquivo estava na MF local, tiveram tempo de execução menor se comparado com as MVs com arquivos remotos. Já a subseção 4.8.1, para Xen, com essa mesma configuração, as MVs tiveram tempo de execução parecido. Pela Figura 4.15, pode-se notar que os tempos médios para a configuração Desbalanceado - Sem Migração, são equivalentes. Por esse motivo, pode-se concluir que o OpenVZ favorecia as MVs com acesso mais rápido ao disco, em detrimento as MVs que tinham acesso mais lento ao disco remoto, essa priorização caracteriza falha de isolamento [25].

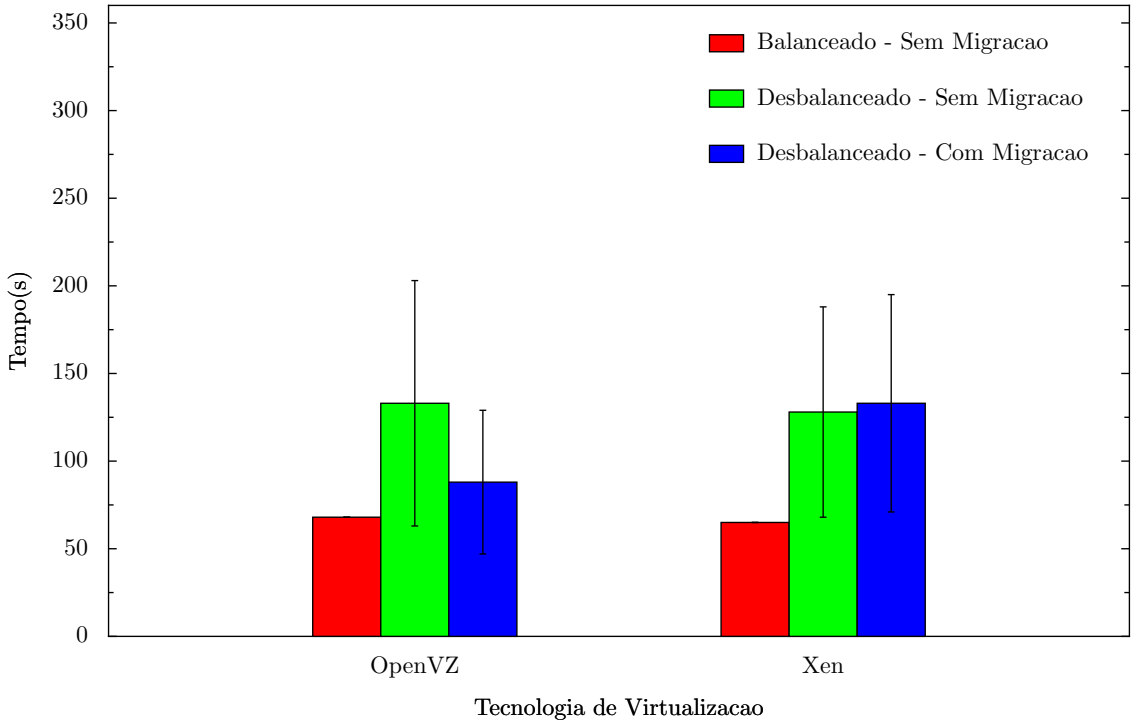

Figura 4.15: Experimento3, XM. Tempo de execução das MVs.

4.9 Discussão dos Resultados

Os experimentos apresentados mostraram que a técnica de migração pode influenciar na qualidade do balanceamento de carga. O Experimento1 e principalmente o experiento3, mostram que, a técnica de *live migration* usada pelo Xen, impediu que o sistema fosse balanceado no tempo desejado, enquanto a técnica usada pelo OpenVZ, permitiu um balanceamento mais eficiente. Os passos iterativos, realizados na pré-cópia implementada pelo Xen, em conjunto com sua técnica de virtualização, geraram grande quantidade de dados que foram transferidos pela rede durante as

fases da pré-cópia. A demora na transferência desses dados, tornou o balanceamento de carga ineficiente. Quanto maior a capacidade das máquinas do *cluster*, maior seria o impacto do tempo de transferência total de MV, pois o recursos das máquinas, estariam temporariamente desperdiçados. Seria interessante verificar se o impacto do tempo de migração seria maior em um *cluster* com mais máquinas.

No Experimento2, apesar das características bastante peculiares da aplicação utilizada, com destaque para o baixo tempo de migração total, as migrações desnecessárias realizadas na tentativa de balancear o sistema, não impactou o tempo execução das MVs. Este fato pode indicar que dificilmente o *overhead* de migração de máquina virtual - quando o tempo total de migração é baixo - e até mesmo o *downtime*, tanto do Xen quanto do OpenVZ, será relevante em algum sistema de balanceamento que não use aplicações interativa ou de tempo real. Por outro lado, um tempo longo de migração pode tornar o balanceamento de carga inviável. Num cenário onde a carga de processamento gerado pelas MVs é muito variável e o tempo de migração destas MVs é muito alto, para este caso balancear a carga do sistema pode até ser prejudicial, piorando a eficiência do *cluster*.

O Experimento3 também pode revelar uma falha de isolamento do OpenVZ. Sobre as condições impostas no Experimento3, o gerenciador do recurso CPU do OpenVZ, favoreceu determinadas MVs em detrimento de outras MVs da mesma MF. Esse fato só foi notado neste experimento, devido a característica da aplicação usada no Experimento3, pois esta, também dependia do tempo de acesso ao disco, enquanto as utilizadas nos outros experimentos, eram puramente *CPU-Bound*. A diferença entre os resultados do Xen e o OpenVZ, pode ser atribuída as características dos escalonadores de processo por eles utilizados.

Ao comparar os resultados do Xen e do OpenVZ obtidos nos experimentos realizados, foi observado que o tempo de migração total de uma MV pode influenciar diretamente na qualidade do balanceamento de carga e consequentemente no *throughput* do sistema.

Capítulo 5

Trabalhos Relacionados

Com o crescente interesse sobre as tecnologias de virtualização, muitas pesquisas realizadas neste campo, buscando geralmente melhorar a eficiência e a segurança de sistemas em diversas áreas da computação. Este fato ressalta a importância desses trabalhos sobre o futuro da virtualização. Neste capítulo, são abordados os trabalhos que serviram como base para esta dissertação ou que exploram algum assunto diretamente relacionado ao tema aqui abordado.

Na primeira seção são relacionados os trabalhos que exploram ou comparam características de sistemas de virtualização. A segunda seção, a 5.2, apresenta os trabalhos que abordam a migração de máquinas virtuais. Na última seção, são discutidos os trabalhos sobre algoritmos de balanceamento de carga.

5.1 Comparação entre Técnicas de Virtualização

Com toda a expectativa sobre o futuro da virtualização, busca-se comparar as tecnologias de virtualização que estão em evidência. O artigo Xen and the Art of Virtualization [40] apresenta a arquitetura e algumas estratégias de virtualização, usadas pelo Xen, para implementar para-virtualização. Ainda neste trabalho, são realizadas comparações de eficiência, que apresentam a superioridade da para-virtualização sobre a virtualização total. Em outro trabalho , Soltesz [19] apresenta a arquitetura de um sistema virtualizado por *containers*, e usa execuções de *microbenchmarks* para comparar sua eficiência com as das outras técnicas de virtualização. Para grande parte das funções básicas do SO testadas, a técnica de *containers* foi superior a de para-virtualização.

Mesmo com algumas restrições, virtualização de sistemas têm sido aplicadas em *clusters* de alto desempenho como citado no trabalho [13]. Apesar dos desafios de eficiência, a virtualização pode melhorar a utilização dos recursos e a comodidade para os usuários de *clusters* de alto desempenho. Em seu trabalho, Walters [24] compara a eficiência de diferentes estratégias de virtualização. São usados *benchmarks*

de computação científica para comparar Xen, Openvz e VMware quanto a eficiência no uso dos recursos de CPU, rede e disco. Toda as análises dos resultados são baseadas nos resultados individuais de cada um destes recursos; não é considerado o interrelacionamento dos recursos nem a possibilidade de migração das máquinas virtuais do sistema.

Em uma tentativa de resolver, simultaneamente, o problema de desempenho da virtualização total e a necessidade de alteração do SO hospedado na para-virtualização, a arquitetura x86 foi estendida para suportar a virtualização clássica. No trabalho de Adams, Agesen e outros [24], são comparadas as primeiras implementações de virtualização com suporte de hardware com a virtualização total. A virtualização por hardware demonstrou ser menos eficiente na maioria dos testes realizados. Os autores concluíram que os resultados refletem a imaturidade da tecnologia. Foi verificado que os pontos de maior dificuldade da virtualização por software e por hardware não são comuns, desta forma, a espectativa de uma combinação mais madura dessas duas técnicas torna a tecnologia de virtualização promissora.

O trabalho [25], se preocupa em comparar somente a capacidade de isolamento das técnicas. Para avaliar o isolamento, é medido o quanto uma aplicação, faminta por recurso e executando em uma MV, pode influenciar a execução de aplicações que estão em outra MV. Nos testes, foram comparados as técnicas de virtualização total, para-virtualização e virtualização por containers. Os resultados foram sensivelmente diferentes para cada uma das classes. A técnica de virtualização total forneceu isolamento perfeito nos testes, a para-virtualização, um nível de isolamento muito próximo do perfeito, já o isolamento da virtualização por containers, foi em alguns casos ruim e estando sempre abaixo das outras classes.

5.2 Migração de Máquinas Virtuais

A migração de instâncias de SOs entre máquinas físicas é resultado de uma separação clara entre hardware e software. Essa característica torna bastante atrativo o uso de virtualização em *data centers* por facilitar o remanejamento por falhas e permitir balanceamento de carga. Chiristopher Clark, em seu trabalho [18], além de revisar alguns métodos de migração de MVs que podem ser usados em *data centers*, apresentam o conceito de WWS(writable working set) que serve como base em sua proposta de melhoria na migração por pré-cópia. Nos testes, a pré-cópia modificada no trabalho apresentou *downtime*, imperceptível, de apenas 60 ms na migração de um servidor de jogo interativo.

Reconhecendo o potencial dos sistemas de virtualização em permitir a consolidação de serviços em um ou mais servidores físicos, Padala, em seu trabalho [22]

preocupa-se em encontrar a tecnologia de virtualização e a configuração mais adequada para consolidar determinados serviços. Neste sentido, são realizados testes combinando a consolidação dos servidores web e de banco de dados em dois servidores físicos, utilizando o Xen ou o OpenVz como tecnologia de virtualização. Um programa simula as requisições realizadas pelos clientes ao servidor web e este, por sua vez, acessa o banco de dados. São realizados testes de escalabilidade, neste caso, aumentando o número de clientes requisitantes, tempo de resposta do servidor e consumo de cpu. Nestes testes o OpenVZ apresentou resultados superiores aos apresentados pelo Xen.

5.3 Balanceamento de Carga

Em 1997, antes da repopularização dos sistemas de virtualização, James e outros [6] usaram um algoritmo de alocação de rotas para Circuitos Virtuais, como base para balancear a execução de *jobs* por diversas máquinas físicas disponíveis. Já no trabalho [3], aproveitando as facilidades oferecidas pela migração de MV, o autor propõe algoritmos para balancear a carga do sistema migrando MVs entre as máquinas físicas. No sistema de balanceamento proposto, são coletadas periodicamente informações sobre a carga de cpu, memória e rede gerada por cada uma destas VMs. Para decidir se deve haver migração, essas informações são tratadas e comparadas com o SLA proposto para cada MV. Uma limitação do sistema implementado é que este só suporta Xen como sistema de virtualização.

Em um trabalho recente [2], foi proposta uma métrica que avalia o desbalanceamento da carga do sistema. Esta métrica resultou na criação de um *framework* de balanceamento de MVs. A arquitetura do *framework* e o algoritmo de balanceamento deste trabalho são semelhantes aos apresentados nesta dissertação. O gerenciamento também é centralizado e o algoritmo é guloso, buscando o menor desbalanceamento para cada iteração, parecido com o que foi implementado aqui. A diferença entre o trabalho [2] e esta dissertação, são os objetivos e as limitações. Enquanto o trabalho se preocupa em comparar o seu algoritmo de balanceamento com os já existentes a dissertação busca comparar as técnicas de virtualização. O *framework* do trabalho suporta somente a tecnologia VMware ESX, já o sistema de balanceamento desta dissertação pode operar com o Xen e com o OpenVZ.

Capítulo 6

Conclusões

Nesta dissertação é realizada uma comparação experimental da eficiência e do comportamento das técnicas de virtualização por containers (OpenVZ) e paravirtualização (Xen), quando suas máquinas virtuais estão sujeitas a migração, com a finalidade de balancear a carga de processamento de um *cluster* de computadores. O principal objetivo, apontado no início desta dissertação, é de comparar experimentalmente a eficiência e o comportamento de duas técnicas de virtualização, quando suas máquinas virtuais podem ser migradas para balancear a carga de CPU do sistema.

Os resultados apontam algumas características presentes na migração implementada pela técnica usada no Xen, que podem ser prejudiciais para a realização do balanceamento de caraga. O alto tempo total de migração, resultante da técnica de migração usada pelo Xen, fez com que o balanceamento acontecesse de maneira lenta, subutilizando a capacidade do sistema e aumentando o tempo total de migração da aplicação. No entanto, o objetivo de comparar a influência da sobrecarga de migração das máquinas virtuais não foi totalmente satisfeito. Para as aplicações experimentadas neste trabalho, o *overhead* de migração foi sempre insignificante, e portanto, não era passível de comparação. Mas, sabe-se que para aplicações interativa ou de tempo real, este *overhead* pode ser determinante na escolha da técnica de virtualização.

Um resultado secundário, que vai além dos objetivos propostos neste trabalho, está a falha de isolamento de MV ocorrida em um dos experimentos. Neste mesmo sentido, fica como legado do trabalho realizado, o gerenciador de máquinas virtuais, hábil para realizar outros experimentos, tanto para avaliar virtualização de sistemas como para comparar algoritmos de balanceamento de carga.

6.1 Trabalhos futuros

Durante a discussão dos resultados deste trabalho, notou-se a necessidade de ampliar e diversificar os experimentos aqui realizados. Outros trabalhos poderiam explorar

algumas ideias ou o próprio ambiente aqui desenvolvido, a fim de ampliar os resultados obtidos nesta pesquisa. Entre estes possíveis trabalhos, estão:

- **Ampliar a Quantidade de Técnicas de Virtualização Avaliadas**
  Neste trabalho, foram avaliadas duas técnicas de virtualização, que são considerados atualmente como as mais eficientes. Mas existem outras técnicas importantes, que poderiam, a partir dos resultados dos experimentos, serem classificadas quanto a sua eficiência em ambientes com balanceamento de carga. A virtualização total, por ser uma das mais utilizadas, e a virtualização com o auxílio de hardware, por demonstrar ser uma técnica promissora [24], deveriam também ter sua eficiência avaliada, quando executada em um *cluster* que permita migrar MVs com a finalidade de balancear a carga do sistema.

- **Ampliar a Quantidade de Máquinas Físicas e Virtuais no Sistema**
  A difusão do uso de *Cloud Computing* pode implicar na necessidade de grandes *clusters* com suporte a virtualização. Neste ambiente, o balanceamento de carga pode aumentar a eficiência do sistema. Como tais *clusters* podem ser compostos por centenas de MFs e MVs, é importante identificar qual técnica ou combinação de técnicas de virtualização pode ser mais adequada para estes grandes sistemas.

- **Aplicações com Características Diferentes**
  Existem importantes tipos de aplicações que poderiam ser avaliadas. Aplicações de execução paralela, aplicações interativas, aplicações com muita escrita em memória e aplicações que demandam uso intenso da rede. São aplicações com características distintas e que podem gerar resultados diferentes dos encontrados nos experimentos aqui realizados. Por exemplo, sabe-se que aplicações com escrita intensa em memória apresentam maior impacto no *dowtime* e no tempo de migração das MVs, isso vale tanto para o OpenVZ[5] quanto para o Xen[26], e este impacto poderia enfatizar algum resultado obtido nesta dissertação. Seria interessante avaliar as técnicas por meio de aplicações que exploram recursos variados do sistema virtualizado, isso tornaria a avaliação mais completa.

- **Acrescentar Novos Parâmetros para Determinar o Balanceamento**
  O algoritmo de balanceamento poderia utilizar outras métricas para determinar o equilíbrio do sistema. Seria válido utilizar como métrica a carga gerada na rede e também quantidade de escritas em memória. Já existem algoritmos que utilizam esses parâmetros, mas seus trabalhos se restringem a avaliar o algoritmo construído e não as técnicas de virtualização.

- **Avaliar as Técnicas de Migração quando Usadas para Balancear Carga do Sistema**

  Conforme já mencionado no Capítulo 2, na seção que trata a migração de máquinas virtuais, as técnicas de migração têm qualidades distintas. Sabe-se que o custo de se ter um baixo *downtime* é a complexidade e o tempo total de migração. Em contrapartida, a migração para e copia tem alto *downtime* mas, baixa complexidade e menor tempo de migração. Enquanto o uso de *live migration* resulta em melhor tempo de execução das aplicações da MV migrada, a migração executada por essa técnica, pode impactar, negativamente, o tempo de execução das aplicações de MVs, que estejam executando na origem ou no destino. Por esse motivo, seria interessante realizar experimentos com técnicas diferentes de migração.

Referências Bibliográficas

[1] DEVINE, SCOTT W., B. *Virtualization system including a virtual machine monitor for a computer with a segmented architecture*, May 2002. Disponível em: <http://www.freepatentsonline.com/6397242.html>.

[2] ARZUAGA, E., KAELI, D. R. "Quantifying load imbalance on virtualized enterprise servers". In: *WOSP/SIPEW '10: Proceedings of the first joint WOSP/SIPEW international conference on Performance engineering*, pp. 235–242, New York, NY, USA, 2010. ACM. ISBN: 978-1-60558-563-5. doi: http://doi.acm.org/10.1145/1712605.1712641.

[3] WOOD, T., SHENOY, P., ARUN. "Black-box and Gray-box Strategies for Virtual Machine Migration". pp. 229–242. Disponível em: <http://www.usenix.org/events/nsdi07/tech/wood.html>.

[4] SINGH, A., KORUPOLU, M., MOHAPATRA, D. "Server-storage virtualization: integration and load balancing in data centers". In: *SC '08: Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, pp. 1–12, Piscataway, NJ, USA, 2008. IEEE Press. ISBN: 978-1-4244-2835-9.

[5] ZHAO, M., FIGUEIREDO, R. J. "Experimental study of virtual machine migration in support of reservation of cluster resources". In: *VTDC '07: Proceedings of the 2nd international workshop on Virtualization technology in distributed computing*, pp. 1–8, 2007. ISBN: 978-1-59593-897-8. doi: http://doi.acm.org/10.1145/1408654.1408659.

[6] ASPNES, J., AZAR, Y., FIAT, A., et al. "On-line routing of virtual circuits with applications to load balancing and machine scheduling", *J. ACM*, v. 44, n. 3, pp. 486–504, 1997. ISSN: 0004-5411. doi: http://doi.acm.org/10.1145/258128.258201.

[7] AWERBUCH, B., AZAR, Y., PLOTKIN, S., et al. "Competitive routing of virtual circuits with unknown duration". In: *SODA '94: Proceedings of the fifth annual ACM-SIAM symposium on Discrete algorithms*, pp. 321–

327, Philadelphia, PA, USA, 1994. Society for Industrial and Applied Mathematics. ISBN: 0-89871-329-3.

[8] FENG, W.-C., CHANG, F., FENG, W.-C., et al. "A traffic characterization of popular on-line games", *IEEE/ACM Trans. Netw.*, v. 13, n. 3, pp. 488–500, 2005. ISSN: 1063-6692. doi: http://dx.doi.org/10.1109/TNET.2005.850221.

[9] KEREN, A., BARAK, A. "Opportunity Cost Algorithms for Reduction of I/O and Interprocess Communication Overhead in a Computing Cluster", *IEEE Trans. Parallel Distrib. Syst.*, v. 14, n. 1, pp. 39–50, 2003. ISSN: 1045-9219. doi: http://dx.doi.org/10.1109/TPDS.2003.1167369.

[10] MOHAMMADPOUR, P., SHARIFI, M., PAIKAN, A. "A Self-Training Algorithm for Load Balancing in Cluster Computing". In: *NCM '08: Proceedings of the 2008 Fourth International Conference on Networked Computing and Advanced Information Management*, pp. 104–109, Washington, DC, USA, 2008. IEEE Computer Society. ISBN: 978-0-7695-3322-3-01. doi: http://dx.doi.org/10.1109/NCM.2008.178.

[11] CHOI, H. W., KWAK, H., SOHN, A., et al. "Autonomous learning for efficient resource utilization of dynamic VM migration". In: *ICS '08: Proceedings of the 22nd annual international conference on Supercomputing*, pp. 185–194, New York, NY, USA, 2008. ACM. ISBN: 978-1-60558-158-3. doi: http://doi.acm.org/10.1145/1375527.1375556.

[12] WERSTEIN, P., SITU, H., HUANG, Z. "Load Balancing in a Cluster Computer". In: *PDCAT '06: Proceedings of the Seventh International Conference on Parallel and Distributed Computing, Applications and Technologies*, pp. 569–577, Washington, DC, USA, 2006. IEEE Computer Society. ISBN: 0-7695-2736-1. doi: http://dx.doi.org/10.1109/PDCAT.2006.77.

[13] MERGEN, M. F., UHLIG, V., KRIEGER, O., et al. "Virtualization for high-performance computing". , 2006. ISSN: 0163-5980.

[14] DEVARAKONDA, M. V., IYER, R. K. "Predictability of Process Resource Usage: A Measurement-Based Study on UNIX", *IEEE Trans. Softw. Eng.*, v. 15, n. 12, pp. 1579–1586, 1989. ISSN: 0098-5589. doi: http://dx.doi.org/10.1109/32.58769.

[15] MCNETT, M., GUPTA, D., VAHDAT, A., et al. "Usher: an extensible framework for managing custers of virtual machines". In: *LISA'07: Proceedings of the 21st conference on Large Installation System Administration*

Conference, pp. 1–15, Berkeley, CA, USA, 2007. USENIX Association. ISBN: 978-1-59327-152-7.

[16] *CentOS*. Disponível em: <`http://www.centos.org`>.

[17] *Usher*. Disponível em: <`http://usher.ucsd.edu/trac/wiki/UsherDevelopment`>.

[18] CLARK, C., FRASER, K., HAND, S., et al. "Live migration of virtual machines". In: *NSDI'05: Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation*, pp. 273–286, Berkeley, CA, USA, 2005. USENIX Association.

[19] SOLTESZ, S., PÖTZL, H., FIUCZYNSKI, M. E., et al. "Container-based operating system virtualization: a scalable, high-performance alternative to hypervisors", *SIGOPS Oper. Syst. Rev.*, v. 41, n. 3, pp. 275–287, 2007. ISSN: 0163-5980. doi: http://doi.acm.org/10.1145/1272998.1273025.

[20] BHATTIPROLU, S., BIEDERMAN, E. W., HALLYN, S., et al. "Virtual servers and checkpoint/restart in mainstream Linux", *SIGOPS Oper. Syst. Rev.*, v. 42, n. 5, pp. 104–113, 2008. ISSN: 0163-5980. doi: http://doi.acm.org/10.1145/1400097.1400109.

[21] WALTERS, J. P., CHAUDHARY, V., CHA, M., et al. "A Comparison of Virtualization Technologies for HPC", *Advanced Information Networking and Applications, International Conference on*, v. 0, pp. 861–868, 2008. ISSN: 1550-445X. doi: http://doi.ieeecomputersociety.org/10.1109/AINA.2008.45.

[22] PADALA, P., ZHU, X., WANG, Z., et al. *Performance evaluation of virtualization technologies for server consolidation*. Relatório técnico, 2007.

[23] BARHAM, P., DRAGOVIC, B., FRASER, K., et al. "Xen and the art of virtualization". In: *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles*, pp. 164–177, New York, NY, USA, 2003. ACM. ISBN: 1-58113-757-5. doi: http://doi.acm.org/10.1145/945445.945462.

[24] ADAMS, K., AGESEN, O. "A comparison of software and hardware techniques for x86 virtualization". In: *ASPLOS-XII: Proceedings of the 12th international conference on Architectural support for programming languages and operating systems*, pp. 2–13, New York, NY, USA, 2006. ACM. ISBN: 1-59593-451-0. doi: http://doi.acm.org/10.1145/1168857.1168860.

[25] MATTHEWS, J. N., HU, W., HAPUARACHCHI, M., et al. "Quantifying the performance isolation properties of virtualization systems". In: *ExpCS '07: Proceedings of the 2007 workshop on Experimental computer science*, p. 6, New York, NY, USA, 2007. ACM. ISBN: 978-1-59593-751-3. doi: http://doi.acm.org/10.1145/1281700.1281706.

[26] NAGARAJAN, A. B., MUELLER, F., ENGELMANN, C., et al. "Proactive fault tolerance for HPC with Xen virtualization". In: *ICS '07: Proceedings of the 21st annual international conference on Supercomputing*, pp. 23–32, New York, NY, USA, 2007. ACM. ISBN: 978-1-59593-768-1. doi: http://doi.acm.org/10.1145/1274971.1274978.

[27] *Xen CreditScheduler*. Disponível em: <`http://wiki.xensource.com/xenwiki/CreditScheduler.`>.

[28] *Microsoft*, 2010. Disponível em: <`http://www.microsoft.com/`>.

[29] *Xen*, 2010. Disponível em: <`http://www.xen.org/`>.

[30] *VMware*, 2010. Disponível em: <`www.vmware.com`>.

[31] *Amd*, 2010. Disponível em: <`http://www.amd.com/`>.

[32] *Stress*. Disponível em: <`http://weather.ou.edu/~apw/projects/stress/`>.

[33] *OpenVZ*. Disponível em: <`http://wiki.openvz.org/Main_Page`>.

[34] *The Jail Subsystem*. Disponível em: <`http://www.freebsd.org/doc/en_US.ISO8859-1/books/arch-handbook/jail.html`>.

[35] *Solaris Containers*. Disponível em: <`http://www.freebsd.org/doc/en_US.ISO8859-1/books/arch-handbook/jail.html`>.

[36] *Vserver*. Disponível em: <`http://linux-vserver.org/Welcome_to_Linux-VServer.org`>.

[37] *Virtuozzo*. Disponível em: <`http://www.parallels.com/products/pvc45/`>.

[38] *Python Programming Language*. Disponível em: <`http://www.python.org/`>.

[39] SUGERMAN, J., VENKITACHALAM, G., LIM, B.-H. "Virtualizing I/O Devices on VMware Workstation's Hosted Virtual Machine Monitor". In: *Proceedings of the General Track: 2002 USENIX Annual Technical Conference*, pp. 1–14, Berkeley, CA, USA, 2001. USENIX Association. ISBN: 1-880446-09-X.

[40] BARHAM, P., DRAGOVIC, B., FRASER, K., et al. "Xen and the art of virtualization". In: *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles*, pp. 164–177, New York, NY, USA, 2003. ACM. ISBN: 1-58113-757-5. doi: http://doi.acm.org/10.1145/945445.945462.

[41] OSMAN, S., SUBHRAVETI, D., SU, G., et al. "The design and implementation of Zap: a system for migrating computing environments". In: *OSDI '02: Proceedings of the 5th symposium on Operating systems design and implementation*, pp. 361–376, New York, NY, USA, 2002. ACM. ISBN: 978-1-4503-0111-4. doi: http://doi.acm.org/10.1145/1060289.1060323.

[42] BHADANI, A., CHAUDHARY, S. "Performance evaluation of web servers using central load balancing policy over virtual machines on cloud". In: *COMPUTE '10: Proceedings of the Third Annual ACM Bangalore Conference*, pp. 1–4, New York, NY, USA, 2010. ACM. ISBN: 978-1-4503-0001-8. doi: http://doi.acm.org/10.1145/1754288.1754304.

[43] SILBERSCHATZ, A., GALVIN, P. B. *Operating System Concepts*. Boston, MA, USA, Addison-Wesley Longman Publishing Co., Inc., 1997. ISBN: 0201591138.

[44] HENNESSY, J. L., PATTERSON, D. A. *Computer architecture: a quantitative approach*. San Francisco, CA, USA, Morgan Kaufmann Publishers Inc., 2002. ISBN: 1-55860-596-7.

[45] ROBIN, J. S., IRVINE, C. E. "Analysis of the Intel Pentium's ability to support a secure virtual machine monitor". In: *SSYM'00: Proceedings of the 9th conference on USENIX Security Symposium*, pp. 10–10, Berkeley, CA, USA, 2000. USENIX Association.

[46] SMITH, J., NAIR, R. *Virtual Machines: Versatile Platforms for Systems and Processes (The Morgan Kaufmann Series in Computer Architecture and Design)*. San Francisco, CA, USA, Morgan Kaufmann Publishers Inc., 2005. ISBN: 1558609105.

[47] MILOJÍČIĆ, D. S., DOUGLIS, F., PAINDAVEINE, Y., et al. "Process migration", *ACM Comput. Surv.*, v. 32, n. 3, pp. 241–299, 2000. ISSN: 0360-0300. doi: http://doi.acm.org/10.1145/367701.367728.

[48] BARAK, A. "The MOSIX Multicomputer Operating System for High Performance Cluster Computing", *Journal of Future Generation Computer Systems*, v. 13, pp. 4–5, 1998.

[49] SMITH, J. E., NAIR, R. "The Architecture of Virtual Machines", *Computer*, v. 38, n. 5, pp. 32–38, 2005. ISSN: 0018-9162. doi: http://dx.doi.org/10.1109/MC.2005.173.

[50] HENNESSY, J., PATTERSON, D. *Computer Architecture - A Quantitative Approach*. Morgan Kaufmann, 2003.

[51] R. J. ADAIR, R. U. BAYLES, L. W. C., CREASY, R. J. *A Virtual Machine System for the 360/40*. Relatório técnico, IBM Cambridge Scientific Center Report No. G320-2007, 1966.

[52] POPEK, G. J., GOLDBERG, R. P. "Formal requirements for virtualizable third generation architectures", *Commun. ACM*, v. 17, n. 7, pp. 412–421, 1974. ISSN: 0001-0782. doi: http://doi.acm.org/10.1145/361011.361073.

[53] LINDHOLM, T., YELLIN, F. *Java Virtual Machine Specification*. Boston, MA, USA, Addison-Wesley Longman Publishing Co., Inc., 1999. ISBN: 0201432943.

[54] CREASY, R. J. "The origin of the VM/370 time-sharing system", *IBM J. Res. Dev.*, v. 25, n. 5, pp. 483–490, 1981. ISSN: 0018-8646.

[55] AMAZON. *EC2*, 2010. Disponível em: <`http://aws.amazon.com/ec2/`>.

[56] LINUXCHIX. *Kernel Hacking Lesson*, 2006. Disponível em: <`http://www.linuxchix.org/content/courses/`>.

Apêndice A

Migração de Game Interativo no OpenVZ

Este apêndice, é parte de um relatório sobre experimentos que avaliam a migração de máquinas virtuais no OpenVZ. Os resultados obtidos nos experimentos, antecederam e motivaram parte do trabalho realizado na dissertação. Na seção A.1 é feita uma breve introdução sobre o conteúdo do apêndice. A seção A.2, trata a migração de MVs da virtualização por containers. Na seção A.3 é explicado como foram realizadas as configurações de rede, para suportar o primeiro experimento. A seção A.4 apresenta o ambiente experimental. Na seção A.5, é apresentado o primeiro conjunto de experimentos. Na seção A.6, tem-se o segundo conjunto de experimentos. E por fim, na seção A.7, é feita a conclusão do apêndice.

A.1 Introdução

Este relatório apresenta os resultados dos testes de migração de uma máquina virtual entre duas máquinas físicas, bem como a interpretação crítica dos resultados apresentados. O objetivo dos experimentos é identificar os problemas e encontrar possíveis soluções para modelo de migração MV usado na virtualização por containers.

São realizados experimentos migrando uma MV sem aplicações, variando a taxa de transmissão da rede e usando um compactador para compactar os dados transmitidos. E também são feitos experimentos com a migração, em rede Gigabit, de MVs executando diferentes aplicações.

A.2 Migração de Conteiners

As MVs de *conteiners*, são mas complicadas para migrar as MVs que usam migração total ou paravirtualização. Este fato se deve a diferença entre estas duas arquiteturas. Na virtualização total, basta ver a MV, com todos os seus processos como parte da memória; neste caso, abstratamente, pode-se dizer que basta copiar toda a área de memória da MV, que se deseja migrar, para a máquina destino, e após a cópia desta área de memória a MV passa a executar no destino. Para os containers, não se pode migrar todo o SO, já que ele é compartilhado entre as MVs. Neste caso, deve-se migrar, juntamente com as páginas dos processos, informações com as quais seja possível recriar as estruturas da MV no kernel da máquina destino. Este fator dificulta a implementação de *live migration* com baixo downtime e explica porque os mecanismos de migração de MVs virtualizada por containers, são considerados ruins quando comparado com os mecanismos disponíveis para migrar MVs de sistemas de virtualização total ou paravirtualização.

A.2.1 Live migration no Openvz

Atualmente, a migração do openvz pode ser dividida nas seguintes fases:

1. *Suspend* - Nesta fase todos os processos da MV são interrompidos, não podendo mais executar. Isso é necessário para manter a consistência de memória entre o fim da execução da MV na origem e o início da execução da MV no destino.

2. *Dump* - O processo de Dump é realizado com a finalidade de coletar as informações necessárias para recriar a MV no destino. Entre essas informações coletadas estão: as estruturas de kernel e as páginas de memória de cada processo. Todas estas informações são colocadas em um arquivo chamado arquivo de dump.

3. *Copy Dump File*. Após a conclusão do arquivo de *dump*, é necessário enviar este arquivo para a máquina destino, isso é feito neste passo.

4. *Undump*. Agora que a máquina destino possui as informações, contidas no arquivo de *dump*, necessárias para recriar a máquina virtual, é realizado o *Undump*. Por meio do arquivo de dump pode-se saber as estruturas de kernel e outros recursos que devem ser alocados para executar uma MV consistente com a MV que estava executando na MV fonte.

5. *Restart* - Este último passo, habilita os processos da MV para serem executados na máquina de destino.

A.3 Migração em Bandas Limitadas

Para realização dos experimentos, foi necessário utilizar uma ferramenta que limitasse taxa de transmissão da rede.

A.3.1 TC

O TC (*Traffic Control*), em português, controle de tráfego, atua por meio de algoritmos que manipulam como os dados são transferidos pela interface física de rede. É sua responsabilidade por enfileirar, descartar, priorizar, assim como, dar garantia a um determinado tráfego, muitas vezes, sendo necessário reduzir o tráfego das demais conexões.

O kernel do linux é estruturado utilizando os conceitos de *Qdiscs*, *Classes* e *Filters*, para gerenciar o enfileiramento do tráfego de saída das interfaces de rede. O **TC** utiliza estes conceitos, abaixo explicados, para controlar o tráfego da rede.

Qdiscs As disciplinas de enfileiramento - *Queueing Discipline* -, também chamadas de algoritmos de escalonamento de pacotes, são utilizadas para controlar os pacotes, antes de serem transmitidos para a interface de rede, adicionando a capacidade de prover qualidade de serviço - *Quality of Service* -, limite de tráfego, além de poder priorizar um determinado tráfego. Existem vários algoritmos que implementam estas funcionalidades de controle dos dados transmitidos, e cada interface de rede possui um algoritmo principal associado, conhecido como qdisc root.

Classes As classes são utilizadas para agrupar um ou vários tráfegos, que compartilham um mesmo controle ou limite. Utiliza uma estrutura de árvore, onde a classe principal é chamada de *class root*, esta, indica o valor máximo a ser utilizado pela interface, e consequentemente, o valor a ser distribuído entre as sub-classes. Assim como, a *class root* tem uma ou várias classes ligadas a ela, cada classe que está ligada a classe *root*, pode possuir uma ou várias classes, assim,sucessivamente. Com isto, é possível elaborar complexas estruturas de controle de tráfego.

Filters Os filtros definem as regras e direcionam os pacotes para uma determinada classe. O algoritmo mais utilizado é *Universal 32bit comparisons* (U32), capaz de identificar pacotes por IP de origem, IP de destino, porta de origem e porta de destino.

Segue abaixo o script que foi utilizado para o controle do tráfego e a explicação dos comandos:

```
#Retirando qualquer controle aplicado a interface Eth0:
tc qdisc del dev eth0 root
#Disciplina de enfileiramento aplicada a Eth0 será HTB:
qdisc add dev eth0 handle 1: root htb default 1000
#Class principal, class root, com limite e garantia de 600000Kbit:
tc class add dev eth0 classid 1:1000 root htb rate 60000Kbit ceil 60000Kbit
#Classe que compartilha o limite da class root, com limite e garantia de $LIMITE:
tc class add dev eth0 classid 1:1001 parent 1:1000 htb rate $LIMITE ceil $LIMITE
#Disciplinas de enfileiramento para atuar sob cada classe, classless qdiscs:
tc qdisc add dev eth0 handle 1001: parent 1:1001 cbq default 1000
#Filtros que direcionam os pacotes para as respectivas classes, filters:
tc filter add dev eth0 parent 1: protocol ip prio 5 u32 flowid 1:1001 match
ip src 10.10.40.0/24
```

A.4 Ambiente Experimental

Os testes coletam os tempos inerentes a migração MV para diferentes taxas de transmissão da rede, variando também a forma de migração que pode ser com ou sem compactação. Usado compactador Bzip2, para compactar o arquivo de *dump* (*Dump File*), migrado entre as máquinas fonte e destino. Os principais tempos a serem coletados são: tempo de migração total e o `downtime`. O resultado dos teste deve revelar se é vantajoso ou não utilizar compactação na migração de uma máquina virtual, que migra em uma rede com uma determinada largura de banda.

As máquinas são IBMs com processador Pentium D 3.0GHz, 2048KB de cache, ligadas por uma rede Gigabit. São usadas duas máquinas com as características acima. Os experimentos ocorrem migrando as MVs entre essas duas máquinas.

A.5 Migração de MV sem Aplicações

Nesta seção são apresentados os gráficos que relacionam os tempos parciais e totais do processo de migração, com a largura de banda disponível para tal migração.

A.5.1 Tempo Total de Migração

A figura 1 apresenta o tempo total de migração, com compactação, identificado pela letra (**Z**), ou sem compactação, referido por (**NC**), para diferentes limitações de banda na conexão entre as máquinas fonte (10.10.40.70) e destino (10.10.40.40).

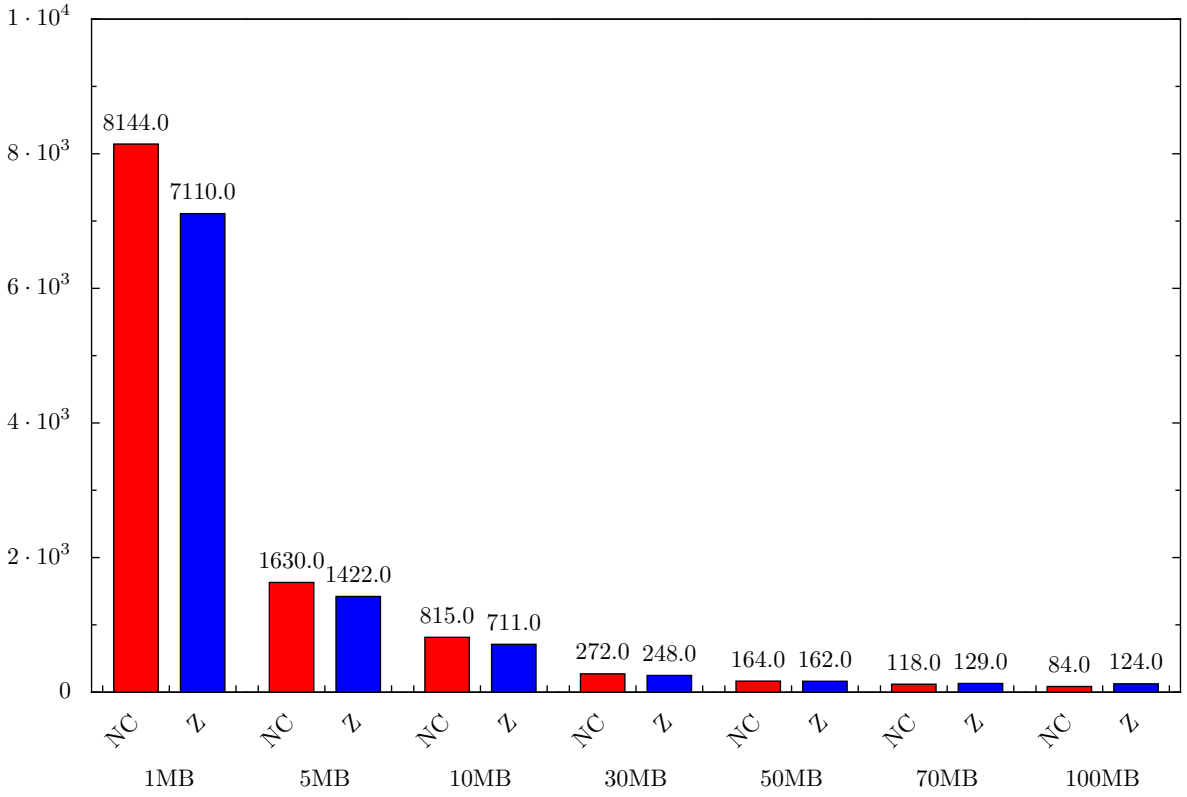

Figura A.1: Tempo Total de Migração: tempo(s) X Banda (MB)

A.5.2 Downtime

Na figura A.2 é apresentado o *downtime* ocorrido durante a migração da *MV*, com (**Z**) ou sem compactação (**NC**), para diferentes limitações de banda na conexão das máquinas fonte(10.10.40.70) e destino(10.10.40.40).

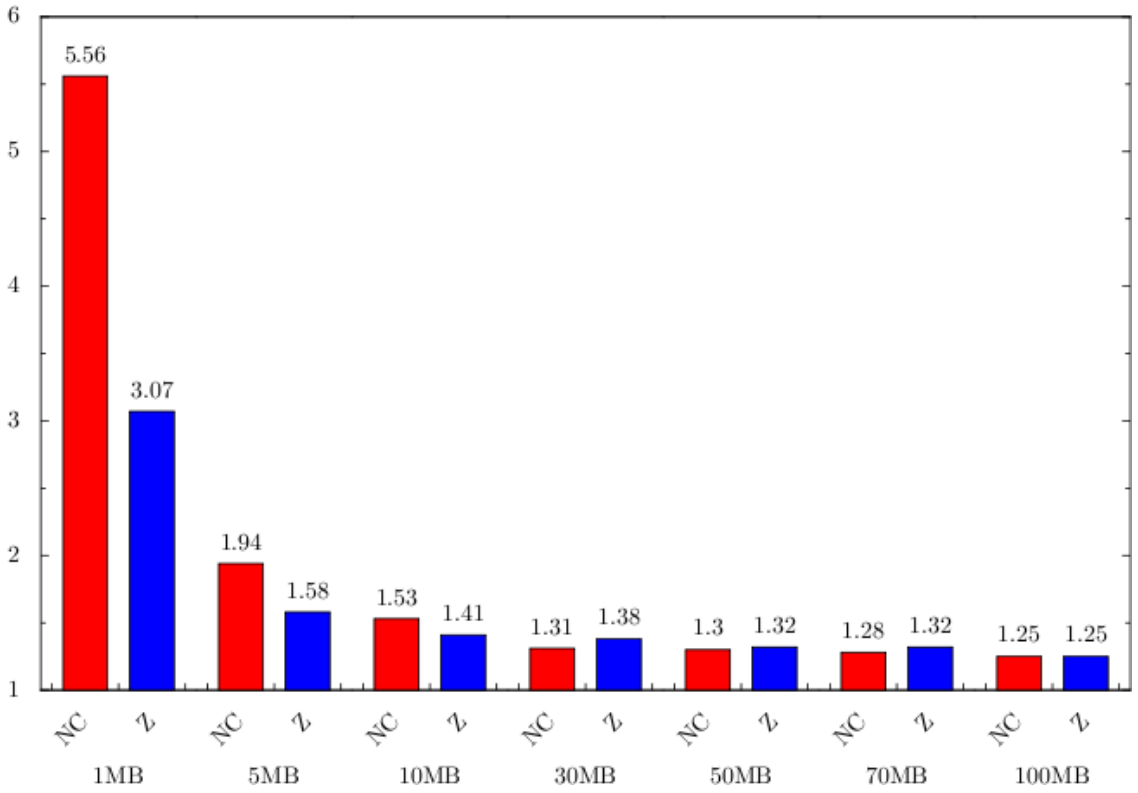

Figura A.2: Downtime: tempo(s) X Banda (MB)

A.6 Migração de MVs Com Aplicações Reais

Neste experimento, é apresentado com detalhes o tempo de *dowtime* na migração de uma MV. Durante todo período de migração, são executadas sobre as MVs, aplicações com características distintas. O tempo decorrente do *downtime*, é detalhado para cada uma delas, com o objetivo de identificar as fases críticas da *live migration* no OpenVZ, para estas aplicações. Aqui, a migração foi dividida em sete fases, são elas: *First Rsync*, *Suspend*, *Dump*, *Copy Dump File*, *Sencond Rsync*, *Undump* e *Resume*. Destas fases, as últimas seis são as que mais interessam neste momento, pois sua soma, informam o tempo de *downtime* da migração.

A.6.1 Descrição dos Experimentos

As máquinas virtuais a serem migradas são Debian e executam no Openvz. Para cada aplicação foram feitas cinco migrações sendo que, os resultados abaixo apresentados se referem a média dos resultados obtidos nestas cinco migrações. A seguir é apresentada uma breve descrição de cada uma das aplicações testadas.

Allocator Programa escrito em C que aloca 500MB de memória e escreve o número 1 em cada uma destas posições toda a memoria alocada. Esta aplicação utiliza muita memória, gerando um arquivo grande de *dump*.

Compiler Compilação do *kernel* linux 2.6. Compilação do *kernel* exige muito processamento.

Game Um servidor de jogos, Openarena-server, é executado na MV que será migrada, simultaneamente duas máquinas físicas da mesma rede rodam o OpenArena-client. O OpenArena é uma versão livre do Quake-3. Este tipo de jogo requer muita interatividade, muita troca de mensagens entre o servidor e o cliente, e esta troca de mensagens deve ocorrer em tempo real. Esta aplicação utiliza muito os recursos de rede.

Http-server A MV roda o servidor http, Apache2. Para simular a utilização deste servidor foi executado em uma outra máquina física da rede o httperf com o seguinte comando: 'httperf –server 10.10.40.237 –port 80 –uri /sites/ –rate 150 –num-conn 60000 –num-call 20'. Este comando diz que serão efetuadas conexões ao servidor 10.10.40.237 à uma taxa de 150 conexões por segundo, cada conexão fará 20 chamadas antes de ser fechada. Esta aplicação exige muito dos recursos de rede.

Mysql-server É executado na MV o servidor Mysql e o sql-bench. O comando sql-bench aqui executado cria e manipula tabelas do banco de dados. Estas operaçoes requerem processamento e memória.

A.6.2 Resultados

A tabela seguinte apresenta todos os dados da migração coletados para cada aplicação. As últimas seis linhas representam, sequencialmente, as fases na qual a MV está parada, ou seja, as fases da migração onde a aplicação não está sendo executada.

	Allocator	Compilator	Game	Http-Server	Mysql-Server
MV size (MB)	545	711	682	272	541
Dump size (MB)	503	16	21	5.7	55.4
1^o rsync (s)	13.44	18.84	13.82	7.32	14.62
Suspend (s)	0.06	0.03	0.02	0.59	0.05
Dump (s)	2.52	0.13	0.16	0.87	0.32
Copy Dump File (s)	11.10	0.51	0.76	0.24	1.34
2^0 rsync (s)	0.42	1.38	0.44	0.84	0.80
Undump (s)	1.00	0.73	0.38	0.31	1.03
Resume (s)	0.14	6.10	0.34	1.07	0.83
Dowtime (s)	15,24	8.08	2.1	3.92	4.37

A tabela mostra onde estão os gargalos da migração realizadas pelo OpenVZ. Enquanto o maior responsável, pelos *downtimes*, sofridos pelas aplicações Allocator e Game, foi a cópia do arquivo de *dump*, a maior parte do *dowtime*, sofrido pelas aplicações Compilator e Http-Server, foi por conta do tempo de *resume*.

A.7 Conclusão

Notou-se que os tempos de *downtime* são muito elevados, muitas vezes inaceitáveis para algumas aplicações, principalmente as que exigem interação em tempo real. Observou-se que um dos fatores responsáveis por esse tempo, é a ineficiência dos mecanismos de *dump*, *umdump* e *restart*. O outro fator, que é destacado quando se tem um MV que utiliza muita memória física, é o tempo de migração do arquivo de *Dump*.

Apêndice B

O Gerenciador de MVs e o Ambiente Experimental

Este apêndice expões os detalhes técnicos do gerenciador de máquinas virtuais. O apêndice, contempla também, os passos necessários para a instalação e configuração do ambiente experimental desta dissertação. A Seção B.1, apresenta detalhes da implementação do gerenciador e do algoritmo de balanceamento de máquinas virtuais, usado no gerenciador; a Seção B.2, apresenta os passos que devem seguidos para instalar o ambiente experimental. Na seção B.3.2, são apontadas a estrutura e as modificações realizadas para implementar o gerenciador de máquinas virtuais usado nos experimentos; por fim, a Seção B.4 apresenta uma dica, fruto da experiência adquirida com a montagem do ambiente experimental.

B.1 Detalhes do Algoritmo de Balanceamento

O Algoritmo de balanceamento implementado é uma versão para MVs do algoritmo Assign-U estendido [7], que foi concebido para balancear o sistema por meio de migrações de MVs. O pseudo algoritmo apresentado na listagem B.1, apresenta o *loop* do algoritmo que foi implementado em python. No *loop* principal, é verificado se existe alguma máquina em desequilíbrio, isto é, alguma máquina com CPU sobrecarregada em relação as demais. Caso exista tal máquina, é calculado qual máquina de destino que apresenta menor custo marginal. Então, uma máquina virtual é migrada.

O código do algoritmo de balanceamento implementado foi copiado para a pasta de *plugins* do gerenciador.

```
;MF  - representa o conjunto de maquinas virtuais do
   sistema.
;MV_i - representa o conjunto de maquinas virtuais da
   maquina i.
;migra(a,x,y) - migra a mv a da maquina x para a y.
;destino(v) - retorna a maquina com menor custo marginal.

for all i in MF do:
   for all v in MV_i do:
     for all j in MF, j not i do:
       if not equilibrio(i,j,v) do:
            l <- destino(v)
            migra(v,i,l)
            exit
       end if;
     end for;
  end for;
end for;
```

Listagem B.1: Algoritmo de Balanceamento Implementado

B.2 Instalação do Gerenciador de Máquinas Virtuais

Vale ressaltar, a dificuldade enfrentada para instalar este *framework*. Além de ser dependente de módulos que não são oficialmente suportados, o tutorial não está

completo e, aparentemente, estes problemas são novidades na comunidade. Os dois problemas enfrentados na instalação são:

1. Uma das ferramentas usadas pelo Usher, a libxenstat, não é oficialmente suportada pelo Xen. Eis a origem dos problemas seguintes. Como a biblioteca não é oficialmente suportada, é necessário compila-la a partir de um código fonte do Xen.

2. O segundo problema é decorrente do primeiro. Após a compilação e instalação da libxenstat, ocorre um erro relacionado a incompatibilidade entre a versão compilada da libxenstat e a versão instalada do Xen. Este erro é relatado pela comunidade do Xen, mas em um contesto diferente.

Os passos aqui descritos é um complemento ao tutorial de instalação oficial **??**, desenvolvidos para sobrepor a incompatibilidade com os módulos do Xen. Como a linguagem usada na implementação é interpretada, o gerenciador desenvolvido é instalado da mesma maneira que o Usher, só alterando os arquivos do gerenciador.

Abaixo, são enumerados os passos para a instalação do Xen, OpenVZ e do Gerenciador.

1. Instalar o CentOS 5.2 conforme sugerido no site do **??**.

2. *$mkdir /usr/src/redhat*

3. *$cd /usr/src/redhat/SOURCES*

4. *$rpm -i http://bits.xensource.com/oss-xen/release/3.2.0/centos-5.1/xen-3.2.0-0xs.centos5.src.rpm*

5. *Edite o arquivo: ../SPECS/xen.spec*

   - *Altere a versão: de "Version: 3.2.0"para "Version 3.2.1"*
   - *Altere a fonte: de "Source0: xen-3.2.0.tar.gz"para "Source0: xen-3.2.1.tar.gz"*
   - *Descomente a linha: "# /usr/lib/xen/boot/hvmloader"para "/usr/lib/-xen/boot/hvmloader"*

6. *$yum -y install transfig texi2html tetex-latex gtk2-devel libaio-devel gnutls-devel rpm-build.i386 (dependencies needed for rpmbuild of xen.spec in CentOS)*

7. *$cd /usr/src/redhat/SPECS ; rpmbuild -ba ./xen.spec*

8. *$yum install -y libidn-devel SDL-devel curl-devel python-devel ncurses-devel dev86 openssl-devel glibc-devel gcc gnutls-devel*

9. *Edite o arquivo:/etc/yum.conf*

    - *Altere "gpgcheck=0"para "gpgcheck=1".*

10. *$cd /usr/src/redhat/RPMS/i386; yum -y install xen-3.2.1-0xs.i386.rpm xen-libs-3.2.1-0xs.i386.rpm xen-devel-3.2.1-0xs.i386.rpm*

11. *Edite o arquivo: /etc/grub.conf*

    - *Altere a linha: "kernel"para "kernel /xen.gz-3.2*

12. *$reboot*

13. *$wget http://dag.wieers.com/rpm/packages/mercurial/mercurial-0.9.5-1.el5.rf.i386.rpm*

14. *$rpm -ivh mercurial-0.9.5-1.el5.rf.i386.rpm*

15. *$cd /usr/src/redhat/SOURCES/;tar -zxvf xen-3.2.0.tar.gz*

16. *$cd xen-3.2.0; make dist*

17. *$yum install swig*

18. *$cd tools/xenstat/libxenstat/*

19. *Edite o arquivo: /etc/grub.conf*

    - *Altere a linha: "PYTHON_VERSION=2.43" para "PYTHON_VERSION=2.4"*

20. *$make*

21. *$make python-bindings*

22. *$gcc -O2 -fomit-frame-pointer -m32 -march=i686 -DNDEBUG -std=gnu99 -Wall -Wstrict-prototypes -Wno-unused-value -Wdeclaration-after-statement -D___XEN_TOOLS___ -D_LARGEFILE_SOURCE -D_FILE_OFFSET_BITS=64 -D_LARGEFILE_SOURCE -D_LARGEFILE64_SOURCE -mno-tls-direct-seg-refs -Isrc -I../../../../tools/libxc -I../../../../tools/xenstore -Lsrc -L../../../../tools/xenstore/ -L../../../../tools/libxc/ -I/usr/include/python2.4 -lpython2.4 -shared -lxenstat xenstat.o xc_private.o xc_linux.o xc_domain.o xc_misc.o xs.o xs_lib.o xenstat_linux.o -o ../bindings/swig/python/_xenstat.so ../bindings/swig/python/_xenstat.c*

Para instalar o o gerenciador e o OpenVZ, pode ser seguido o procedimento de instalação sugerido em seus sites oficiais, respectivamente, *http://usher.ucsd.edu/trac/wiki/UsherDocumentation* e *http://wiki.openvz.org/Main_Page*.

B.3 Modificações no Gerenciador Base

Foram alteradas diversas funções para que o gerenciador suportasse os experimentos. Como o Usher ainda está incompleto e ainda não é implementado balanceamento de carga, foi necessário acrescentar alguns métodos ao seu código. O objetivo inicial era de trabalhar com o *framework*, Usher, como usuário deste, mas logo foi notado que seria impossível implementar balanceamento de carga desta forma. Foram então acrescentadas mais funcionalidades ao *framework*, com a finalidade de disponibilizar aos plugins todos os dados e métodos necessários para o balanceamento de carga.

B.3.1 Estrutura do Gerenciador

As modificações, acima citadas, foram facilitadas pelas modularidade do gerenciador base, e pelas características da linguagem de programação utilizada, que é interpretada e orientada a objeto. A pasta principal do gerenciador é composta dos seguintes diretórios:

Controle_Central Este diretório contém arquivos que serão executados somente no nó que servirá de controle central. Os arquivos deste diretório contém funções que manipulam os dados recebidos dos controles locais.

Controle_Local Este diretório contém arquivos que serão executados somente no nó que servira de controle local. Os arquivos deste diretório contém as funções responsáveis por interagir com as tecnologias de virtualização, a fim de coletar informações das máquinas físicas e virtuais. Sempre que o controle central determinar, as funções deste arquivo devem pedir para a tecnologia de virtualização migrar uma determinada MV.

Plugins Contém a inteligência do gerenciador. As funções deste diretório são responsáveis por decidir sobre as migrações. O algoritmo descrito na seção anterior, está codificado neste diretório.

B.3.2 Modificações

Uma das mais importantes funcionalidades adicionadas, foi a criação de mais um evento que é disparado periodicamente "acordando" a função do *plugin* e passando

para esse, o objeto controle. O objeto de controle possui todos os outros objetos relacionados ao controle central. Assim, a função tem a sua disposição os métodos que coletam os dados necessários para definir se alguma migração deve ser efetuada, caso alguma migração seja necessária, um dos métodos implementados no objeto controle será capaz de efetuar tal migração, também estará disponível no objeto de controle. Foi também adicionado ao controle central, um método de coleta periódica, que solicita a todos os controles locais, os dados sobre a utilização de CPU de suas máquinas. Para satisfazer tal requisição, foi implementado um método em uma função do diretório de controle local, para que este repasse ao controle, tais informações. Outro método periódico foi acrescentado ao controle. O objetivo deste outro método é atualizar as informações sobre a carga de CPU e memória das máquinas virtuais gerenciadas.

Foi verificado uma incoerência na nomenclatura das MVs. Sempre que o controle central renomeava uma MV, a renomeação ocorria na tecnologia de virtualização e no controle central, mas não ocorria no controle local. Uma forma de correção deste erro, que foi implementada, é não permitir que o controle central renomeie as MVs. Essa opção foi a escolhida, pois para os objetivos dos experimentos, tais renomeações são supérfluas e prejudiciais.

Para que o gerenciador também suportasse a tecnologia de virtualização OpenVZ, foi necessário adicionar ao módulo do gerenciador local, funções com a capacidade de reconhecer as máquinas virtuais hospedadas em sua máquina, ler na máquina a carga gerada por cada máquina virtual, executar comando de migração de uma determinada máquina virtual, entre outras.

B.4 Dica

Após a montagem de todo ambiente experimental. Foi verificado um problema que ocorreu tanto para o Xen quanto para o OpenVZ. Sempre que ocorria uma migração de uma MV cuja data da máquina de origem era maior que a data da máquina de destino, o processamento de qualquer aplicação desta MV ficava extremamente lendo.

Para resolver foi instalado um servidor de horas (NTP) para que a data de todas as máquinas envolvidas no experimento fosse sempre a mesma.

Lista de Abreviaturas

Algoritmo de Aproximação	Em linhas gerais, Algoritmos de aproximação são algoritmos que não necessariamente produzem uma solução ótima, mas soluções que estão dentro de um certo fator da solução ótima., 30
CFQ	*Completely Fair Queuing* é o escalonador de E/S usado no *kernel* do Linux. A ideia deste escalonador é limitar a vazão de E/S de um determinado processo, de acordo com a prioridade pré estabelecida a este processo., 25
Cloud Computing	Termo inglês que cuja tradução é Computação na Nuvem. Computação na Nuvem é um modelo de computação onde o processamento, o armazenamento e o software, são acessados remotamente e estão em algum lugar não necessariamente conhecido por quem os usa., 1
Cluster	Aglomerado de computadores, é formado por um conjunto de computadores interligados que podem trabalhar junto, podendo ser visto como um único computador., 1, 20, 22, 55, 60
Container	*Conteiner* é como pode ser chamada uma máquina virtual da classe de virtualização no nível do SO., 17, 26, 27, 57, 68
Credit scheduler	Em português, escalonador por créditos. Este escalonador atribui créditos as VCPUs, e a prioridade de execução destas VCPUs, pode estar relacionado com a quantidade de crédito atribuída., 24

Downtime	Tempo que uma máquina virtual, em migração, permanece sem executar. Este tempo geralmente ocorre no momento em que a máquina virtual, que está sendo migrada, deixa de executar na origem e passa a executar no destino., 26, 48, 55
E/S	Entrada/Saida, ou em inglês,*IO*, representa operações de entrada ou saída de dados por meio de um software ou hardware. São exemplos de unidades de saída de um computador: monitor, caixas de som, impressora, disco rígido., 15
Fair-Share	Fair-share é um algoritmo de escalonamento. A estratégia deste algoritmo é que cada processo receba determinada fração do poder de processamento da CPU., 24
HPC	HPC (*High-performance computing*) é um termo inglês, aplicado para designar super computadores ou *clusers* de alto desempenho. O uso dos HPC geralmente está relacionado com processamento de pesquisas científicas., 67
IA-32	Termo inglês, Intel Architecture 32-bit, que se refere ao conjunto de instrução do processador da Intel., 6, 14
ISA	Instruction Set Arquitecture. Interface entre a última camada de software e o hardware., 8
Job	Termo inglês que descreve uma tarefa ou um processo que deve ser alocado a uma máquina., 42

Jogos interativos on-line	Neste classe de jogos, a máquina dos jogadores recebem e enviam dados para um servidor remoto que geralmente está na *wan*. Jogos iterativos exigem baixa latência na troca de mensagens ponto-a-ponto., 1
Kernel	Termo inglês que significa núcleo, é o componente central do SO da maioria dos computadores; ele serve de ponte entre aplicativos e o processamento real de dados feito a nível de hardware, 17, 23, 38
Live Migration	Termo usado para se referir a um tipo de migração cujo *downtime* é geralmente imperceptível., 54
Migração de máquina virtual	A migração de máquinas virtuais consiste na migração de instancias de SOs entre máquinas., 3, 21, 27, 36, 59
MV	Máquina virtual de sistema., 2, 21, 22, 26, 53, 60
MVs	Máquinas virtuais de sistema., 2, 21, 22, 26, 53, 60
OpenVZ	OpenVZ é uma tecnologia de virtualização em nível de sistema operacional baseada no sistema operacional e núcleo Linux. O OpenVZ é uma base do Parallels Virtuozzo Containers, um software proprietário fornecido pela Parallels. O OpenVZ é licenciado sob a versão 2 da GPL., 23, 45, 60
Overhead	Overhead é a quantidade extra de processamento que determinada tecnologia necessita para funcionar, 4, 25, 49
Overhead de Migração	*Overhead* de Migração é a quantidade extra de processamento que uma determinada tecnologia de virtualização tem que usar a fim de migrar uma máquina virtual., 34, 41, 59

Reboot	Termo inglês para se referir a reinicialização do SO., 8
SO	Sistema operacional., 6, 14, 21, 56
Swap	Termo em inglês que significa troca ou paginação. *Swap* é a troca de dados entre o disco rígido e a memória principal. O *swap* ocorre quando não existe mais memória física disponível para um processo. O SO realiza *swap* pra tentar estender virtualmente a memória física da máquina.., 51
TC	Traffic Control. Ferramenta usada para realizar controle de tráfego da interface de rede., 70
Tempo total de migração	Tempo decorrido desde o início até o fim da migração da máquina virtual., 20, 27, 36, 61
Vazão	Em inglês, *Througput*. Vazão é a quantidade de dados transferidos de um lugar a outro, ou a quantidade de dados processados em um determinado espaço de tempo, pode-se usar o termo vazão para referir-se a quantidade de dados transferidos em discos rígidos ou em uma rede, por exemplo, 2
VCPU	VCPU (Virtual CPU), são as CPUs virtuais que são atribuídas as máquinas virtuais do Xen. Cada MV possui um ou mais VCPU. Existem filas de CPU para executar nas CPUs reais., 24
VMware	VMware é um software de virtualização que implementa virtualização total. A empresa desenvolvedora do VMware, a VMware Inc., 14, 58

WWS	WWS (*Writable Working Set*) é um termo inglês, usado para referenciar as páginas de memórias que foram escritas durante uma das fase de migração por pré-cópia, e que devem ser reenviadas para o destino., 57
Xen	Xen é um software livre de virtualização para as arquiteturas x86, x86-64, IA-32, IA-64 e PowerPC. O Xen implementa a técnica de paravirtualização.Xen foi originalmente como um projeto de pesquisa na Universidade de Cambridge, liderado por Ian Pratt, fundador da XenSource, Inc. Em 15 de agosto de 2007, a XenSource foi adquirida pela Citrix System Inc., 14, 24, 46, 59