Pontifícia Universidade Católica do Rio Grande do Sul
Faculdade de Informática
Programa de Pós-Graduação em Ciência da Computação

ALOCAÇÃO DINÂMICA DE RECURSOS NO XEN

Fábio Diniz Rossi

Dissertação apresentada como requisito parcial à obtenção do grau de mestre em Ciência da Computação

Orientador: Prof. Dr. Avelino Francisco Zorzo

Porto Alegre
2008

Dados Internacionais de Catalogação na Publicação (CIP)

R831a Rossi, Fábio Diniz
Alocação dinâmica de recursos no Xen / Fábio Diniz Rossi. – Porto Alegre, 2008.
68 f.

Diss. (Mestrado) – Fac. de Informática, PUCRS
Orientador: Prof. Dr. Avelino Francisco Zorzo

1. Informática. 2. Sistemas Operacionais (Computação). 3. Processamento Paralelo. 4. Redes de Computadores – Gerência. 5. Software. I. Zorzo, Avelino Francisco. II. Título.

CDD 005.43

Ficha Catalográfica elaborada pelo
Setor de Tratamento da Informação da BC-PUCRS

Pontifícia Universidade Católica do Rio Grande do Sul
FACULDADE DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

TERMO DE APRESENTAÇÃO DE DISSERTAÇÃO DE MESTRADO

Dissertação intitulada "**Alocação Dinâmica de Recursos no Xen**", apresentada por Fábio Diniz Rossi, como parte dos requisitos para obtenção do grau de Mestre em Ciência da Computação, Ciência da Computação, aprovada em 15/01/08 pela Comissão Examinadora:

Prof. Dr. Avelino Francisco Zorzo – Orientador — PPGCC/PUCRS

Prof. Dr. César Augusto Fonticielha De Rose – — PPGCC/PUCRS

Prof. Dr. Antonio Marinho Pilla Barcellos – — UNISINOS

Homologada em 04/08/08, conforme Ata No. 16/08 pela Comissão Coordenadora.

Prof. Dr. Fernando Gehm Moraes
Coordenador.

PUCRS
Campus Central
Av. Ipiranga, 6681 – P32 – sala 507 – CEP: 90619-900
Fone: (51) 3320-3611 – Fax (51) 3320-3621
E-mail: ppgcc@inf.pucrs.br
www.pucrs.br/facin/pos

Agradecimentos

Foi uma longa caminhada. Tenho certeza de que muitos me ajudaram no decorrer desse percurso e que, no processo de lhes agradecer, alguns serão inevitavelmente esquecidos. A estes últimos peço desde já as minhas mais sinceras desculpas e dou meu muito obrigado.

Primeiramente agradeço a Deus, fonte de toda a vida.

Agradeço a minha família; em especial aos meus pais, por terem me proporcionado sempre o apoio necessário em minhas empreitadas, espero corresponder sempre as suas expectativas; e as minhas irmãs, sempre presentes nas minhas orações.

Agradeço a minha noiva Andréia pelo amor, compreensão e paciência nestes dois anos de distância. Te amo muito.

Agradeço muito ao meu orientandor Avelino Zorzo, que mesmo sem conhecer o meu trabalho, confiou a oportunidade de sua orientação. Obrigado pelos ensinamentos, experiência, dedicação e pela amizade. Estou certo que nossa convivência me proporcionou crescer pessoal e profissionalmente.

Agradeço muito aos meus colegas e amigos do mestrado: Mauro Storch, Rafael Antonioli, Guilherme Rodrigues, Ana Winck, Jeferson Prevedello, Felipe Franciosi, Tiago Silva, que compartilharam dificuldades, partilharam idéias e fomentaram discussões, sempre prontos para "varar a madrugada"quando necessário em busca de resoluções. Rogo a Deus que ilumine todos vocês.

Em especial, agradeço ao amigo e colega Guilherme Rodrigues, que assim como eu, largou um emprego e uma vida, em busca do mesmo ideal intelectual. Obrigado por tudo.

Agradeço também aos amigos e colegas do doutorado: Cristina Nunes, Rodrigo Calheiros, Luciano Ost, Fábio Delamare, que em meio as mais inúmeras atividades, ainda tinham tempo para nos guiarem com sua dedicação, incentivo, convívio e conhecimento compartilhado. Meus sinceros agradecimentos a vocês.

Agradeço aos amigos e colegas de pesquisa: Juliano Potrich, Rodrigo Tolledo e Dionatan Korb, vocês foram imprescindíveis para a realização desta dissertação. Torço para que sejam felizes nas suas escolhas, e tenho certeza de que serão.

Meu muito obrigado a HP (Hewlett&Packard) e a PUCRS, que por meio de concessão de bolsa de estudos, possibilitou que eu pudesse realizar este curso de mestrado, através dos projetos PeSO e CPPH/PUCRS.

Mais um caminho finalizado, surgem novos caminhos a trilhar. Espero que nesses novos caminhos eu tome decisões tão acertadas quanto a realização do curso de mestrado, que me proporcionou um aprendizado efetivo através da vivência em um ambiente de pesquisa de alta qualidade. Novamente obrigado a todos.

Resumo

A demanda por poder computacional é cada vez maior, e conseqüentemente leva ao desenvolvimento de equipamentos com capacidades de processamento cada vez maiores para suprí-la. Hoje em dia temos acesso a computadores com poder computacional cada vez maior, porém em sua grande maioria, esse poder computacional é apenas parcialmente utilizado, causando uma ociosidade dos recursos disponíveis, o que pode acarretar aumento de custos.

Ao analisarmos a situação do aumento do poder computacional, a idéia de ociosidade de processamento da maioria dos computadores e em contrapartida, a possibilidade de uma melhor utilização de recursos, podemos explicar a renovação de tecnologias que podem vir a suprir essas necessidades.

Entre várias destas tecnologias podemos citar *clusters* e grades computacionais, e entre outras, uma das tecnologias em ascensão são as máquinas virtuais. Uma máquina virtual consiste em um *software* que cria um ambiente sobre um sistema operacional, possibilitando uma execução abstraída do *hardware* de vários outros sistemas, sendo transparente para o usuário essa interação.

Dentre vários ambientes que suportam o uso de máquinas virtuais, utilizamos o Xen que proporciona a portabilidade de sistemas operacionais sobre um sistema operacional Linux e permite compartilhar uma simples máquina para vários clientes rodando sistemas operacionais distintos. O Xen pode utilizar um de três escalonadores, onde o *SMP Credit Scheduler* é o escalonador padrão, recomendado para máquinas multiprocessadas por permitir balanceamento de processadores virtuais entre os processadores reais. Porém, o *SMP Credit Scheduler* ainda tem algumas limitações referentes a uma utilização melhor dos recursos da máquina.

Com o objetivo de superar algumas dessas limitações, este trabalho apresenta a proposta e implementação de um subsistema que altera dinamicamente configurações do escalonador *SMP Credit*, realocando recursos destinados a máquinas virtuais que não estejam utilizando todo o processamento disponível, direcionado-as às máquinas virtuais que necessitem desse processamento. Por fim, apresentamos uma avaliação do uso desse subsistema frente ao escalonador *SMP Credit* em diversas configurações possíveis.

Palavras-chave: Máquinas virtuais, Xen, escalonamento, alocação de recursos.

Abstract

The demand for computer processing power has increased in the past years, resulting in computers that provide such capacity. Sometimes different approaches have also being developed to improve computing power by joining together a set of computers, for example in clusters of computers. Currently we have access to this type of solutions but we do not use all their computing power the best way we could. This may lead to a situation in which resources are being wasted.

In order to avoid the waste of computing resources, lately the use of virtual machines have been widely used. A virtual machines is a software layer that creates an environment in which several systems can be executed as if they had their own private computer. One solution that allows this approach is Xen. Xen is a paravirtualizer that allows several different operating systems to run as if they were using different computers. The scheduling of the different operating systems that are running in the same computer is performed by one of three possible strategies provided by Xen. The standard scheduler is called "SMP Credit Scheduler", which is recommended when running Xen on multiprocessing computers because it allows load balancing among virtual and real processors. Despite being the best current Xen scheduler, the SMP Credit Scheduler still does not fully use the computing power of a machine.

This work proposes to improve the use of the machine by the operating systems (virtual machines) that are running on Xen. We propose a system that dynamically changes the configuration of the virtual machines. Our system will reallocate resources that are not being used by a virtual machine to a virtual machine that needs more resources.

Keywords: Virtual machines, Xen, scheduling, resource allocation.

Lista de Figuras

Lista de Tabelas

Lista de Siglas

ABI	*Application Binary Interface*	21
ISA	*Instruction Set Architecture*	21
VMM	*Virtual Machine Monitor*	22
CPU	*Central Processor Unit*	22
MMU	*Memory Management Unit*	27
NUMA	*Non-Uniform Memory Access*	30
SMP	*Symmetric Multiprocessor*	33
VCPU	*Virtual CPU*	33
SLA	*Service Level Agreement*	37
POSIX	*Portable Operation System Interface*	40
XML	*Extensible Markup Language*	46
CAP	*Capability*	52
RAM	*Read-Only Memory*	54
SLO	*Service Level Objective*	58

Sumário

1 Introdução

Na década de 60, a IBM introduziu no seu modelo S/370 a possibilidade de compartilhar esse computador com a utilização de mais de um sistema operacional ao mesmo tempo, através de uma técnica nova chamada virtualização [1].

Hoje em dia, com o aumento do poder computacional, a utilização de máquinas virtuais possibilita compatibilidade, desempenho e simplicidade. Compatibilidade no sentido de poder executar qualquer *software* em qualquer ambiente computacional; desempenho porque a virtualização aproveita ciclos ociosos da máquina; e simplicidade na manutenção dos sistemas virtualizados.

Outro grande ponto a favor da virtualização é a redução de custos, pois *datacenters* têm como objetivo gerenciar servidores e a utilização da virtualização reduz custos de modo que servidores subutilizados por determinadas aplicações podem alocar outras aplicações concomitantemente, reduzindo a utilização de energia, manutenção, etc. [2]. Essa solução de utilizar ao máximo os servidores alocando sistemas em máquinas virtuais é chamada consolidação de servidores.

Porém, a virtualização não está restrita à consolidação de servidores pois hoje em dia está presente desde aplicações de pequeno porte, até ambientes de alto desempenho. Em ambientes de pequeno porte, novas tecnologias tem utilizado a virtualização para criar ambientes portáveis em dispositivos como *Thinstall* [3], *MojoPac* [4] e *Moka5* [5], que possibilitam a criação de ambientes virtualizados em dispositivos portáteis como *pendrives*.

Ainda, ambientes de grande porte também podem ser virtualizados, como é o caso de *clusters* [6] e *grids* [7], o que possibilitam testar aplicações em nodos heterogêneos ou mesmo simular o comportamento de um *cluster* ou *grid* em menor escala com determinada aplicação.

Como vemos atualmente, computadores pessoais possuem algumas características, por exemplo mais de um processador, de ambientes de alto desempenho como *clusters* e *grids*, que propõem aumento de desempenho através do incremento do número de processadores ou *cores*. A virtualização também está presente nesse nicho, pois possibilita virtualização em nível de *hardware* [8], através dos processadores *Intel Vanderpool* e *AMD Pacifica* que permitem dividir uma mesma máquina física em diversas máquinas virtuais, com a vantagem de não ser necessário modificar a configuração dos sistemas operacionais convidados.

Esta dissertação utiliza como foco principal o monitor de máquinas virtuais Xen, que se difere de outras técnicas de virtualização, uma vez que o Xen não interpreta as instruções passadas ao *hardware* como faria um emulador, o que acarretaria uma perda no desempenho [9]. O Xen apenas se encarrega de repassar as instruções ao sistema principal, fornecendo uma

transparência ao sistema convidado de estar sendo executado diretamente sobre o *hardware*.

Embora o Xen ser utilizado em diversas situações, ele ainda apresenta algumas limitações, por exemplo: não possibilita limites de porcentagem máxima/mínima de fatias de processador para máquinas virtuais individuais; não possibilita limites de procentagem máxima/mínima de fatias de memória para máquinas virtuais individuais; não permite a utilização de níveis de acordo de serviços sobre máquinas virtuais; e principalmente não possibilita balanceamento de recursos entre máquinas virtuais.

Visando solucionar a última limitação mencionada acima, essa dissertação apresentará a proposta, implementação e avaliação de um subsistema de alocação dinâmica de recursos, que executará juntamente com o escalonador padrão (*Credit Scheduler*) do Xen [10].

A realocação dinâmica de recurso será feita a partir da análise de uma árvore de decisão gerada por um processo de mineração de dados.

Essa dissertação está organizada da seguinte forma. O Capítulo 2 apresenta conceitos importantes sobre máquinas virtuais, bem como seus tipos, técnicas e usos; o Capítulo 3 apresenta o monitor de máquinas virtuais Xen, foco principal deste trabalho, bem como sua gerência do processador, memória e seu comportamento em máquinas SMP (*Simmetric Multiprocessor*); o Capítulo 4 apresenta o subsistema de realocação de recursos proposto, descrevendo a sua implementação. Ainda, vemos o estado da arte sobre monitores/atuadores em sistemas operacionais, mostrando taxonomia, utilizações e alguns trabalhos relacionados; o Capítulo 5 apresenta toda a metodologia de validação do subsistema proposto; o Capítulo 6 traz conclusões do trabalho e aponta para possíveis trabalhos futuros.

2 Virtualização

A virtualização é uma metodologia que divide os recursos computacionais em múltiplas execuções, criando múltiplas partições, isoladas umas das outras, chamadas máquinas virtuais ou servidores virtuais privados, unidos em um único servidor físico.

A utilização da virtualização possibilita um ambiente mais dinâmico e flexível, onde se pode executar cargas de trabalho em um número menor de sistemas físicos, facilitando suporte e manutenção. Isso não significa que o aumento no custo de aquisição de *hardware* com grande poder de processamento não vai mais existir, pois para suportar sistemas virtualizados robustos é também necessário um *hardware* com desempenho suficiente para responder a inúmeras requisições.

As máquinas virtuais surgiram na década de 60, no IBM S/370, onde a IBM fez um modelo em que cada máquina virtual era uma cópia exata de uma máquina real, porém com uma capacidade de memória reduzida, e com essa noção, um computador poderia ser dividido em várias máquinas virtuais leves, utilizando recursos tanto quanto o original [11].

Existem vários tipos de máquinas virtuais, com diferentes objetivos e implementações. As máquinas virtuais são divididas primeiramente entre máquinas virtuais de processos (*ABI VMs - Application Binary Interface Virtual Machines*) e máquinas virtuais de sistema (*ISA VMs - Instruction Set Architecture Virtual Machines*) [12].

A diferença entre máquinas virtuais de processos e máquinas virtuais de sistema é a persistência. Uma máquina virtual de processo é um ambiente que executa um processo individual em particular, e existe apenas para suportar este processo em específico.

Uma ABI permite a um programa acessar os recursos do *hardware* e os serviços disponíveis através da interface de chamadas do sistema. Uma ABI não inclui instruções de sistema, portanto toda aplicação interage com os recursos de *hardware* indiretamente pela invocação de serviços do sistema operacional via interface de chamadas de sistema. As chamadas de sistema provém uma maneira para o sistema operacional executar operações de interesse dos programas do usuário, validando sua autenticidade e segurança.

Assim como na emulação, onde o caminho mais simples é a interpretação, o maior desafio de máquinas virtuais de processos é o suporte a binários de programas compilados para instruções diferentes daquelas que o *host* executa. Para uma melhoria no desempenho podemos nos utilizar de tradução binária dinâmica, que converte instruções do sistema convidado para instruções do *host*, utilizando blocos de instrução para instrução, salvando em *cache* para reuso. Repetidas execuções e instruções traduzidas diminuem a sobrecarga da tradução. Já uma máquina virtual de sistema provê um completo ambiente que suporta a um sistema operacional

com seus vários processos em execução.

Uma ISA marca a divisão entre *hardware* e *software*, onde a área de usuário inclui aspectos referentes às aplicações e a área de sistema em gerenciar os recursos do *hardware*. Para o usuário, todos os tipos de máquinas virtuais proporcionam essencialmente as mesmas funções, porém elas são diferentes na sua implementação. Classicamente, os monitores de máquinas virtuais (VMM - *Virtual Machine Monitor*) rodam no modo mais privilegiado do sistema operacional, enquanto seus sistemas convidados rodam com privilégios reduzidos, onde o VMM pode interceptar e emular todas as ações dos sistemas convidados que normalmente tenham que acessar ou manipular recursos de *hardware*.

Em uma visão de alto nível, podemos dividir a virtualização em suas técnicas, como veremos a seguir.

2.1 Técnicas de Virtualização

A virtualização está sendo utilizada nas mais diversas áreas, e dependendo do *hardware* ou da necessidade do *software* que deve ser utilizado, as técnicas de virtualização podem variar. Portanto, existem similaridades entre as técnicas de virtualização, onde a diferença entre elas está no nível de abstração e nos métodos usados para a virtualização [13].

As três técnicas de virtualização são: emulação, virtualização e paravirtualização. As três técnicas apresentadas diferem na complexidade da implementação, suporte ao sistema operacional, desempenho e nível de acesso aos recursos comuns.

2.1.1 Emulação

Nesta técnica (virtualização do *hardware*) é emulado um *hardware* real ou fictício, requerendo os recursos reais quando necessários, da máquina que roda a máquina virtual [14].

Normalmente em um emulador de sistemas é permitido rodar um sistema operacional sem a necessidade de modificações por que o sistema operacional não está ciente que não está rodando em um *hardware* real. As instruções privilegiadas requeridas à CPU (*Central Processor Unit*) não podem ser executadas pelo usuário, sendo requeridas ao monitor da máquina virtual, que analisará a execução do código e o fará em modo protegido. A emulação de *hardware* é utilizada pelo VMWare [15].

Podemos notar na Figura 1 que o emulador abstrai das aplicações o *hardware* real, provendo uma camada virtual que simula o *hardware* necessário para as aplicações. Vemos que, tanto a camada de emulação quanto a camada de *hardware virtual* fazem parte de uma mesma camada, apenas separadas pelas suas características, porém ambas rodando sobre o sistema operacional e, proporcionando uma camada base às aplicações.

Figura 1 – Estrutura da Emulação.

2.1.2 Virtualização

A virtualização é feita em nível de sistema operacional o que proporciona uma maior segurança de acesso entre máquinas virtuais. A maioria das aplicações rodam em um servidor e poderiam facilmente ser compartilhadas com outras máquinas, desde que existisse segurança. Em muitas situações, sistemas operacionais diferentes não são necessários no mesmo servidor, meramente múltiplas instâncias de um único sistema operacional.

Os sistemas operacionais de virtualização devem prover isolamento das requisições e segurança para rodar múltiplas aplicações ou cópias do mesmo sistema operacional, no mesmo servidor [16]. O OpenVZ é um exemplo de sistema de virtualização.

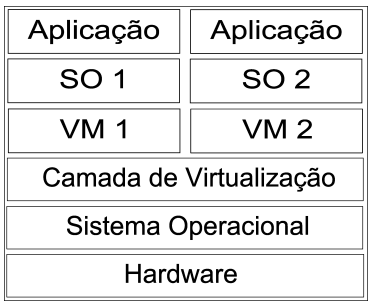

Figura 2 – Estrutura da Virtualização.

Na Figura 2, vemos que embora a virtualização também seja uma camada de abstração como a emulação, ela roda juntamente com o sistema operacional, e repassa instruções para o *hardware* real. Acima dessa camada rodam as máquinas virtuais (VM 1, VM 2) que suportam os sistemas operacionais convidados (SO 1, SO 2) independentes um do outro, que por sua vez suportam aplicações também independentes.

2.1.3 Paravirtualização

A paravirtualização é uma técnica que também necessita de uma máquina virtual, porém a maior parte da carga de trabalho é executada no código do sistema operacional convidado, que é modificado para suportar a máquina virtual, controlando assim o uso de instruções privilegiadas [17]. A paravirtualização também permite rodar diferentes sistemas operacionais em um único servidor e temos como exemplo o Xen e User-Mode Linux.

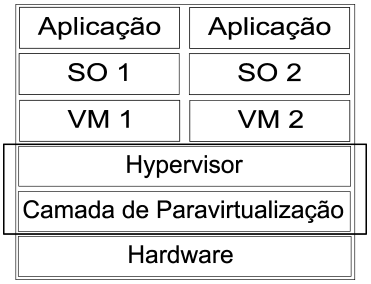

Figura 3 – Estrutura da Paravirtualização.

Na paravirtualização, a camada que da suporte à virtualização (camada de paravirtualização) é o próprio sistema operacional modificado para esse fim, e como vemos na Figura 3, o *hypervisor* é a camada de monitoria que roda em conjunto à camada de paravirtualização, fazendo interface da comunicação entre as máquinas virtuais e o *hardware*.

A paravirtualização não simula um novo *hardware* para as aplicações, porém através de modificações no *kernel* possibilita que as chamadas de sistema que se relacionam com o *hardware* sejam controladas pelo *hypervisor*.

Na técnica de virtualização existe uma camada que controla a comunicação entre as máquinas virtuais e o sistema operacional, enquanto na paravirtualização, essa camada de virtualização é o próprio sistema operacional modificado para hospedar máquinas virtuais.

2.2 Softwares Mais Usados em Virtualização

Com o crescimento da área de virtualização, cada vez mais surgem ferramentas que fornecem à sua maneira ambientes virtualizados para as mais diversas aplicações e sistemas. Dentre elas, existem soluções livres ou proprietárias, que concorrem pelo mercado de consolidação de servidores das grandes empresas que estão cada vez mais se utilizando dessa tecnologia.

Para suprir essa necessidade, estão surgindo novos sistemas que dão suporte a utilização de máquinas virtuais, que lutam por uma fatia do mercado empresarial de tecnologia da informação.

Nesta seção podemos acompanhar os sistemas virtualizadores atualmente mais utilizados pelo mercado.

2.2.1 VMWare

O VMWare permite a emulação de vários sistemas operacionais ao mesmo tempo sobre um sistema virtualizado onde cada máquina virtual funciona como um computador inteiro, contendo processador, memória, disco, vídeo, som, unidades de disquete e cd-rom [18].

Um ponto muito interessante sobre a uilização do VMWare é a possibilidade de ligação entre o sistema *host* e todas as máquinas virtuais como se estivessem em uma rede tradicional, estando cada qual com seu endereço IP.

O VMWare é um *software* que cria máquinas virtuais que virtualizam um computador pessoal completo, proporcionando a utilização de sistemas operacionais para a plataforma x86 [15].

2.2.2 User-Mode Linux

O User-Mode Linux é uma implementação do *kernel* (versão 2.6 x86) do Linux onde a arquitetura suportada é outro *kernel* do Linux [19]. Normalmente, os programas aplicativos fazem suas requisições ao kernel, que por sua vez aciona os recursos de *hardware* da máquina conforme solicitado (sem esquecer que este *kernel* sempre é específico para a arquitetura do computador em questão).

Mas no caso do UML, os programas do usuário fazem requisições ao *kernel* e, embora eles não suspeitem disso, este *kernel* vai passar todas as requisições para o *kernel* da máquina hospedeira, e este sim vai fazer a comunicação com o *hardware*.

2.2.3 FAUMachine

A FAUmachine funciona como um processo normal do usuário (nenhum módulo dos privilégios ou do *kernel* são necessários) no sistema operacional Linux. Na configuração tradicional, é usado um *bootloader* adaptado e um *kernel* ligeiramente modificado do Linux [20].

A camada de abstração do *hardware* da máquina virtual FAUmachine é (na maior parte) o *kernel* Linux do sistema anfitrião. Os dispositivos de leitura do *hardware* que a FAUmachine utiliza, inclui tamanho de memória principal, o *cd-rom* e o número e o tamanho dos discos rígidos que podem ser configurados.

A máquina virtual do FAUmachine é usada como plataforma no projeto europeu DBench (Dependable *benchmarking*), que desenvolve *benchmarks* para verificar a confiabilidade de sis-

temas operacionais.

2.2.4 OpenVZ

A arquitetura do OpenVZ [21] é diferente da arquitetura virtual tradicional das máquinas virtuais porque funciona sempre no mesmo *kernel* do sistema operacional anfitrião. Esta tecnologia permite execução em um único *kernel*, permitindo que usuários virtuais rodem suas aplicações com perda de desempenho pouco significativas.

Do ponto de vista das aplicações que rodam no OpenVZ, cada sistema virtual é um sistema independente. Esta independência é fornecida por uma camada do virtualização no *kernel* do sistema operacional anfitrião e onde somente uma parte insignificante dos recursos do processador central está utilizando esta virtualização (ao redor 1-2%).

Todo o sistema virtual comporta-se como um sistema Linux. Tem inicialização padrão; vários *softwares* podem funcionar dentro de um sistema virtual sem modificações especificas do OpenVZ ou ajustes; um usuário pode mudar toda a sua configuração e instalar algum *software* adicional; os ambientes virtuais dos usuários são isolados completamente (sistema de arquivos, processos, IPC, sysctl , etc).

2.2.5 Virtual PC

Com o crescente interesse na área de virtualização, a Microsoft lançou em 2 de dezembro de 2003 o Virtual PC, que permite rodar diversos sistemas operacionais simultaneamente.

O Virtual PC foi projetado para fazer uso das vantagens da tecnologia nova de Intel chamada *Intel Virtualization*, assim aumentando o desempenho do sistema convidado.

A Microsoft possui dois produtos de virtualização hoje no mercado, que são o Virtual PC para sistema operacional cliente (Windows XP), e o Virtual Server 2005, e sua instalação pode ser realizada apenas em sistemas operacionais servidores como (Windows 2000 Server e Windows 2003) [22].

2.2.6 Plex86

O Plex86 tem as mesmas características do Vmware, mas suporta apenas Linux como anfitrião e como hóspede. Isso torna a virtualização em arquitetura IBM-PC bem mais fácil, pois o Linux possibilita uma ótima plataforma de comunicaçãp com o *hardware* (e o código-fonte aberto permite a monitoração do acesso ao *hardware*) [23].

O *kernel* Linux do hóspede precisa ser recompilado para ser virtualizável. Na verdade, em

sua forma atual, ele se parece mais com o *User-Mode Linux*, que já está mais maduro e é mais bem suportado pela comunidade.

2.2.7 Bochs

O Bochs é um emulador de computadores pessoais x86 bastante completo, *open source*, escrito em C++, multiplataforma e possui uma boa documentação. [24].

Porém, o Bochs ainda não consegue ser viável para os usuários que precisem de uma solução para o *software* legado da plataforma Windows. Infelizmente, para estes usuários, ainda é necessário recorrer a *softwares* proprietários.

Contudo, com a evolução constante na velocidade dos processadores, talvez este rendimento baixo acabe não sendo tão significativo futuramente, além do que o projeto continua em plena atividade, podendo surgir melhorias no código em termos de desempenho.

2.2.8 QEmu

Apesar de a filosofia ser basicamente a mesma do Bochs, o QEmu consegue um desempenho melhor através de um processo de compilação das instruções da CPU emulada, transformando-as num formato intermediário de interpretação rápida [25].

A compilação é lenta, mas o resultado é armazenado em *cache*, o que dilui o custo de compilação se o mesmo código é executado várias vezes [26], e o sistema emulado roda bastante rápido (depende muito do aplicativo).

Outra grande vantagem do QEmu é estar preparado para emular vários processadores e arquiteturas. Um certo número de sistemas-base é suportado, com graus variáveis de maturidade (Linux é bem suportado, Mac OS X está em versão *beta*).

O QEmu tem um modo rápido, que faz uso da MMU (*Memory Management Unit*) do processador nativo para fazer as tarefas de gerenciamento de memória da emulação. Isto exige a instalação de um módulo no *kernel* do sistema anfitrião, o que é geralmente indesejável.

Além do modo "rápido", o QEmu tem um modo *user-level*, apenas para Linux. Neste modo, a emulação é feita no contexto de um executável Linux. Apenas a CPU e a memória virtual são emuladas, não o *hardware*. Este modo permite rodar, digamos, um executável Linux/Intel em um Linux/PowerPC.

2.3 *Parallels Desktop*

O *Parallels Desktop* foi desenvolvido pela *Apple* para prover ambientes virtualizados tendo como hospedeiro o sistema operacional Mac OS X, com compatibilitade total na versão *Leopard*.

Isto proporciona execução de *Windows* e Linux, e utilizando a técnica de emulação, onde cada máquina virtual tem emulado o seu próprio *hardware* [27].

Como a emulação repassa um *hardware* genérico para todas as máquinas virtuais de um computador, o *Parallels Desktop* proporciona portar uma máquina virtual de um computador para outro, sem necessidar de nenhuma modificação no sistema hóspede.

2.4 Comparação Entre Virtualizadores

Na Tabela 1 vemos uma comparação entre os *softwares* de virtualização citados, bem como ver em que tipo de processador cada um suporta, qual o tipo do sistema operacional serve como anfitrião e quais seus sistemas convidados, qual técnica cada um utiliza, seu tipo de licença e suas características.

Podemos notar a grande variedade de virtualizadores que estão disponíveis no mercado, com suporte aos mais diversos sistemas operacionais e plataformas, a maioria com licenças livres para utilização, alguns com ênfase acadêmica e em constante desenvolvimento.

2.5 Outros Aspectos

Máquinas virtuais provém compartilhamento de recursos entre aplicações e usuários do sistema dando a ilusão da utilização total dos recursos disponíveis.

Alocação de recursos físicos é definida na inicialização da máquina virtual, porém algumas podem ser alteradas em tempo de execução.

A maioria dos monitores de máquinas virtuais permitem especificar uma quantidade total de processadores que serão necessários alocando esse recurso para essa máquina virtual específica. Porém, cada máquina virtual, apesar de poder especificar que necessita de um certo número de processadores, irá compartilhar esses processadores com as outras máquinas virtuais.

Tratando-se de memória, o compartilhamento total da memória alocada para a máquina virtual é feita em espaços de grande granularidade, por exemplo 1 *Megabyte*, pois pequenos espaços de memória aumentam o ônus de busca feita pelo monitor.

Ainda, a maioria dos monitores de máquinas virtuais tem o seu próprio subsistema de entrada/saída, pois os dispositivos de entrada/saída pode ter múltiplas portas que podem ser com-

Tabela 1 – Comparativo entre virtualizadores.

Nome	Processador	SO Anfitrião	SO Convidado	Técnica	Licença
QEMU	Intel x86, AMD64, IA-64, PowerPC, Alpha, SPARC 32/64, ARM, S/390, M68k	Linux, Windows, MAC OS, FreeBSD, BeOS	Windows, DOS, Linux, *BSD	emulação	GPL/LGPL
UML	Intel x86	Linux	Linux	paravirtualização	GPL2
VMWare	Intel x86, AMD64	Linux, Windows	Windows, DOS, Linux, Unix BSD, Netware, Solaris	emulação	Proprietário
Xen	Intel x86, AMD64, IA-64	Linux, NetBSD	Linux, Unix BSD, Windows XP	paravirtualização	GPL
FAUmachine	Intel x86	Linux	Linux	virtualização	GPL2
OpenVZ	Intel x86, AMD 64, IA-64, PowerPC, UltraSPARC	Linux	Linux	virtualização	GPL2
VirtualPC		Windows	DOS, Windows, OS/2	virtualização	Proprietário
Plex86	Intel x86	Linux	Linux	emulação	LGPL
Bochs	Intel x86, IA-32	Linux	Linux, DOS, Windows, *BSD	emulação	LGPL
Parallels	Intel x86	Mac OS X	Linux, Windows, *BSD, Solaris, OS/2	emulação	Proprietário

partilhadas entre várias máquinas virtuais, e geralmente são alocadas quando a máquina virtual é inicializada.

Várias limitações são impostas por arquiteturas multiprocessadas aos projetos de máquinas virtuais. Uma dessas limitações existem nas diferenças no acesso a memória em máquinas

NUMA (*Non-Uniform Memory Access*) , onde o desenvolvedor precisa se preocupar com implementações no subsistema de memória, problemas de sincronização em estruturas de dados globais, e mesmo falhas de *hardware* que não ficam limitadas a apenas uma parte do sistema.

Porém, a utilização de máquinas virtuais em ambientes multiprocessados está ganhando força a medida que esse tipo de arquitetura está sendo bastante utilizada por corporações que necessitam de grande poder computacional. Um exemplo é a *Cellular Disco* [28] que transforma uma máquina multiprocessada em um *cluster* virtual, gerenciando recursos e provendo contenção de falhas de *hardware*.

2.6 Considerações Finais

Com a grande diversidade de aplicações nas corporações, as necessidades dos usuários são imprevisíveis, altamente mutáveis, porém o parque de máquinas geralmente não acompanha esse raciocício. Com isso tornam-se enormes os custos de se manter uma administração adequada.

Na visão da virtualização, tais questões são abstraídas por um modelo mais flexiveis. A grande vantagem de se dispor de uma infra-estrutura como a virtualização é que a soma de suas capacidades de disco, processadores e memória podem ser gerenciados e distribuídos transparentemente, tornando o ambiente mutável e dinâmico.

No próximo capítulo, veremos o monitor de máquinas virtuais Xen, foco principal deste trabalho.

3 Xen

O Xen foi desenvolvido pelo *Systems Research Group* da Universidade de *Cambridge*, e é parte de um projeto maior chamado XenoServers, que provê um ambiente de computação global distribuída. O Xen permite compartilhar uma simples máquina para vários clientes rodando sistemas operacionais e seus respectivos programas [9].

Com a renovação da utilização de máquinas virtuais através da consolidação de servidores, o Xen tem se tornado acessível a um número cada vez maior de usuários, proporcionando ganhos de desempenho, o que o torna uma alternativa interessante para vários sistemas de computação, através de suas vantagens como custo e portabilidade.

O Xen utiliza o conceito de paravirtualização (ver seção 2.1.3), onde o sistema operacional rodando sobre uma máquina virtual tem a ilusão de estar sendo executado diretamente sobre o *hardware*. O Xen se encarrega de organizar as requisições feitas pelas máquinas virtuais e repassá-las ao sistema principal. Ele se limita a repassar as instruções, sem interpretá-las como faria um emulador, o que causa uma diminuição de desempenho muito pequena.

3.1 Estrutura Interna do Xen

A maioria dos sistemas operacionais em arquitetura i386, por padrão, funcionam em nível 0 (*ring level 0*) [17], onde se pode executar qualquer instrução privilegiada e qualquer acesso de entrada/saída. O Xen modifica o *kernel* do nível 0 para nível 1, passando o próprio Xen a rodar em nível 0.

As aplicações não percebem essa mudança por que executam em nível 3, que é o menos privilegiado deles. No caso do Xen, o sistema que vai ser executado na máquina virtual precisa ser modificado, ou seja, é necessário instalar um *patch* no *kernel* para que o Xen funcione.

O Xen tem a estrutura similar a um sistema operacional *microkernel* [9], onde se pode ter múltiplos sistemas operacionais, gerenciando todo o acesso à memória e dispositivos [29]. O Xen provê uma camada, chamada *hypervisor*, ao *hardware* onde são portados seus sistemas operacionais, e em retorno se tem a habilidade para rodar múltiplas instâncias de sistemas operacionais (ver Figura 4).

Na Figura 4, observamos que o Xen se divide em quatro níves. O primeiro nível (nível mais superior) corresponde às aplicações que rodam dentro das máquinas virtuais juntamente com os *softwares* adicionais que monitoram o funcionamento do Xen. No segundo nível temos o

sistema operacional hospedeiro com seu *kernel* modificado para proporcionar virtualização e os sistemas operacionais convidados (máquinas virtuais). No terceiro nível temos o *hypervisor* que é uma camada que controla as chamadas de sistema entre as máquinas virtuais e o *hardware*, e por fim, temos o *hardware*.

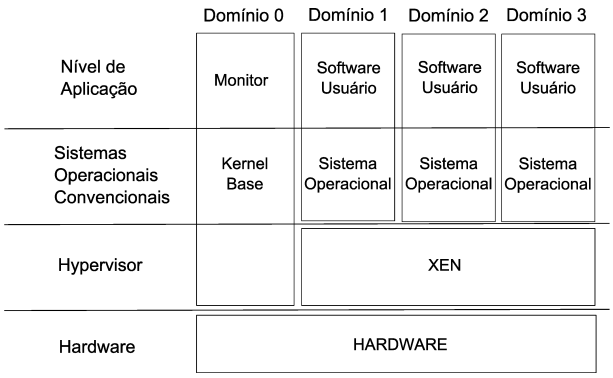

Figura 4 – Estrutura do Xen.

Uma virtualização completa acarretaria perda de desempenho, portanto o Xen faz uma paravirtualização, abstraíndo o *hardware* e perdendo pouco desempenho se comparado à máquina real [9] . No Xen, somente o *hypervisor* tem privilégios totais aos recursos como processador e à memória física, ficando a seu cargo a alocação de memória dos domínios virtuais (máquinas virtuais, também chamados Domínio 1, 2, 3, 4 - ver Figura 4), e todo estado privilegiado e controlado pelo Xen, assegurando assim, um uso seguro de paginação e segmentação.

Como o sistema operacional virtualizado executa em um nível menos privilegiado, instruções privilegiadas são validadas pelo Xen. O Xen reserva para seu uso uma parcela de memória física, e também reserva uma parcela fixa pequena para cada espaço de endereço virtual. Quando a memória física é alocada ou liberada, não há nenhuma garantia que um domínio receberá um aumento contíguo da memória física.

O Xen utiliza acesso de somente-leitura às tabelas de páginas da memória e fica a cargo do sistema operacional base explicitamente requerer qualquer modificação. O Xen valida toda requisição e somente aplica modificações seguras ao ambiente. Isto se faz necessário para prevenir que domínios possam adicionar mapeamentos arbitrários em sua tabela de páginas. Na validação, o Xen associa um tipo e um contador de referência com a página da memória. Cada página é mutuamente exclusiva em algum ponto do tempo.

O Xen também provê um modo alternativo de operação em que o sistema operacional convidado tem a ilusão que suas tabelas de páginas são diretamente graváveis, porém não é assim que ocorre pois o *hypervisor* do Xen valida todas as modificações.

3.1.1 Gerência do Processador no Xen

O Xen tem suporte a escalonadores tanto estáticos quanto dinâmicos. Para máquinas multiprocessadas a utilização de escalonadores dinâmicos é mais apropriada do que os escalonadores estáticos uma vez que estas máquinas podem sofrer reconfiguração em tempo de execução. Por exemplo, mais processadores poderiam ser anexados à máquina multiprocessada, como é o caso em diversas arquiteturas NUMA.

O primeiro escalonador, chamado *Borrowed Virtual Time* [30] provê um compartilhamento proporcional entre as CPUs, porém existe neste escalonador uma grande dúvida no que diz respeito à precisão na predição dos recursos necessários para o processamento, o que se mostra significativo em aplicações que necessitam de processamento uniforme e também o serviço de interrupção podem efetivamente roubar ciclos, fazendo o número de ciclos do processador incerto em algum período de tempo.

O segundo escalonador, chamado *Scan Earliest Deadline First* [31] usa uma fila de prioridades onde o processo de menor *deadline* é escolhido para a execução, porém além de problemas de afinidade com os processadores, neste escalonador não existe como controlar quanta CPU cada máquina virtual vai consumir do domínio 0, embora tenham sidos configurados pesos de processamento para cada domínio.

O último escalonador e atualmente o padrão no Xen é chamado *SMP Credit* [32], tido como ideal para máquinas SMP (*Symmetric Multiprocessor*) pelos desenvolvedores do Xen, pois tem a capacidade de quando uma CPU real estiver parada, buscar CPUs virtuais (VCPUs ou Virtual CPUs) de outras CPUs reais para execução. Na Figura 5, é mostrado como a CPU0 é dividida em duas CPUs virtuais e como uma CPU virtual da CPU0 é realocada para a CPU1, e depois como a CPU virtual que estava originalmente na CPU1 passa para a CPU0, e assim por diante.

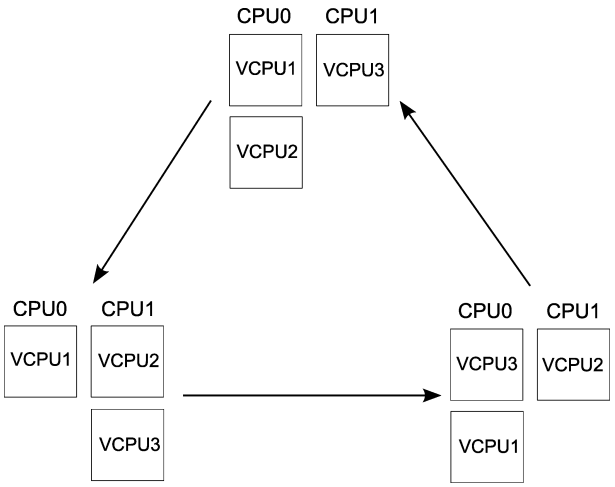

Figura 5 – VCPUs Dinâmicas.

Embora indicado para máquinas SMP, o *Credit* só consegue fazer um bom balanceamento de recursos se existir algum processador livre, o que é difícil de ocorrer em um ambiente de

produção utilizando máquinas virtuais. Ainda, quando utiliza processos que utilizam intensivamente entrada/saída não existe como fixar limites aos domínios sobre a utilização das CPUs, pois a escrita/leitura é controlada pelo domínio 0 e não pelas máquinas virtuais.

Na gerência de processos o escalonador *Credit* também apresenta algumas limitações, por exemplo se tivermos dois processos (A1 e A2) onde A1 está utilizando o processador e A2 está bloqueado, o processo A1 está consumindo créditos e proporcionalmente perdendo prioridade. Quando o processo A2 entrar em estado de execução, ele receberá uma fatia de processamento maior que o processo A1, pois terá menos créditos e maior prioridade, não garantindo um compartilhamento justo que é o princípio do escalonador *Credit*, pois penaliza o processo A1 [10].

3.1.2 Gerência de Memória no Xen

O Xen reserva uma quantidade fixa de memória durante o início do sistema. Essa região de memória se localiza no início da memória física e é mapeada no topo de cada região de memória virtual. Apesar de essa memória estar sempre mapeada, ela não é sempre acessível aos domínios virtuais [33].

A memória física que não é usada pelo monitor virtual é dividida em páginas é se torna disponível para alocação pelos domínios virtuais. O monitor virtual rastreia quais páginas estão livres e quais páginas estão sendo usadas por domínios virtuais.

Quando um novo domínio é criado, o monitor virtual aloca memória para o domínio a partir da lista de páginas livres. O total de memória requerida para o novo domínio virtual é passada para o monitor virtual a partir do *domain builder*.

Um domínio virtual nunca pode alocar mais memória do que a memória que foi inicialmente alocada para ele, porém ele pode retornar ao monitor virtual páginas que não estão sendo utilizadas. Essas páginas podem ser requisitadas novamente pelo domino virtual, porém somente se o sistema não estiver sobre pressão de memória ou se a página em questão não estiver alocada.

Também é possível que vários domínios compartilhem páginas de memória, porém somente para leitura.

3.2 Limitações no Escalonamento do Xen

Desde o seu desenvolvimento, o Xen teve como objetivo principal o desempenho e a escalabilidade. O código atual de Xen suporta sistemas convidados em máquinas monoprocessadas e SMP [34].

Apesar do Xen estar sendo bastante usado para consolidação de servidores, ainda apresenta alguns problemas a serem resolvidos. Focando no escalonador *Credit*, podemos apresentar

algumas das pricipais limitações:

- Não existe previsibilidade sobre o comportamento dos processos para uma melhor alocação da CPU, ou seja, o escalonador não diferencia entre uma sobrecarga momentânea ou duradoura de processamento, possiblitando um ônus de realocação sobre realocação;

- A fatia de tempo em que os processos podem estar em estado de execução é de 30 milissegundos fixos, o que não é um parâmetro otimizado quando o Xen está com o escalonador dinamicamente configurado em máquinas SMP;

- A contabilização no escalonador é feita de uma maneira simplista de 1 *tick* a cada 10 milissegundos, o que ocasiona sempre uma perda de 10 milissegundos para a máquina virtual que está em execução, portanto é necessário um esquema mais exato e mais eficiente da contabilidade;

- Quando está configurado o escalonamento dinâmico através da utilização de VCPUs, não se pode garantir níves de serviços para máquinas virtuais, pois o escalonador não permite determinar intervalos de processamento ideais para aplicações que estejam rodando nas máquinas virtuais, pois se baseia apenas na necessidade de processador.

3.3 Considerações Finais

A consolidação do servidor é uma estratégia importante para a redução de custos, e as atuais soluções de *software* para virtualização tornam fácil a execução de aplicações múltiplas de forma segura.

Obviamente, o planejamento é importante em um projeto de consolidação bem sucedido. Para a maioria das organizações, a virtualização envolve novos produtos e novas tecnologias, assim como novos procedimentos de TI e novos modelos de uso. As políticas para tomada de decisão normalmente também precisam mudar, já que os servidores físicos individuais poderão ser compartilhados entre múltiplas unidades de negócios.

Virtualização é um tema de grande pesquisa nos dias de hoje, haja visto suporte a novas tecnologias de processadores que visam a suportar essa tecnologia. A virtualização tende a ajustar uma arquitetura específica as necessidades computacionais de sistemas de *software*.

Xen é uma tecnologia relativamente nova, mas o suporte da indústria junto com as soluções prontas para serem desenvolvidas estão começando a surgir, e avanços rápidos podem ser esperados.

No Xen, todo o gerenciamento do processador, memória e dispositivos é realizada pelo *hypervisor*, que oferece uma interface para cada máquina virtual, permitindo uma execução independente. Esse gerenciamento é feito através de *system calls* realizadas pelas máquinas

virtuais ao domínio principal (sistema operacional que da suporte à virtualização ou Domínio 0), e o *hypervisor* vai controlar o acessos dessas chamadas ao *hardware*.

O Xen permite uma separação lógica entre o *hardware* e as máquinas virtuais ou sistemas operacionais convidados, permitindo uma maior flexibilidade, e aproveitando melhor o *hardware*, controlando ainda, o acesso seguro ao *hardware* pelas máquinas virtuais.

A seguir apresentaremos o subsistema de realocação de recursos desenvolvido para possibilitar direcionar a quantidade de recursos ociosos para máquinas virtuais que necessitem desses recursos.

4 Subsistema de Alocação Dinâmica de Recursos

O subsistema proposto foi desenvolvido para auxiliar o escalonador *Credit* quando configurado no modo padrão, onde o escalonador é configurado com pesos fixos de processador e memória para cada máquina virtual e esses valores não se alteram. O escalonador *Credit* pode ser configurado de duas maneiras, com limites máximos de processamento (padrão) e sem limites máximos de processamento. Quando configurado sem limites máximos de processamento, o escalonador se ajusta dinamicamente às cargas das máquinas virtuais e a sobra de processamento é direcionada para o domínio principal (Domínio 0 - ver Seção 3.1).

Entretanto a falta de limite máximo de processamento para as máquinas virtuais pode causar alguns problemas em ambientes onde existe a definição de algum acordo de nível de serviço (*Service Level Agreement* - SLA) . Esta situação acontece pois uma determinada máquina virtual pode receber no máximo a carga que está sobrando no sistema ou então uma fatia igual as demais máquinas virtuais que estão executando.

Caso seja necessário estipular uma quantia maior de recursos, ou um valor máximo para uma determinada máquina virtual, então deve-se estabelecer estes limites. No entanto, quando utilizamos o escalonador com limites de processamento individual para cada máquina virtual, os valores máximos definidos permanecem fixos.

Isto pode não ser apropriado em diversas situações. Por exemplo, dependendo das aplicações em uma máquina virtual, ela poderá necessitar de maior processamento que o seu limite configurado. Caso exista disponibilidade, pois as outras máquinas virtuais podem não estar utilizando os recursos que lhes foram alocados, a máquina virtual que necessita mais recursos não os receberá.

O subsistema proposto faz a realocação dos recursos para as máquinas virtuais neste caso, onde os limites máximos de processamento precisam ser ultrapassados. O subsistema irá aumentar os recursos somente se outras máquinas virtuais não estiverem utilizando parte dos recursos que elas possuem disponíveis, proporcionando um melhor balanceamento do ambiente.

Basicamente o subsistema pretende transformar a situação exemplo mostrada na Figura 6 para a situação mostrada na Figura 7. Na Figura 6, as máquinas virtuais A até D possuem disponíveis os mesmos limites máximos de recursos - mostrados pela linha pontilhada.

Portanto, o subsistema proposto é dividido em duas partes: o monitor e o atuador. O módulo monitor que verifica de tempos em tempos as necessidades de processamento das máquinas virtuais e o atuador que modifica as configurações de pesos de processador e memória para melhorar o desempenho do ambiente.

Como pode ser visto na figura, a máquina virtual C necessita mais recursos do que possui

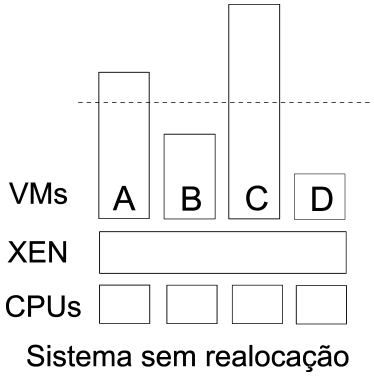

Figura 6 – Xen sem utilização do subsistema.

disponível, enquanto a máquina virtual B não está utillizando todos os recursos que lhe foram disponibilizados. O subsistema irá redistribuir a alocação de recursos conforme mostrado na Figura 7.

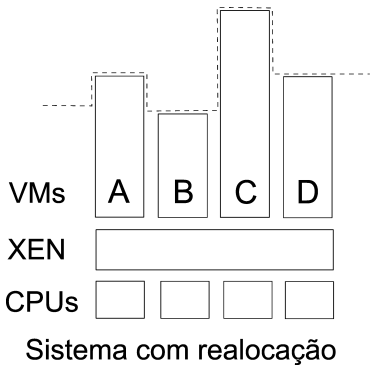

Figura 7 – Xen com utilização do subsistema.

Como verificado, a idéia é que todas as máquinas tenham os recursos que necessitem, e não existam recursos sub-utilizados. Claro que nem sempre esta situação é possível. Poderia haver a situação onde todas as máquinas virtuais estivessem utilizando todos os recursos disponibilizados e portanto a máquina que necessitasse mais recursos teria que ficar esperando tendo um menor desempenho do que o esperado acarretando menor desempenho global.

O subsistema de realocação de recursos é composto por duas partes: monitoramento e realocação.

4.1 Monitores em Sistemas Operacionais

Hoje em dia existem várias aplicações de monitoramento de sistemas operacionais, pois cada vez mais se faz necessário ter o controle sobre o sistema, e temos como exemplo as seguintes aplicações: *HP Openview Operations* [35], *Tivoli Monitoring* [36], *Microsoft Operati-*

ons Manager (MOM) [37] e *Nagios* [38].

Os *softwares* de monitoramento de sistemas operacionais são componentes lógicos bastante utilizados em sistemas de tempo real com a característica de obter informações do sistema, onde as aplicações mais comuns são utilizadas para gerenciar usuários, processos, arquivos, diretórios e dispositivos, tendo como principal função auxiliar ao administrador do sistema em decisões gerenciais e técnicas [39].

Alguns tipos de monitores de sistemas operacionais, além de verificarem eventos específicos do sistema operacional, tomam decisões sobre a melhor configuração para determinado evento, possibilitando uma melhor funcionamento do sistema baseado em métricas pré-definidas.

4.1.1 Taxonomia de Monitoramento de Sistemas

Os sistemas monitores existentes podem ser classificados em primeiro plano, entre interativos ou automáticos [40]. Sistemas de monitoramento interativos são aqueles que necessitam da intervenção humana para sua tomada de decisão ou sugestões de como atuar sobre o sistema, por outro lado, sistemas de monitoramento automáticos tem autonomia para chegar em uma decisão de como atuar sobre o sistema.

Essa atuação leva a outra definição, que classifica os monitores de sistema entre sistemas de observação ou sistemas de manipulação. Sistemas de monitoramento de observação são aqueles que verificam o sistema buscando por determinados eventos, e avisam quando esses eventos ocorrem. Sistemas de monitoramento de manipulação, além de verificarem o sistema esperando determinados eventos, quando esses eventos ocorrem, atuam sobre o sistema.

Quanto a observação, ainda podemos distinguir sistemas de monitoramento entre *on-line* e *off-line*. A grande diferença entre sistemas *on-line* e *off-line* é que em sistemas *on-line*, o sistema de monitoramento está rodando junto à aplicação monitorada, enquanto em sistemas *off-line* rodam após a aplicação ter terminado de rodar.

4.1.2 Metodologia de Monitoramento de Sistemas

A utilização de *threads* é bastante difundida no monitoramento de sistemas paralelos, pois permite uma maior facilidade de comunicação e sincronização de dados pois coexistem no mesmo espaço de endereçamento, ao contrário de subprocessos criados através do *fork* [41].

Apesar de ser uma técnica bastante invasiva, podemos diminuir seu ônus de processamento utilizando técnicas que permitem deixar a *thread* bloqueada através de semáforos até que o evento que satisfaça a sua condição de execução aconteça.

Outra medotodologia de monitoramento, que é utilizada pelas aplicações nativas de monitoramento do Xen (por exemplo: Xentop, Xenmon), são programas com laços que utilizam

a função *sleep* para monitorar de tempos em tempos o sistema, causando menos impacto na utilização do processador.

Enquanto o programa monitor está em *sleep*, fica suspenso esperando por um sinal de *SIGALARM* (alarme de relógio - *time out*) do *kernel* para recomeçar a sua execução (no padrão *POSIX*). O único problema desse tipo de implementação é quando termina o período de segundos configurados no *sleep*, pois não há garantia que o programa retome imediatamente a execução, porque isso depende da política de escalonamento da fila de processos em estado de pronto, o que pode acarretar atraso no monitoramento.

4.2 Alocação de Recursos

O Xen possibilita configurar a alocação estática do escalonador através da ferramenta xm. O xm é a ferramenta mais importante de gerenciamento do Xen através do console e provê um grande número de parâmetros que possibilitam esse gerenciamento. A sua sintaxe tem o seguinte formato:

```
# xm command [switches] [arguments] [variables]
```

Esse comando proporciona a criação de domínios (máquinas virtuais), listagem de domínios ativos, destruição de domínios, migração de domínios bem como a obtenção de informações sobre o respectivo domínio.

Em caso de arquiteturas SMP (*Symmetric Multiprocessor*) os processos são balanceados simetricamente entre os processadores obedecendo ao escalonador em uso, por padrão não tendo preferência entre máquinas virtuais e processadores, podendo estar, ora em um, ora em outro processador.

Porém, pode ser configurado em tempo de execução, de modo que se direcione os processos advindos de uma máquina virtual para apenas um processador determinado, utilizando o comando xm, e parâmetros como segue abaixo:

```
# xm vcpu_pin dom_id vcpu cpu
```

Onde:

- *vcpu_pin*: comando para restringir um domínio virtual para uma determinada CPU;
- *dom_id*: identificador do domínio;
- *vcpu*: identificador da CPU virtual;
- *cpu*: identificador da CPU física.

O comando xm envia um valor *TRUE* para *opt_dom0_vcpus_pin* no arquivo *schedule.c*, que fixa um domínio virtual à uma CPU física.

Alguns parâmetros do comando xm dizem respeito também ao escalonador *credit*, o que vem a permitir em tempo de execução a mudança dos valores de peso e limite.

Vemos abaixo como mudar esses valores, respectivamente:

```
# xm sched_credit -d <dominio> -w <peso>

# xm sched_credit -d <dominio> -c <limite>
```

O subsistema desenvolvido se utiliza da ferramenta xm para realocar dinamicamente recursos como quantidade de processador e memória para cada máquina virtual, quando o monitor avaliar que uma delas necessita de recursos.

A alocação dinâmica dos recursos computacionais e a realocação automática sem intervenção humana possibilita conforme a demanda, reconfigurar o sistema em intervalos de tempo, mudando suas características de acordo com a necessidade de cada máquina virtual.

As decisões são tomadas em resposta às condições de carga de trabalho, associados a limites de recursos, o que possibilita uma maior flexibilidade do ambiente em se auto-reconfigurar.

4.2.1 Trabalhos Relacionados

A realocação de recursos visa à otimização, permitindo maior flexibilidade na utilização da capacidade computacional disponível, promovendo equidade sem prejuízo para o funcionamento do sistema. O estudo sobre alocação dinâmica é apresentado na literatura sob muitos pontos de vista e com métodos de análise diferentes como: modelos analíticos e implementações que vão desde otimizações na área de microeletrônica [42], sistemas operacionais [43], *clusters* [44] e *grids* [45]. Baseado nestas áreas podemos citar:

- realocação de recursos é utilizada entre outros, para a redução do consumo de energia em memórias cache na microeletrônica, possibilitando assim direcionar mais energia ao processador, colocar mais memória na mesma área ou até mesmo diminuir a temperatura do processador;

- garantia de níveis de serviço para sistemas operacionais através da realocação de recursos visa garantir tempo de processamento de determinada tarefa [46];

- reconfiguração dinâmica do tráfego da rede em *clusters* possibilita balanceamento de carga entre os nodos processadores e em *grids* computacionais a realocação de nodos provê maior agilidade não só no que tange redundância, mas também na escalabilidade do *grid*, incrementando sua capacidade de processamento;

- redes de sensores [47] atraem a atenção devido a sua facilidade de distribuição, possibilitando uma redundância entre os sensores e aumentando a área de cobertura. A realocação está justamente em uma distribuição dos usuários ou objetos conectados a essas redes, o que possibilita re-balancear a rede de tempos em tempos;

- ainda, existem modelos que simulam realocação de recursos computacionais reproduzindo o comportamento de sistemas complexos, possibilitando uma maior gama de configurações possíveis, o que torna fáceis ajustes ao modelo, proporcionando uma futura implementação mais precisa.

Todos estes trabalhos tentam definir parâmetros mais apropriados para o melhor funcionamento do sistema como um todo, através do uso da realocação de recursos.

4.3 Subsistema Proposto

O subsistema proposto e implementado nessa dissertação baseia-se nas configurações estáticas utilizadas pelo comando xm, para prover alocação de recursos *on-the-fly*, buscando por fatias de processamento ociosas em máquinas virtuais do ambiente, podendo assim direcionar esses recursos para outras máquinas virtuais que necessitam dos mesmos.

Para isso, é necessário saber qual configuração possibilita uma melhoria de desempenho para determinado ambiente, então foram executados *benchmarks* em diversas configurações, e sobre os resultados dessas execuções foram aplicados algoritmos de mineração de dados (*Data Mining*), mostrando relações entre as execuções e as configurações, podendo assim chegar a uma estrutura com as configurações possíveis e as suas configurações ótimas para cada configuração atual.

O subsistema de alocação de recursos é parte da arquitetura proposta na Figura 8. A idéia principal nesta arquitetura é a descoberta da melhor alocação de recursos através da geração de um *Data Warehouse* com dados de execução de diversos *benchmarks*.

Os dados são coletados através de centenas de execuções de *benchmarks* com diferentes configurações de parâmetros para máquinas virtuais. Sobre esse *Data Warehouse* são executados algoritmos de mineração de dados que produzem um modelo preditivo. Este modelo preditivo será utilizado pelo subsistema de realocação de recursos.

A partir disto, o subsistema de realocação de recursos (Figura 9) utilizará a nova configuração para as máquinas virtuais dependendo da configuração e das necessidades atuais das máquinas virtuais. Estas necessidades de recursos são determinadas pelo módulo de monitoramento que está analisando os recursos sendo utilizados pelas máquinas virtuais existentes. Este subsistema de monitoramente e realocação de reursos é o objetivo principal desta dissertação, e todo o processo de descoberta de conhecimento faz parte de outra pesquisa de mestrado (Figura 10) do mesmo grupo de pesquisa.

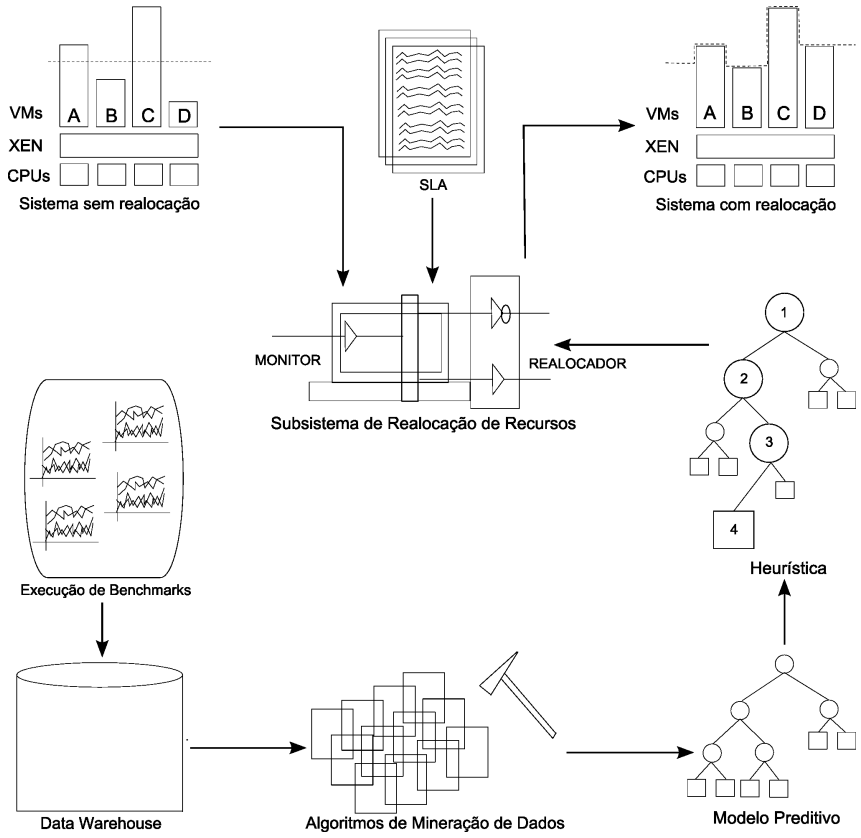

Figura 8 – Arquitetura Proposta.

Acompanhando o processo completo proposto na Figura 8 podemos partir de uma configuração onde as máquinas virtuais estão sem realocação de recursos, passando pela execução de *benchmarks*, a utilização de um *data warehouse* onde são tabuladas e carregadas as informações sobre os relatórios dos *benchmarks*.

A aplicação de algoritmos de mineração de dados sobre essa base de dados, a extração de um modelo preditivo gerado baseado nas métricas necessárias para realocação de processador e memória, criação de uma heurística que será importada pelo subsistema de realocação de recursos, possibilitando através de acordos de níveis de serviço, máquinas virtuais com realocação de recursos onde o ambiente estará com um desempenho global melhor.

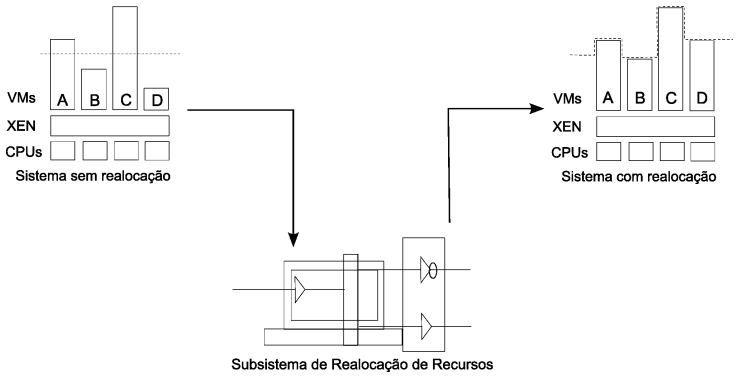

Figura 9 – Subsistema na Arquitetura Proposta.

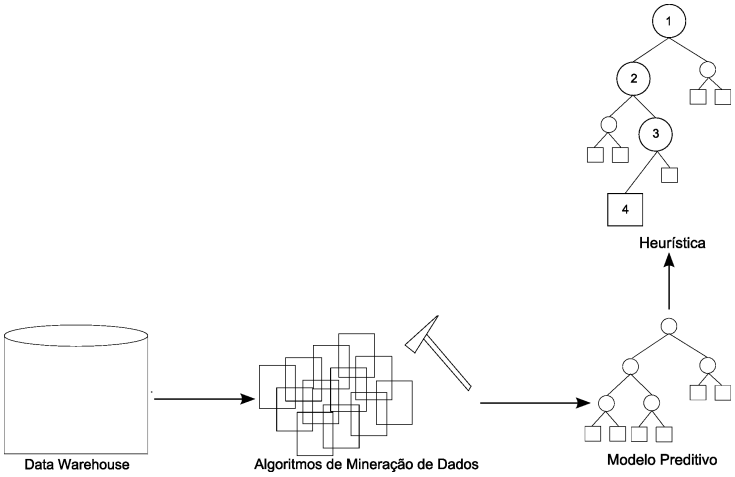

Figura 10 – Descoberta de Conhecimento na Arquitetura Proposta.

A seguir poderemos acompanhar todo o processo de descoberta de conhecimento utilizado como base para desenvolver o subsistema de alocação de recursos neste trabalho. Toda a técnica de descoberta de conhecimento é baseada em outa dissertação de mestrado do mesmo grupo de pesquisa

4.3.1 Execução de Benchmarks

Antes da execução dos *benchmarks* que vão gerar a base de dados de informação necessárias para a criação do modelo preditivo, deve-se planejar de que maneira esses dados serão mensurados, bem como quais as métricas mais importantes para esse fim.

Para tanto, a criação de um plano de execução de *benchmarks* se faz necessário, pois permite objetividade nas execuções, limitando dentre um grande número de possibilidades de execuções, somente às que interessam para suprir os objetivos da pesquisa.

Para cobrir a maioria das configurações com diversas fatias de processador e memória, todas as possiblidades de configuração dentro do escopo definido pelo plano de execução devem ser testadas, podendo-se assim gerar um grafo de execuções mostrando, dependendo da configuração atual, quais configurações podem melhorar o desempenho global do ambiente.

Para tanto, foram executadas várias possibilidades de configurações, alterando fatias de processador e memória, gerando um grande número de relatórios com os mais diversos desempenhos de máquinas virtuais para as mais diversas configurações propostas.

Essas execuções geraram centenas de relatórios, com os mais variados valores finais de desempenho para as métricas escolhidas, mostrando o comportamento de cada máquina virtual nos mais diversos cenários.

4.3.2 *Data Warehouse*

O grande acúmulo de dados gerados por execuções de *benchmarks* nos mostram desempenhos individuais, porém de modo *ad hoc* não conseguimos ver o conjunto de informações como um todo com suas correlações, nem nos mostrar suas tendências e padrões. Para que isso seja possível, é necessário que todas essas informações sejam armazenadas em uma aplicação de forma que fiquem acessíveis e de fácil manipulação, e essa aplicação se chama *Data Warehouse* [48].

Data Warehouse consiste em uma coleção de dados com informações organizadas por assunto, onde o propósito é promover um suporte a utilização de técnicas de seleção de dados pertinentes e automatizando a extração dessas informações de forma integra.

Os relatórios que foram gerados através da execução dos *benchmarks* foram formatados de maneira que pudessem ser importados pelo *Data Warehouse* criado, onde foram organizadas as informações, disponibilizando assim, um ambiente onde podem ser utilizados algoritmos de mineração de dados para encontrar padrões e relações entre as informações.

4.3.3 Mineração de Dados em *Benchmarks*

A mineração de dados (*data mining*) é o processo que consiste em explorar grandes quantidades de dados, se utilizando de algoritmos específicos, buscando por padrões, regras de associação ou seqüências temporais que possibilitem obter relacionamentos entre as variáveis, gerando por fim novos subconjuntos de dados afins [49].

Para a mineração de dados sobre os *benchmarks* utilizou-se o algoritmo de aprendizagem e classificação C4.5 [50], que gera estruturas de dados baseado em um conjunto de treino. A sua escolha se deve à vantagem de proporcionar na própria construção dessa estrutura, uma poda que elimina os ramos desnecessários à pesquisa, o que resulta em uma classificação com maior precisão nos dados, bem como rapidez.

Então, através desse algoritmo de mineração de dados, foi gerada uma estrutura de dados com as configurações possíveis das máquinas virtuais, e seus caminhos de melhoria de desempenho. Baseado nesses grupos de dados, o sistema poderá escolher fatores correlacionados com determinado objeto de saída como o primeiro nodo correspondente a configuração atual, e a partir dele, pode-se rapidamente ver qual a melhor opção de saída, em nosso caso, uma configuração mais otimizada.

4.3.4 Modelo Preditivo

A estrutura de dados desejada vem a ser um método de visualização que contém decisões a serem tomadas, buscando alcançar um objetivo. Então, consiste em um modelo preditivo onde através de observações de seus itens, podemos chegar a conclusões sobre todos os valores envolvidos.

O modelo preditivo gerado pelo algoritmo de mineração de dados é utilizado pelo subsistema de alocação dinâmica de recursos para ir de uma determinada configuração para outra que possibilite um melhor desempenho global, seguindo os ramos de configurações indicados como melhorias para aquela configuração.

Esse modelo preditivo é utilizado pelo subsistema de alocação de recursos proposto, para decidir qual máquina virtual necessita de recurso, bem como quais máquinas virtuais tem fatias de processamento ociosas que possam ser deslocadas para a primeira.

4.3.5 Interface entre Sistemas

Para comunicação com o subsistema, o modelo preditivo é exportado para XML (*EXtensible Markup Language*), e carregado em memória pelo subsistema de alocação de recursos, onde foram utilizadas técnicas de otimização para sua utilização.

O XML foi escolhido por ser um dentre vários formatos padrões para dados estruturados, e consiste em uma especificação técnica do W3C (*World Wide Web Consortium*) que é o órgão responsável pela área gráfica da *internet* [51].

O XML traz algumas vantagens pois permite buscas eficientes à informações através do grande número de atributos que podem referenciar um determinado objeto, os dados em XML são de fácil interoperorabilidade com qualquer tipo de sistema, pode integrar informações de vários tipos de fontes de dados diferentes em um único arquivo e escalabilidade pois separa dados para visualização, permitindo ao usuário uma visão fragmentada de um conjunto total independente da quantidade de informações.

Hoje em dia, é uma das maneiras mais flexíveis de troca de dados, através de um padrão aberto e independente de dispositivo. Para o subsistema de alocação de recursos dessa dissertação, é de grande valia por promover uma fácil manutenção na tomada de decisões do sistema, pois bastaria subsituir a estrutura de dados em XML para uma reconfiguração de parâmetros do subsistema.

4.3.6 Tempo de Realocação

Um ponto importante do subsistema proposto é calcular quando é válido ao subsistema realocar os recursos das máquinas virtuais evitando *trashing*, ou seja, evitar que durante uma realocação de recursos onde aconteça uma mudança dinâmica nas cargas das máquinas virtuais, não ocorra a necessidade de uma nova realocação logo após, novamente, ou então quando o gasto para fazer a realocação seja maior que o benefício.

Para tanto, o subsistema calcula o ônus da realocação para as máquinas virtuais (O), se utiliza também de uma variável λ (parêmetro de ajuste) que serve para ajustar o tempo, e a soma desses dois valores não pode ser maior que o tempo de realocação (TR - Intervalo entre realocações) para que o subsistema avalie como viável a reconfiguração do ambiente, como podemos ver abaixo:

$$TR > (O + \lambda)$$

Se o tempo de realocação é maior que o ônus somado com a variável de ajuste de tempo, então deve ser feita a realocação minimizando o problema do *trashing*, porém se o tempo de realocação é menor, o subsistema não realoca recurso e volta a monitorar, pois o ônus que levaria para realocar os recursos seria maior do que o ganho de desempenho que essa realocação resultaria.

Durante os *benchmarks* realizados podemos notar que o tempo de realocação em média, para cada máquina virtual era de 0.106 segundos, o que resulta em nosso ambiente com 4 máquinas virtuais um ônus de 0.424 segundos para uma realocação total do ambiente com os novos pesos para processador e memória definidos.

4.3.7 *Service Level Agreement* - SLA

Um acordo de nível de serviço é um contrato entre o fornecedor de serviços na área de tecnologia da informação e o cliente, que especifica mensuravelmente, quais serviços e qual a qualidade do serviço que o seu fornecedor vai prestar. Esses níveis de serviço servem para monitorar o desempenho do fornecedor, de maneira que sempre exista um serviço de qualidade, baseado nos critérios acordados [52].

A solução proposta pelo subsistema de realocação de recursos proporciona uma base para se trabalhar com garantias de níveis de serviço (*Service Level Agreement* - SLA), através de modificações no XML, pois hoje em dia, através da consolidação de servidores, o compartilhamento de uma infra-estrutura se torna de uso comum, porém as aplicações devem ser tratadas individualmente, conforme as suas necessidades de carga de processamento.

Então, para aplicações que necessitam limites ou garantias de fatias de processamento podem ser facilmente configurados com a manipulação do XML do subsistema, o que vem a garantir que máquinas virtuais com aplicações que possuam intervalos fixos de processamento, e mesmo que esses intervalos estejam ociosos não sejam direcionados para outras máquinas virtuais, garantindo o SLA configurado.

O que vem a garantir que uma máquina virtual que tem um SLA que lhe garante 70% da utilização do processador, e mesmo que a aplicação nessa máquina virtual não esteja utilizando todo esse limite de processamento, não é permitido ao escalonador redirecionar uma parte ociosa dessa fatia para outra máquina virtual que esteja necessitando de processamento.

4.3.8 Monitoramento

Após todo o processo de coleta de conhecimento que possibilita a realocação dos recursos de forma otimizada para as cargas determinadas nas máquinas virtuais, veremos agora como o subsistema carrega essas informações na inicialização e como o subsistema atua sobre as máquinas virtuais modificando-as.

Como mostrado anteriormente, subsistema proposto é dividido em duas partes: o monitor e o atuador. A parte do subsistema que faz a monitoria das máquinas virtuais utiliza a biblioteca de programação *libxenstat* [33], que é padrão em aplicações do monitor de máquinas virtuais Xen para extração de informações sobre as máquinas virtuais.

Através de *libxenstat* podemos extrair informações sobre vários dispositivos como processadores, memória, redes, etc. Utilizamos a *libxenstat* para monitorar a carga de processamento das máquinas virtuais, e quando alguma máquina virtual passa de um certo percentual de processamento, o subsistema busca fatias de processamento que estejam ociosas em outras máquinas virtuais, e baseado na estrutura de dados gerada pela mineração de dados, muda a configuração atual para uma configuração otimizada possibilitando uma melhoria global do ambiente.

Na inicialização do subsistema, a estrutura de dados é carregada, e para tanto, é utilizada outra biblioteca de programação chamada *libxml2*, que consiste em um *parser* XML para linguagem C [53].

O monitoramento é baseado no modelo empregado por todas as outras ferramentas de monitoramento nativas do Xen (Xenmon, Xentop), que consistem em laços infinitos que aguardam um determinado tempo, e executam o seu monitoramento, voltando ao estado de espera durante o tempo determinado, ou seja, periódico.

4.3.9 Realocação

O módulo responsável pela realocação de recursos percorre a árvore de decisão para encontrar a melhor situação baseada nos dados gerados pelo conjunto de *benchmarks* e determinados pelo algoritmo de mineração conforme explicado na Seção 4.3.3.

Na Figura 11 acompanhamos o processo de realocação, que depende de o módulo de monitoramento achar ou não a configuração atual do ambiente em um dos ramos da árvore de decisão, e caso encontre, reconfigure todo o ambiente para uma configuração em que o ambiente tenha um melhor desempenho.

```
ENQUANTO SUBSISTEMA ESTIVER RODANDO
    PARA CADA VMi: VERIFICAR CARGA DE Pi E QUANTIA DE Mi
    SE Pi OU Mi > LIMITE CONFIGURADO
        BUSCAR CARGA DE Pi EM VMi SUBUTILIZADA
        BUSCAR QUANTIA DE Mi EM VMi SUBUTILIZADA
        DIRECIONAR CARGA DE Pi PARA VMi LIMITADA
        DIRECIONAR QUANTIA DE Mi PARA VMi LIMITADA
```

Figura 11 – Algoritmo de monitoramento/realocação.

Quando módulo de monitoração do subsistema encontra uma máquina virtual (VMi) que necessita de mais recursos (processador Pi e memória Mi), busca nas outras máquinas virtuais por recursos que não estão sendo utilizados, e que pode ser transferido para essa máquina virtual. Esta realocação já deve estar prevista na árvore de decisão e quando sistema de monitoramento encontra o ambiente em um estado existente na árvore de decisão, automaticamente reconfigura o ambiente para um estado mais otimizado.

Assim, após a análise da situação atual das máquinas virtuais e da possível nova configuração para as máquinas virtuais, a partir da árvore de decisão, é possível ajustar parâmetros de escalonamento em tempo de execução.

4.4 Considerações Finais

O Xen está sendo cada vez mais utilizado pelo setor industrial, onde novas aplicações estão surgindo e as máquinas virtuais estão cada vez mais se adaptando para garantir serviço de qualidade.

Esta dissertação mostra uma forma de otimizar a realocação de recursos para o Xen, provendo um balanceamento de processador e memória, ajustando parâmetros estáticos do escalonador, gerando uma realocação dinâmica dos recursos.

Exibimos técnicas de mineração de dados para determinar uma melhor configuração automática de parâmetros. Ainda que essa estratégia seja utilizada em outros contextos, acreditamos que é uma boa maneira de análise dos resultados produzidos pelos *benchmarks*. Esta parte da

pesquisa faz parte de outra dissertação de mestrado, e foi apresentada nessa dissertação para possibilitar um melhor entendimento da inserção do subsistema no processo total.

A estrutura em XML gerada facilita a realocação dinâmica dos recursos do sistema, pois é um meio eficiente de construir classificadores, filtrando a melhor escolha para a realocação.

O trabalho realizado busca garantir acordos de níveis de serviço provido por uma máquina virtual para aplicações. Para que a máquina virtual consiga atender aos acordos de nível de serviço é fundamental que o sistema virtualizador também possua acordos de nível de serviço com as máquinas virtuais.

O trabalho apresentado gera a infra-estrutura para a geração de acordos de nível de serviço entre o Xen e as máquinas virtuais, onde apenas com a modificação no arquivo XML, pode-se gerar diversos cenários possibilidando diversas configurações para o ambiente de máquinas virtuais.

A sequir, mostraremos as técnicas e ferramentas utilizadas para avaliarmos os ambientes com e sem o subsistema de realocação desenvolvido.

5 Metodologia de Validação e Resultados

A validação do subsistema desta dissertação foi feita através da utilização de *benchmarks*, onde foram seguidos planos de execuções, rodando *benchmarks* com diversas possibilidades, permitindo assim comparar o comportamento do ambiente sem a utilização do subsistema e com a utilização do subsistema.

Neste capítulo mostraremos como foi feita a análise dos resultados sobre a utilização do subsistema proposto, bem como ferramentas utilizadas nas medições, e seus recultados.

5.1 Avaliação de Desempenho

Existem três técnicas de avaliação de sistemas: simulação, métodos analíticos e o monitoramento. A simulação possibilita a criação de um modelo com as mesmas características do sistema real, e esse modelo é colocado a prova, porém sendo um modelo lógico, essa abstração nem sempre é fiel ao sistema real. Métodos analíticos são modelos matemáticos, que simulam o sistema com um nível maior de abstração. A técnica de monitoramento foi a escolhida nesse trabalho, onde a avaliação de desempenho avalia a qualidade desse sistema pela capacidade de processamento de uma certa quantidade de dados, ou seja, possibilita por uma avaliação quantitativa de carga, chegar a um índice qualitativo de desempenho. Como temos acesso ao ambiente proposto, tanto de *hardware* quanto em *software*, podemos ter resultados mais próximos ao real.

Neste trabalho, utilizamos *benchmarks* [54], *softwares* com estágios cíclicos, que interagem com o sistema operacional, retirando estatísticas do seu funcionamento, a medida da carga de processamento executada por eles. Através de seus relatórios podemos comparar os diversos cenários e chegar a conclusões de ganho ou perda de desempenho.

5.2 Ferramentas de Avaliação de Desempenho Utilizadas

Para avaliarmos as diferenças entre o uso do escalonador *Credit* e do escalonador *Credit* com o subsistema desenvolvido, escolhemos como *benchmark* o *UnixBench* [55], que vem a ser uma série sintética de *benchmarks*.

Esta série é melhor indicada para encontrar gargalos em partes específicas de subsistemas do *kernel*. O *benchmark UnixBench* utiliza cálculos de dupla precisão em ponto flutuante para ve-

rificar a capacidade de processamento do processador. Avalia também o *throughput* do sistema de arquivos através de leitura, cópia e gravação; calcula o *throughput* do compilador através de algoritmos de recursividade como Torre de Hanói; avalia o *overhead* de chamadas de sistema; criação de processos; concorrência de *shell scripts*.

Para comparar as saídas mostradas pelo *Unixbench*, utilizamos outras 2 ferramentas de *benchmark*, que são o *LMBench* e o TCPW.

O *LMBench* consiste em um *benchmark* sintético, composto de uma série de *micro-benchmarks*, utilizado para mensurar vários aspectos da performance de *kernel*, muito utilizado pela comunidade de desenvolvimento do *kernel* do Linux para encontrar gargalos de subsistemas específicos, contém grande quantidade de *micro-benchmarks* que testam vários aspectos do *hardware* e desempenho do sistema operacional.

O *TCPW* é um *benchmark* que simula operações de compra e venda em uma página de comércio eletrônico, utilizando-se de consultas à banco de dados remotos, possibilitando assim uma análise do desempenho do conjunto máquina/sistema.

5.3 Avaliações de Desempenho

Foram feitas avaliações de desempenho em quatro ambientes distintos utilizando máquinas virtuais, que vem a mostrar o desempenho do subsistema de realocação de recursos frente à configurações estáticas e dinâmicas possíveis com o escalonador do Xen.

As Figuras 13, 14 e 15 mostram através de setas, as execuções individuais dos *benchmarks*, separados por uma linha pontilhada que exibem intervalos de tempo. Por exemplo, na Figura 12 notamos a execução individual de um *benchmark* exibido por uma seta A, após, temos um intervalo de tempo mostrado pela primeira linha pontilhada, e a seguir a execução de outro *benchmark* exibido pela seta B, e com outro intervalo de tempo logo após. Cada *benchmark* roda em intervalos de tempo diferentes, em máquinas virtuais diferentes, possibilitando a simulação de um ambiente de produção que necessite de uma realocação de recursos nova para cada intervalo de tempo.

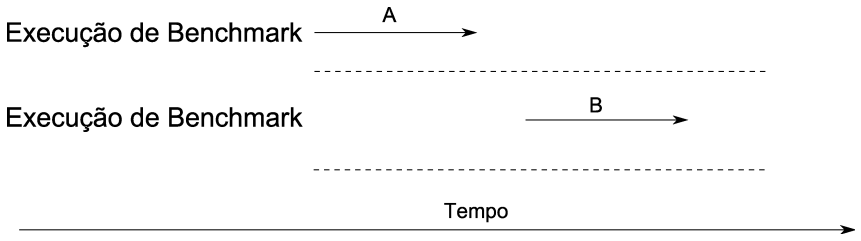

Figura 12 – Exemplo de Teste.

Na Figura 13 o primeiro ambiente de teste, que consiste no escalonador Credit do Xen, sem o subsistema de realocação de recursos, configurado com CAPs (porcentagem de processador)

fixos, com 25% de processamento entre as quatro máquinas virtuais testadas.

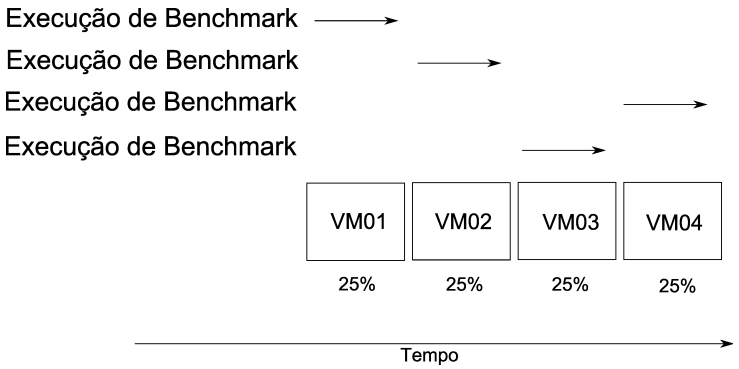

Figura 13 – Teste do Escalonador com CAPs Fixos.

Essa configuração mostra um ambiente estático, onde as máquinas virtuais tem uma porcentagem máxima de recursos (25% de CAP), e mesmo que a aplicação necessite de mais recursos, não os terá. Também, não está configurado a utilização de CPUs virtuais, o que não possibilita reconfigurações de balanceamento de carga automáticas do escalonador. Essa configuração está muito próxima do modo de configuração estática padrão do escalonador, a única diferença é que ao invés de 25% de CAP para cada máquina virtual, a configuração padrão utiliza 0% de CAP.

No segundo ambiente, Figura 14, temos o teste ainda sem a utilização do subsistema de realocação de recursos, porém com CPUs virtuais configuradas, possibilitando balanceamento de carga próprio do escalonador Credit.

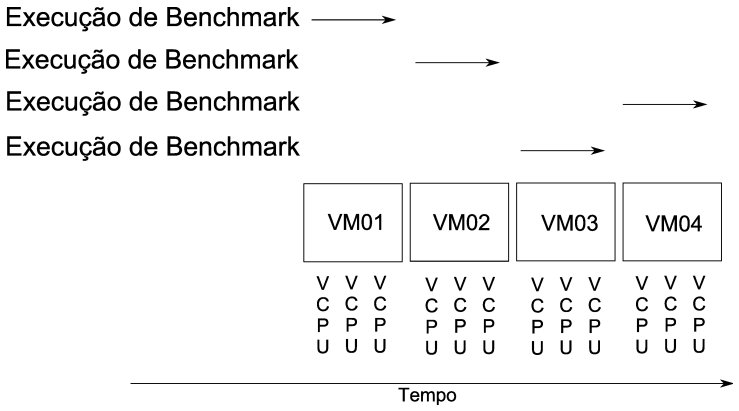

Figura 14 – Teste do Escalonador com CAPs Dinâmicos.

Cada CPU virtual corresponde a uma fatia de processamento de cada processador real, ou seja, se tivermos 3 CPUs virtuais em cada CPU real, cada uma delas corresponde a 33% da utilização deste processador. Isso possibilita um melhor balanceamento de carga, pois CPUs reais que estejam ociosas podem buscar CPUs virtuais de outros processadores vindo a aumentar o desempenho global do sistema.

No terceiro ambiente, visto na Figura 15, temos a mesma tendência dos *benchmarks* nos ambientes testados anteriormente, porém com a utilização do subsistema de realocação de re-

cursos junto ao escalonador, possibilidando à medida da necessidade das máquinas virtuais, reconfiguração de processador e memória desse ambiente.

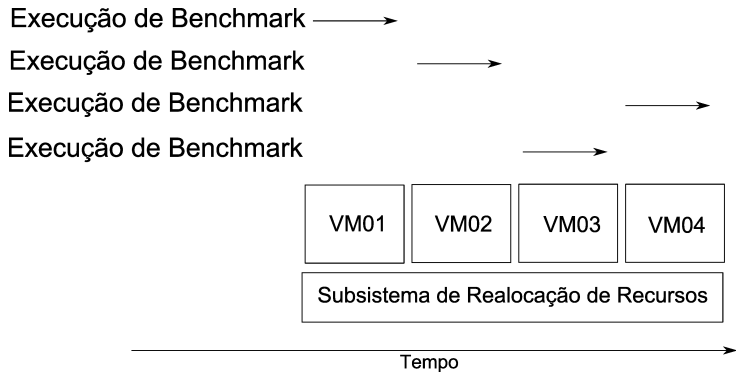

Figura 15 – Teste do Escalonador com Subsistema.

Com essa execução, podemos comparar a utilização do subsistema perante o pior caso que seria a primeira configuração proposta, onde os CAPs são fixos e sem utilização de CPUs virtuais, e o melhor caso onde temos o escalonador configurado para dinamicamente fazer balanceamento de carga utilizando CPUs virtuais.

Com essa série de execuções de *benchmarks* podemos notar as diversas configurações testadas, podendo-se comparar o desempenho do subsistema de realocação de recursos tanto no que se refere ao escalonador dinâmico, quanto a configurações fixas de escalonamento.

5.4 Resultados Obtidos

Foram considerados os ambientes propostos na Seção 5.3, cada resultado é obtido através da média de 10 execuções desprezando-se os extremos. Ao todo foram realizadas 600 execuções. Os testes foram realizados em um computador *HP Integrity rx2600*. Essa máquina é composta de 2 processadores Itanium 2 1.8 Ghz, com 2 Gb de memória RAM (*Read-Only Memory*) .

Em todas as avaliações de desempenho utilizamos os mesmos ambientes, de 3 tipos diferentes de execuções, que são:

- 1. Escalonador com CAPs fixos sem uso de CPUs virtuais (Figura 13);
- 2. Escalonador com o subsistema de realocação de recursos (Figura 15);
- 3. Escalonador com CAPs dinâmicos com uso de CPUs virtuais (Figura 14);

O primeir *benchmark* utilizado foi o *Unixbench*, e seus resultados podem ser acompanhados na Figura 16.

Podemos notar através da Figura 16 que no primeiro ambiente proposto com configurações fixas do escalonador (Figura 13), mostra o pior caso, pois as máquinas virtuais estão com limites

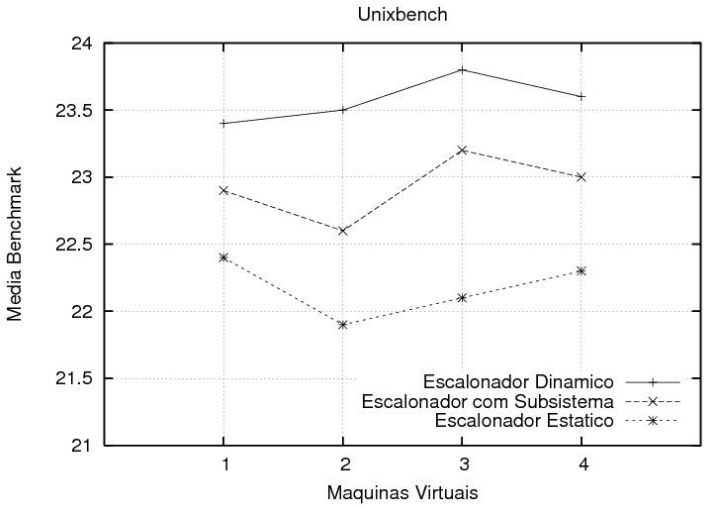

Figura 16 – Desempenho com LMBench.

máximos de processamento para cada uma. Com o escalonador e o subsistema (Figura 15) no segundo ambiente, notamos que o ônus causado pelo subsistema não chega a ser muito relevante, pois permite uma realocação satisfatória para o que se propõe o subsistema, sendo melhor que a configuração estática do escalonador.

Este resultado da utilização do subsistema mostra tendência a melhoria na alocação dos recursos disponibilizados para as máquinas virtuais. Para uma verificação efetiva dessa melhora, é importante que o subsistema possa ser utilizado em um ambiente de produção, ou um conjunto de testes maiores seja executado.

No terceiro ambiente, proposto com a utilização do escalonador em modo dinâmico, notamos que é o melhor desempenho, porém não permite a manipulação através de regras pré-configuradas que o subsistema de realocação de recursos permite, não sendo possível nesse ambiente, limitações de balanceamento entre as máquinas virtuais, não permitindo a utilização de *SLAs*.

Mais dois *benchmarks* foram utilizados, e mostraram que o comportamento do escalonador junto ao subsistema criado tenderam ao mesmo resultado, independente da ferramenta de avaliação utilizada, e são eles: LMBench e TCPW.

O LMBench é um *benchmark* que mede a latência do acesso à sistema de arquivos, e exibe conforme vemos na Figura 17, um ganho de desempenho do escalonador junto ao subsistema de realocação de recursos frente ao escalonador configurado em modo estático. Ainda demonstra o melhor caso com a utilização do escalonador configurado em modo dinâmico, como o *benchmark* Unixbench demonstrou anteriormente. Para estes resultados, executamos 1000 vezes o mesmo *benchmark* nos ambientes apresentados anteriormente para podermos chegar a uma média dos valores apresentados.

Utilizamos também o TCPW, que simula uma operação real de compra *online* com acesso a um banco de dados e permite configuração em três modos, onde temos o modo *Browsing* com

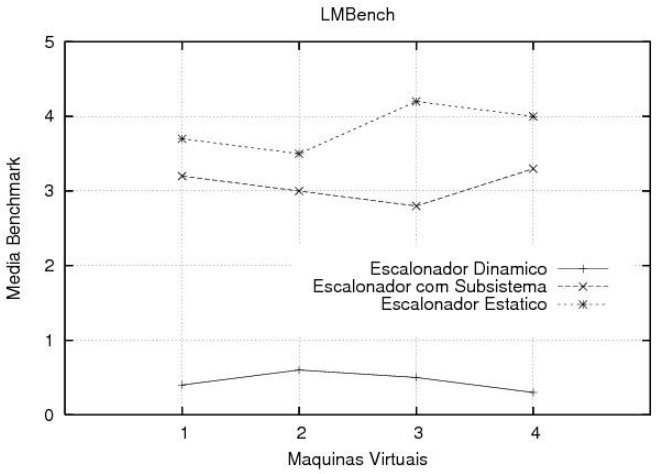

Figura 17 – Desempenho com LMBench.

pouco acesso à base de dados e grande quantidade de acessos ao *site*, o modo *Shopping* com metade da carga sobre o acesso ao *site* e a outra metade de operações no banco de dados, e o modo *Ordering* onde a maioria dos acessos é feito diretamente ao banco de dados com poucas operações diretas no *site*.

Podemos ver na Figura 18 que em modo Browsing o *benchmark* apresenta uma pequena melhora sobre a configuração estática do escalonador, devido à realocação de recursos para as máquinas virtuais serem pouco necessárias, pois o acesso ao sistema de arquivos nesse modo é pequeno, infuenciando mais à rede. Para esse teste, foram realizadas 600 execuções e avaliada a média desses resultados.

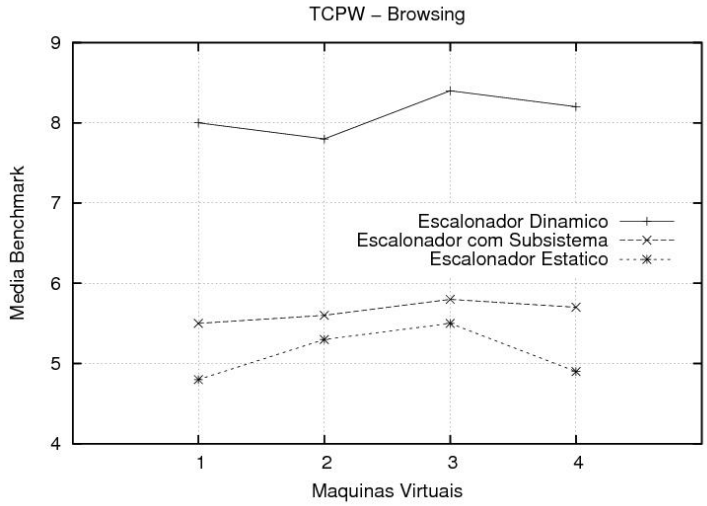

Figura 18 – Desempenho com TCPW - Browsing.

No segundo modo de execução do *benchmark* TCPW, que é o acesso em modo *Shopping*, temos uma melhora mais representativa do escalonador junto ao subsistema de realocação de recursos se comparado com o exemplo anterior pois, como nesse modo temos maior acesso

ao banco de dados, foi exigido do ambiente uma maior realocação de recursos. Notamos na Figura 19 que embora o a configuração do escalonador com o subsistema de realocação não se compare ao escalonador configurado em modo dinâmico, seu comportamento é bem melhor que o escalonador configurado estaticamente. Para esse teste, também foram feitas 600 execuções com esse modo de configuração do TCPW e o resultado é baseado na média das execuções.

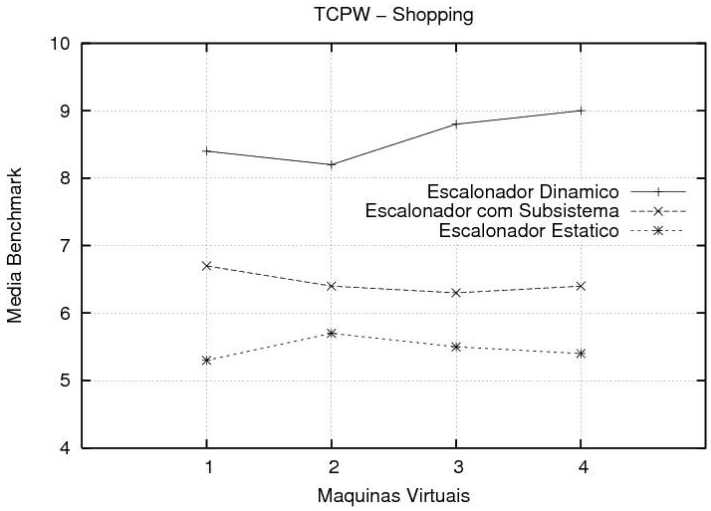

Figura 19 – Desempenho com TCPW - Shopping.

O modo *Ordering* foi o modo que mostrou o melhor resultado, pois necessita de uma maior carga de processamento e consequentemente uma maior realocação de recursos, portanto, como podemos ver na Figura 20, embora o escalonador juntamente com o subsistema de realocação (linha amarela) não tenha sido melhor do que a configuração dinâmica do escalonador, foi a que mais se aproximou desse resultado.

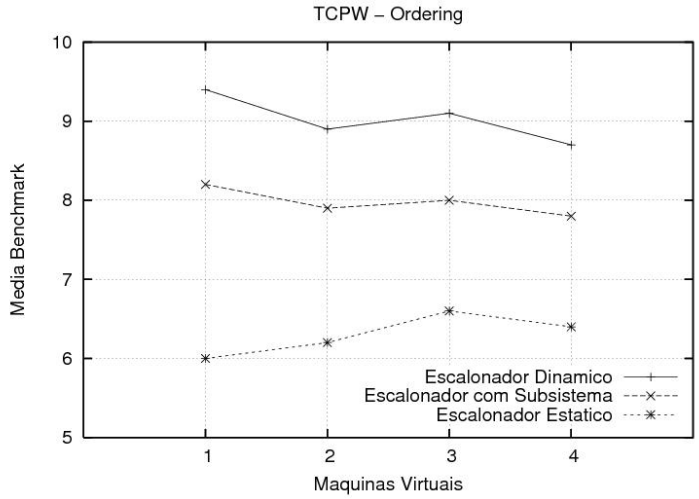

Figura 20 – Desempenho com TCPW - Ordering.

Essas execuções e testes demonstram que embora o subsistema de realocação junto às con-

figurações do escalonador não sejam tão otimizadas quanto o escalonador em modo dinâmico, o subsistema de realocação permite configurações que o escalonador por si só não permite, sem uma perda de desempenho muito importante a ponto de causar um ônus muito grande ao ambiente.

5.5 Considerações Finais

A vantagem na utilização do subsistema de realocação de recursos junto com o escalonador Credit é a possibilidade de mudar as opções de escalonamento através da leitura do arquivo de entrada em XML. O escalonador quando configurado dinamicamente, faz um balanceamento de carga através da migração de CPUs virtuais entre as CPUs reais, porém quando configurado com limites (CAPs) máximos não consegue extrapolar esses limites mesmo que uma máquina virtual necessite de mais processamento e exista processamento ocioso.

Ainda, o subsistema propicia além da funcionalidade do escalonador com configuração de balanceamento de carga, a capacidade adicional de prover a utilização de SLAs através da mudança nas escolhas de caminhos otimizados pela estrutura em XML gerada.

Por exemplo, se quisermos garantir alguma fatia de processamento, e essa configuração consiste em um nodo informado como otimização para a configuração atual pelo XML, podemos não percorrer algum caminho no XML ou alguma configuração específica, não ferindo as regras acordadas por algum SLO (*Service Level Objective*) .

O escalonador *Credit* quando configurado de maneira dinâmica possibilita, embora sem garantias de níveis de serviço, realocação de fatias de processamento destinadas à máquinas virtuais que mais necessitam, porém o subsistema de realocação, além de realocar fatias de processador, também possibilita direcionar fatias de memória para máquinas virtuais que necessitam utilizá-la.

Com as avaliações de desempenho realizados, podemos notar que a utilização do subsistema de realocação de recursos embora não tenha um desempenho igual ao escalonador em modo dinâmico, possibilita uma configuração adicional ao escalonador, que antes não era possível, e que o ônus de sua utilização não chega a ser muito intrusiva ao sistema operacional, o que viabiliza a sua utilização.

6 Conclusão

Esse trabalho tratou de realocação de recursos em máquinas virtuais, com foco especial no escalonador *SMP Credit*, escalonador padrão do monitor de máquinas virtuais Xen. Descrevemos a proposta, implementação e avaliação de um subsistema que realoca recursos como processador e memória para máquinas virtuais do Xen. O subsistema utiliza um grafo para escolher a melhor configuração para todas as máquinas virtuais. Este grafo está configurado para, após a identificação das configurações de processador e memória junto com a necessidade de processamento de determinada máquina virtual, modificar o ambiente para uma nova configuração mais otimizada.

Para geração deste grafo utilizou-se técnicas de mineração de dados. Inicialmente diversas configurações de ambientes com diversas máquinas virtuais foram testadas através da utilização de *benchmarks*. Os resultados destes *benchmarks* foram organizados em uma *data warehouse* devido a grande quantidade de dados gerados pelos *benchmarks* (foram executadas centenas de testes, que geraram milhares de registros). Esta técnica foi utilizada pois a quantia de dados era muito grande para uma análise manual. A estruturação destes dados e a geração do grafo fizeram parte de outra dissertação de mestrado, mas os resultados foram discutidos em conjunto.

Uma vez gerado o grafo, o subsistema proposto por esta dissertação, carrega o mesmo para a memória e toda a reconfiguração de recursos utilizados pelas máquinas virtuais é baseada neste grafo. Conforme apresentamos no Capítulo 5 os resultados obtidos pelo subsistema são promissores, pois já e possível verificar que consegue-se algum ganho com a utilização desta estratégia. Naturalmente, conforme já mencionado, maiores testes devem ainda ser efetuados, de preferência em um ambiente de produção onde as cargas das máquinas virtuais serão reais.

As duas primeiras partes, geração do grafo utilizando técnicas de mineração de dados e o subsistema de realocação de recursos, fazem parte de um projeto maior que pretende utilizar SLAs (Acordos de Nível de Serviço, ou, *Service Level Agreement*) para determinar a forma como os recursos de máquinas virtuais devem ser concedidos/utilizados pelas máquinas virtuais. Atualmente o subsistema não considera acordos de níveis de serviço para a utilzação de recursos por parte das máquinas virtuais. Entendemos que podemos através de nosso sistema estar violando alguns requisitos das máquinas virtuais, mas este não era o objetivo desta dissertação de mestrado. Já está em andamento um trabalho que pretende expandir o subsistema apresentado aqui com a possibilidade de análise de SLAs.

Referências

[1] CREASY, R. J. The origin of the vm/370 time-sharing system. *IBM Journal of Research and Development*, IBM Systems Journal, NY, USA, v. 25, n. 5, p. 483–490, 1981.

[2] SINGH, A.; KORUPOLU, M.; BAMBA, B. Integrated resource aladdress in heterogeneous san data centers. In: *PODC '07: Proceedings of the twenty-sixth annual ACM symposium on Principles of distributed computing*. Portland, Oregon, USA: ACM, 2007. p. 328–329.

[3] APPLICATION virtualization. Acessado em 10/10/2007. Disponível em: <http://thinstall.com/assets/docs/ThinstallWP-ApplicVirtualization4a.pdf>.

[4] OVERVIEW MojoPac. Acessado em 25/10/2007. Disponível em: <http://www.mojopac.com/portal/content/what/>.

[5] OVERVIEW Moka5. Acessado em 26/10/2007. Disponível em: <http://www.moka5.com/products/index.html>.

[6] FALLENBECK, N. et al. Xen and the art of cluster scheduling. In: *VTDC '06: Proceedings of the 2nd International Workshop on Virtualization Technology in Distributed Computing*. Washington, DC, USA: IEEE Computer Society, 2006. p. 4.

[7] BANNON, D. et al. Experiences with a grid gateway architecture using virtual machines. In: *VTDC '06: Proceedings of the 2nd International Workshop on Virtualization Technology in Distributed Computing*. Washington, DC, USA: IEEE Computer Society, 2006. p. 12.

[8] DOORN, L. van. Hardware virtualization trends. In: *VEE '06: Proceedings of the 2nd international conference on Virtual execution environments*. Ottawa, Ontario, Canada: ACM, 2006. p. 45–45.

[9] BARHAM, P. et al. Xen and the art of virtualization. In: *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles*. Bolton Landing, NY, USA: ACM Press, 2003. p. 164–177.

[10] CHERKASOVA, L.; GUPTA, D.; VAHDAT, A. Comparison os the three cpu schedulers in xen. p. 13, May 2007. Disponível em: <http://www.xensource.com/files/xensummit_4/3schedulers-xen-summit_Cherkosova.pdf>.

[11] CASE, R. P.; PADEGS, A. *Architecture of the IBM system/370*. New York, NY, USA, January 1978. v. 21, n. 1, 73–96 p.

[12] SMITH, J. E.; NAIR, R. *Virtual Machines: Versatile plataforms for systems and processes*. San Francisco: Morgan Kauffmann, 2005. 50–68 p.

[13] SMITH, J. E.; NAIR, R. The architecture of virtual machines. *Computer*, IEEE Computer Society Press, Los Alamitos, CA, USA, v. 38, n. 5, p. 32–38, May 2005.

[14] SALOMON, F. A.; TAFURI, D. A. Emulation - a useful tool in the development of computer systems. In: *ANSS '82: Proceedings of the 15th annual symposium on Simulation*. [S.l.]: IEEE Computer Society Press, 1982. p. 55–71.

[15] WALTERS, B. Vmware virtual platform. *Linux Journal*, Specialized Systems Consultants, Inc., Seattle, WA, USA, v. 1999, n. 63es, p. 6, Jul, 1999.

[16] MERGEN, M. F. et al. Virtualization for high-performance computing. *SIGOPS Operating System Review*, ACM Press, New York, NY, USA, v. 40, n. 2, p. 8–11, April 2006.

[17] SCHROEDER, M. D.; SALTZER, J. H. A hardware architecture for implementing protection rings. *Communications of ACM*, ACM Press, New York, NY, USA, v. 15, n. 3, p. 157–170, March 1972.

[18] WALDSPURGER, C. A. Memory resource management in vmware esx server. *SIGOPS Operation System Review*, ACM Press, New York, NY, USA, v. 36, n. SI, p. 181–194, December 2002.

[19] HOSKINS, M. E. User-mode linux. *Linux Journal*, Specialized Systems Consultants, Inc., Seattle, WA, USA, v. 2006, n. 145, p. 2, March 2006.

[20] POTYRA, S.; SIEH, V.; CIN, M. D. Evaluating fault-tolerant system designs using faumachine. In: *EFTS '07: Proceedings of the 2007 workshop on Engineering fault tolerant systems*. Dubrovnik, Croatia: ACM, 2007. p. 9.

[21] OPENVZ Users Guide. Acessado em 15/10/2007. Disponível em: <http://download.openvz.org/doc/OpenVZ-Users-Guide.pdf>.

[22] YANG, L. Teaching system and network administration using virtual pc. *Journal of Computing in Small Colleges*, Consortium for Computing Sciences in Colleges, , USA, v. 23, n. 2, p. 137–142, 2007.

[23] THE new Plex86 x86 Virtual Machine Project. Acessado em 11/09/2007. Disponível em: <http://plex86.sourceforge.net/>.

[24] LAWTON, K. P. Bochs: A portable pc emulator for unix/x. *Linux Journal*, Specialized Systems Consultants, Inc., Seattle, WA, USA, v. 1996, n. 29es, p. 7, September 1996.

[25] BARTHOLOMEW, D. Qemu: a multihost, multitarget emulator. *Linux Journal*, Specialized Systems Consultants Inc., Seattle, WA, USA, v. 2006, n. 145, p. 3, March 2006.

[26] BELLARD, F. Qemu, a fast and portable dynamic translator. In: *USENIX Annual Technical Conference*. Anaheim, CA, USA: ACM Press, 2005. p. 41–46.

[27] PARALLELS Desktop for Mac - Quick Start Guide. Acessado em 01/12/2007. Disponível em: <http://download.parallels.com/GA/Parallels_Desktop_for_Mac_Quick_Start_Guide.pdf>.

[28] GOVIL, K. et al. Cellular disco: resource management using virtual clusters on shared-memory multiprocessors. *ACM Transactions on Computer Systems*, ACM, New York, NY, USA, v. 18, n. 3, p. 229–262, August 2000.

[29] HAND, S. et al. Controlling the XenoServer Open Platform. In: *Proceedings of the Sixteen International Conference on Open Architectures and Network Programming (OPENARCH)*. [S.l.]: IEEE Comunication Society, 2003.

[30] DUDA, K. J.; CHERITON, D. R. Borrowed-virtual-time (bvt) scheduling: supporting latency-sensitive threads in a general-purpose scheduler. In: *SOSP '99: Proceedings of the seventeenth ACM symposium on Operating systems principles*. Charleston, South Carolina, United States: ACM Press, 1999. p. 261–276.

[31] GOVINDAN, S. et al. Xen and co.: communication-aware cpu scheduling for consolidated xen-based hosting platforms. In: *VEE '07: Proceedings of the 3rd international conference on Virtual execution environments*. San Diego, California, USA: ACM Press, 2007. p. 126–136.

[32] ANTONIOU, Z.; STAVRAKAKIS, I. An efficient deadline-credit-based transport scheme for prerecorded semisoft continuous media applications. *IEEE/ACM Transactions on Networking*, IEEE Press, Piscataway, NJ, USA, v. 10, n. 5, p. 630–643, October 2002.

[33] XEN Interface Manual. Acessado em 5/12/2006. Disponível em: <http://www.xen.org/files/xen_interface.pdf>.

[34] INOUE, H. et al. Dynamic security domain scaling on symmetric multiprocessors for future high-end embedded systems. In: *CODES+ISSS '07: Proceedings of the 5th IEEE/ACM international conference on Hardware/software codesign and system synthesis*. Salzburg, Austria: ACM, 2007. p. 39–44.

[35] ZITELLO, T.; WILLIAMS, D.; WEBER, P. *HP OpenView System Administration Handbook: Network Node Manager, Customer Views, Service Information Portal, OpenView Operations*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2003. 112–145 p.

[36] COOK, C. et al. *Managing Peoplesoft with Tivoli*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2000. 78–91 p.

[37] DOMINEY, A.; MEABURN, G. *Microsoft Operations Manager 2005 Field Guide (Expert's Voice)*. Berkely, CA, USA: Apress, 2006. 25–40 p.

[38] IMAMAGIC, E.; DOBRENIC, D. Grid infrastructure monitoring system based on nagios. In: *GMW '07: Proceedings of the 2007 workshop on Grid monitoring*. Monterey, California, USA: ACM, 2007. p. 23–28.

[39] MOORE, A. W.; MCGREGOR, A. J.; BREEN, J. W. A comparison of system monitoring methods, passive network monitoring and kernel instrumentation. *SIGOPS Operation System Review*, ACM Press, New York, NY, USA, v. 30, n. 1, p. 16–38, January 1996.

[40] WISMULLER, R.; TRINITIS, J.; LUDWIG, T. Ocm - a monitoring system for interoperable tools. In: *SPDT '98: Proceedings of the SIGMETRICS symposium on Parallel and distributed tools*. Welches, Oregon, United States: ACM, 1998. p. 1–9.

[41] BUTENHOF, D. R. *Programming with POSIX threads*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1997. 155–170 p.

[42] XUE, L. et al. Dynamic partitioning of processing and memory resources in embedded mpsoc architectures. In: *DATE '06: Proceedings of the conference on Design, automation and test in Europe*. Munich, Germany: European Design and Automation Association, 2006. p. 690–695.

[43] TAN, Z.; LEAL, W.; WELCH, L. Verification of instrumentation techniques for resource management of real-time systems. *Journal of Systems and Software*, Elsevier Science Inc., New York, NY, USA, v. 80, n. 7, p. 1015–1022, July 2007.

[44] WALKER, E. et al. Personal adaptive clusters as containers for scientific jobs. *Cluster Computing*, Kluwer Academic Publishers, Hingham, MA, USA, v. 10, n. 3, p. 339–350, September 2007.

[45] YANG, C.-T. et al. Improvements on dynamic adjustment mechanism in co-aladdress data grid environments. *Journal of Supercomputing*, Kluwer Academic Publishers, Hingham, MA, USA, v. 40, n. 3, p. 269–280, May 2007.

[46] BERTOLINO, A. et al. Scaling up sla monitoring in pervasive environments. In: *ESSPE '07: International workshop on Engineering of software services for pervasive environments*. Dubrovnik, Croatia: ACM, 2007. p. 65–68.

[47] WU, T.; BISWAS, S. Minimizing inter-cluster interference by self-reorganizing mac aladdress in sensor networks. *Wirelles Networks*, Kluwer Academic Publishers, Hingham, MA, USA, v. 13, n. 5, p. 691–703, August 2007.

[48] BECKER, K. et al. Spdw: A software development process performance data warehousing environment. In: *SEW '06: Proceedings of the 30th Annual IEEE/NASA Software Engineering Workshop*. Washington, DC, USA: IEEE Computer Society, 2006. p. 107–118.

[49] HAN, J.; KAMBER, M. *Data mining: concepts and techniques*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2000.

[50] QUINLAN, J. R. *C4.5: programs for machine learning*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1993.

[51] EASTLAKE, D.; REAGLE, J.; SOLO, D. *Extensible Markup Language - XML-Signature Syntax and Processing*. NY, USA: W3C Press, February 2002.

[52] CHEN, Y.; IYER, S. SLA decomposition: Translating service level objectives to system level thresholds. *The 4th IEEE International Conference on Autonomic Computing*, IEEE Computer Society, Washington, DC, USA, p. 1–11, 2007.

[53] WANG, Z.; CHENG, H. H. Portable c/c++ code for portable xml data. *IEEE Software*, IEEE Computer Society Press, Los Alamitos, CA, USA, v. 23, n. 1, p. 76–81, January 2006.

[54] KALIBERA, T. et al. Automated benchmarking and analysis tool. In: *VALUETOOLS '06: Proceedings of the 1st international conference on Performance evaluation methodolgies and tools*. Pisa, Italy: ACM, 2006. p. 5.

[55] THE BYTE's Unix Benchmark Suite. Acessado em 04/04/2006. Disponível em: <`http://www.tux.org/pub/tux/niemi/unixbench/`>.

ApêndiceA – Arquivo XML

Exemplo do arquivo de XML com a estrutura de dados com as configurações das máquinas virtuais bom como suas otimizações. Esse exemplo mostra uma parte do arquivo com configurações para 4 máquinas virtuais.

```
<?xml version="1.0"?>

<arvore>

   <arqx86>
      <numvm01>
      </numvm01>

      <numvm02>
      </numvm02>

      <numvm03>
      </numvm03>

      <numvm04>

<nodo name="7">
<vm01cap>10</vm01cap>
<vm01mem>130</vm01mem>
<vm02cap>10</vm02cap>
<vm02mem>70</vm02mem>
<vm03cap>10</vm03cap>
<vm03mem>40</vm03mem>
<vm04cap>10</vm04cap>
<vm04mem>40</vm04mem>
<otimizacao>103</otimizacao>
<otimizacao>102</otimizacao>
</nodo>

        <nodo name="103">
<vm01cap>10</vm01cap>
<vm01mem>70</vm01mem>
<vm02cap>10</vm02cap>
<vm02mem>70</vm02mem>
<vm03cap>10</vm03cap>
<vm03mem>70</vm03mem>
<vm04cap>10</vm04cap>
<vm04mem>70</vm04mem>
```

```
<otimizacao>17</otimizacao>
</nodo>

<nodo name="17">
<vm01cap>20</vm01cap>
<vm01mem>70</vm01mem>
<vm02cap>20</vm02cap>
<vm02mem>70</vm02mem>
<vm03cap>20</vm03cap>
<vm03mem>70</vm03mem>
<vm04cap>20</vm04cap>
<vm04mem>70</vm04mem>
<otimizacao>8</otimizacao>
</nodo>

<nodo name="8">
<vm01cap>25</vm01cap>
<vm01mem>70</vm01mem>
<vm02cap>25</vm02cap>
<vm02mem>70</vm02mem>
<vm03cap>25</vm03cap>
<vm03mem>70</vm03mem>
<vm04cap>25</vm04cap>
<vm04mem>70</vm04mem>
</nodo>

<nodo name="102">
<vm01cap>10</vm01cap>
<vm01mem>130</vm01mem>
<vm02cap>10</vm02cap>
<vm02mem>70</vm02mem>
<vm03cap>10</vm03cap>
<vm03mem>40</vm03mem>
<vm04cap>20</vm04cap>
<vm04mem>40</vm04mem>
<otimizacao>69</otimizacao>
<otimizacao>48</otimizacao>
</nodo>

<nodo name="69">
<vm01cap>40</vm01cap>
<vm01mem>70</vm01mem>
<vm02cap>15</vm02cap>
<vm02mem>70</vm02mem>
<vm03cap>10</vm03cap>
<vm03mem>70</vm03mem>
<vm04cap>20</vm04cap>
<vm04mem>70</vm04mem>
```

```
<otimizacao>83</otimizacao>
<otimizacao>89</otimizacao>
</nodo>

<nodo name="83">
<vm01cap>67</vm01cap>
<vm01mem>70</vm01mem>
<vm02cap>15</vm02cap>
<vm02mem>70</vm02mem>
<vm03cap>10</vm03cap>
<vm03mem>70</vm03mem>
<vm04cap>10</vm04cap>
<vm04mem>70</vm04mem>
</nodo>

<nodo name="89">
<vm01cap>65</vm01cap>
<vm01mem>130</vm01mem>
<vm02cap>15</vm02cap>
<vm02mem>70</vm02mem>
<vm03cap>10</vm03cap>
<vm03mem>40</vm03mem>
<vm04cap>10</vm04cap>
<vm04mem>40</vm04mem>
</nodo>

<nodo name="48">
<vm01cap>10</vm01cap>
<vm01mem>70</vm01mem>
<vm02cap>15</vm02cap>
<vm02mem>70</vm02mem>
<vm03cap>25</vm03cap>
<vm03mem>70</vm03mem>
<vm04cap>35</vm04cap>
<vm04mem>70</vm04mem>
<otimizacao>68</otimizacao>
<otimizacao>55</otimizacao>
</nodo>

<nodo name="68">
<vm01cap>30</vm01cap>
<vm01mem>130</vm01mem>
<vm02cap>15</vm02cap>
<vm02mem>70</vm02mem>
<vm03cap>20</vm03cap>
<vm03mem>40</vm03mem>
<vm04cap>35</vm04cap>
<vm04mem>40</vm04mem>
```

```
</nodo>

<nodo name="55">
    <vm01cap>25</vm01cap>
<vm01mem>70</vm01mem>
<vm02cap>15</vm02cap>
<vm02mem>70</vm02mem>
<vm03cap>25</vm03cap>
<vm03mem>70</vm03mem>
<vm04cap>25</vm04cap><vm04mem>70</vm04mem>
</nodo>
      </numvm04>
   </arqx86>

   <arqia64>
   </arqia64>

</arvore>
```