Universidade do Estado do Rio de Janeiro

Centro de Tecnologia e Ciências

Faculdade de Engenharia

Marcus Vinícius do Patrocínio Azevedo

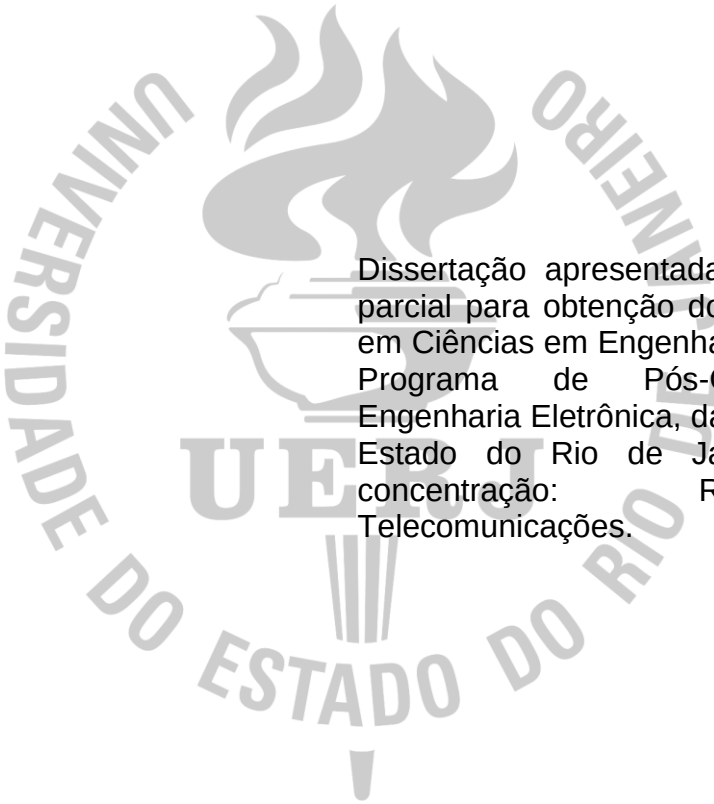
Otimização de recursos e economia de energia em clusters usando virtualização

Rio de Janeiro

2010

Marcus Vinícius do Patrocínio Azevedo

Otimização de recursos e economia de energia em clusters usando virtualização



Dissertação apresentada, como requisito parcial para obtenção do título de Mestre em Ciências em Engenharia Eletrônica, ao Programa de Pós-Graduação em Engenharia Eletrônica, da Universidade do Estado do Rio de Janeiro. Área de concentração: Redes de Telecomunicações.

Orientador: Prof. Dr. Alexandre Sztajnberg

Rio de Janeiro

2010

CATALOGAÇÃO NA FONTE

UERJ / REDE SIRIUS / BIBLIOTECA CTC/B

A994 Azevedo, Marcus Vinícius do Patrocínio.
Otimização de recursos e economia de energia em clusters usando virtualização / Marcus Vinícius do Patrocínio Azevedo. - 2010.
76 f. : il.

Orientador: Alexandre Sztajnberg.
Dissertação (Mestrado) – Universidade do Estado do Rio de Janeiro, Faculdade de Engenharia.

1. Redes de computação – Teses. 2. Virtualização – Teses. 3. Gerência de recursos – Teses. 4. Servidores web – Teses. 5. Clusters de computadores – Teses. 6. Engenharia Eletrônica. I. Sztajnberg, Alexandre. II. Universidade do Estado do Rio de Janeiro. III. Título.

CDU 004.72.057.4

_____ _____

Assinatura Data

Marcus Vinícius do Patrocínio Azevedo

Otimização de recursos e economia de energia em clusters usando virtualização

Dissertação apresentada, como requisito parcial para obtenção do título de Mestre em Ciências em Engenharia Eletrônica, ao Programa de Pós-Graduação em Engenharia Eletrônica, da Universidade do Estado do Rio de Janeiro. Área de concentração: Redes de Telecomunicações.

Aprovada em 19 de julho de 2010.

Banca examinadora:

Prof. Dr. Alexandre Sztajnberg (Orientador)
Faculdade de Engenharia - UERJ

Prof. Dr. Marcelo Gonçalves Rubinstein
Faculdade de Engenharia - UERJ

Prof. Dr. Julius Cesar Barreto Leite
Instituto de Computação - UFF

Rio de Janeiro

2010

AGRADECIMENTOS

Acima de todas as coisas agradeço a Deus, por me permitir concluir esta etapa da minha vida. Cada vez mais tenho certeza de que se não for por Sua vontade, em vão são todas as realizações do homem.

Aos professores Julius Leite e Orlando Loques da UFF por me propiciarem contato com o grupo do laboratório TEMPO, assim como permitir acesso aos recursos do Instituto de Computação e às pesquisas lá desenvolvidas.

Aos colegas da UFF, em especial, ao Vinícius Petrucci e ao Carlos Sant´ana. A troca de experiências sobre virtualização e os trabalhos por eles desenvolvidos foram de grande importância para a proposta aqui apresentada. O sucesso em suas carreiras como futuros doutores é só uma questão de tempo.

Aos meus companheiros de mestrado Flávia e Felipe. Juntos nessa caminhada, o incentivo mútuo entre nós foi fundamental, ou como diria o Felipe, “sensacional”.

Principalmente ao meu orientador e professor Alexandre Sztajnberg. Da mesma forma que uma bússola orienta o navegador, o professor Sztajnberg soube apontar o “norte” do trabalho. Ele contribuiu pessoalmente na solução dos problemas com o laboratório, sempre me incentivando e valorizando os meus esforços.

RESUMO

AZEVEDO, Marcus Vinícius do Patrocínio. *Otimização de recursos e economia de energia em clusters usando virtualização*. 76 f. Dissertação (Mestrado em Engenharia Eletrônica) - Programa de Pós-Graduação em Engenharia Eletrônica, Universidade do Estado do Rio de Janeiro, Rio de Janeiro, 2010.

Este trabalho propõe uma arquitetura reutilizável que permite a gerência de uma infraestrutura de suporte para aplicações Web, utilizando máquinas virtuais. O objetivo da arquitetura é garantir qualidade de serviço, atuando nos servidores físicos (hospedeiros) ou manipulando as máquinas virtuais, e avaliando o cumprimento das restrições de operação (tempo de resposta máximo). Além disso, através do uso racional dos recursos utilizados, a proposta visa à economia de energia. O trabalho também inclui uma avaliação de desempenho realizada sobre um sistema implementado com base na arquitetura. Esta avaliação mostra que a proposta é funcional e o quanto ela pode ser vantajosa do ponto de vista do uso de recursos, evitando desperdício, mantendo-se ainda a qualidade de serviço em níveis aceitáveis pela aplicação.

Palavras-chave: Virtualização. Economia de Energia. Gerência de Recursos. Servidores Web.

ABSTRACT

This work proposes a reusable architecture that enables the management of a supporting infrastructure for Web applications using virtual machines. The goal of the architecture is to ensure quality of service, acting on physical servers (hosts) or manipulating the virtual machines, and evaluating how broadly it complies with the operating restrictions (maximum response time). In addition, through the rational use of resources, the proposal aims at saving energy. The work also includes a performance evaluation carried out over a system implemented based on the architecture. This evaluation shows that the proposal is fully functional and how it can be advantageous in terms of use of resources, avoiding waste, yet maintaining the application's quality of service within acceptable levels.

Keywords: Virtualization. Energy Saving. Resource Management. Web Servers.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS

API	Application Program Interface
CPU	Central Processing Unit
DVFS	Dynamic Voltage and Frequency Scaling
HTML	Hyper-Text Markup Language
HTTP	Hyper-Text Transfer Protocol
IC	Intervalo de confiança
IP	Internet Protocol
KWh	Quilowatt-hora
MAC	Medium Access Control
MMV	Monitor de Máquina Virtual
MV	Máquina Virtual
NFS	Network File System
QoS	Quality of Service
RPC	Remote Procedure Call
RPM	Red Hat Package Manager
SNMP	Simple Network Management Protocol
SSH	Secure Shell
TCP	Transmission Control Protocol
TI	Tecnologia da Informação
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
USB	Universal Serial Bus
VA	Volt-ampere
XML	Extensible Markup Language

SUMÁRIO

INTRODUÇÃO

Com a atual preocupação com o uso de recursos naturais e com o meio ambiente, uma das questões a serem tratadas é a economia de energia. Em sistemas computacionais mais elaborados tais como grades computacionais, *clusters* ou em Centrais de Processamentos de Dados (CPDs), a redução do consumo de energia passa a ser importante sob o ponto de vista econômico. Em 2007, a consultoria Gartner já previa que em 2010, cerca de metade das empresas que fazem parte da lista *Forbes Global 2000* iria gastar mais com energia do que com *hardware* (servidores) [1].

A discussão vai da responsabilidade socioambiental das organizações (adoção de medidas da "TI verde") indo até os aspectos estratégicos, pois o crescimento do parque de equipamentos em função do aumento das demandas dos clientes e sistemas implica em adquirir mais espaço, além de maior necessidade de refrigeração. Entretanto, aspectos importantes como a qualidade de serviço oferecida (ou exigida pelo usuário que está pagando por ela) não devem ser negligenciados em função da economia de energia ou de outros fatores.

A maturidade dos sistemas operacionais, sistemas de gerência e dos equipamentos em si permitem o gerenciamento e controle sobre esses elementos de forma a acompanhar o uso dos recursos, a qualidade dos serviços ofertados e atuar sobre os mesmos através de medidas como, por exemplo, desligar componentes que não estão em uso em determinados momentos ou reconfigurar parâmetros de operação (frequência da CPU, por exemplo). A idéia geral é utilizar menos recursos, porém provendo serviços com a melhor qualidade possível. Refinando esta idéia, a gerência dos recursos pode ser realizada com base em políticas definidas pelos responsáveis por tais sistemas e pelos usuários dos serviços de forma se obter um compromisso entre qualidade, desempenho e custo.

A tecnologia da virtualização, surgida na década de 60 para viabilizar o uso dos *mainframes*, ganha novo enfoque nos dias de hoje, sendo utilizada para permitir que diversos sistemas operacionais distintos possam ser executados sobre um único *hardware*. Isso torna viável a consolidação de servidores dado um conjunto de máquinas físicas o que vai ao encontro dos aspectos de economia de recursos, diminuição do espaço necessário, redução da energia elétrica consumida e da energia térmica gerada, diminuindo também a necessidade de arrefecimento do ambiente e consequentemente gerando mais economia. Com a virtualização também é possível migrar máquinas virtuais entre servidores físicos sem interrupção dos

serviços. Isso abre a possibilidade para que, em determinados momentos, a quantidade de servidores ligados seja reduzida de acordo com a demanda dos serviços oferecidos, sem prejuízo da qualidade destes e permitindo a otimização dos recursos utilizados.

Diversos trabalhos nesta área de pesquisa tratam as questões do melhor aproveitamento dos recursos visando à qualidade de serviços, porém nem sempre endereçam diretamente a economia de energia. Ainda há trabalhos que propõem tratar esta questão, mas não apresentam um modelo reutilizável que possa ser adaptado, por exemplo, para sistemas não virtualizados.

Embora as soluções de virtualização existentes forneçam suas próprias APIs para as operações de gerência, controle e atuação no ambiente, as mesmas são dependentes do produto o que dificultaria a sua utilização caso a solução escolhida para a implementação fosse substituída. Apesar de já existirem propostas de API independentes de solução, estas se mostram incompletas especialmente em operações importantes do ponto de vista da otimização de recursos, como por exemplo, permitir desligamento de núcleos de processadores.

Este trabalho propõe uma arquitetura reutilizável que permite a gerência de uma infraestrutura de suporte para aplicações Web, utilizando máquinas virtuais, porém atuando nos servidores físicos (hospedeiros) tendo como parâmetro as restrições de operação da aplicação, visando à otimização de recursos, qualidade de serviços e economia de energia. A arquitetura é reutilizável, pois o modelo concebido permite adaptação para outros tipos de infraestrutura ou de outros tipos de aplicação. A flexibilidade da arquitetura também é possível graças a uma API também desenvolvida neste trabalho que fornece uma camada de alto nível contendo as operações necessárias para atuar no ambiente, mas de forma independente do virtualizador utilizado.

O trabalho está estruturado da seguinte forma:

- no Capítulo 1 são apresentados alguns dos trabalhos relacionados à gerência de ambientes com qualidade de serviços, visando direta ou indiretamente à economia de energia. Durante a elaboração deste trabalho, foram estudados cerca de 70 trabalhos ligados ao tema, porém foram destacados os que mais se relacionavam com a proposta aqui apresentada;
- o Capítulo 2 introduz a tecnologia da virtualização, apresentando rapidamente o histórico, técnicas, funcionalidades e benefícios. Neste capítulo também é apresentada a solução de virtualização utilizada na implementação do ambiente de máquinas virtuais e hospedeiros;

- no Capítulo 3 é apresentada a arquitetura proposta, onde são mostrados os elementos que a compõem, o funcionamento e a API desenvolvida. Durante todo o capítulo são discutidos os aspectos que levaram à concepção do modelo e outras definições;
- no Capítulo 4 é apresentada uma avaliação de desempenho utilizando uma implementação de sistema de gerência baseado na arquitetura proposta. Também são descritos neste capítulo o ambiente e as ferramentas utilizadas na implementação e na avaliação, além de testes preliminares. O objetivo é mostrar que a proposta é funcional e o quanto ela pode ser vantajosa do ponto de vista de otimização de recursos;
- o Capítulo 5 apresenta as conclusões e os futuros trabalhos.

1 TRABALHOS RELACIONADOS

Neste capítulo serão apresentadas soluções e propostas relacionadas ao nosso trabalho. São mostradas diferentes abordagens sobre o uso de virtualização e controle do uso de recursos computacionais, visando direta ou indiretamente às questões da qualidade de serviço e economia de energia. Para cada abordagem são feitas uma pequena descrição e uma avaliação sobre as estratégias adotadas. Também é discutida a contribuição deste trabalho em relação aos demais.

O trabalho de [2] propõe uma arquitetura de gerência que provê balanceamento de carga utilizando migração de máquinas virtuais (MVs) em um conjunto de hospedeiros. Quando a utilização de CPU de um dos hospedeiros está acima de um limite previamente definido, o sistema migra as MVs para outros hospedeiros com mais recursos disponíveis. Desta forma, a proposta utiliza o uso de recursos dos hospedeiros e não o desempenho das aplicações hospedadas, na tomada de decisão de reconfiguração do ambiente. A estratégia adotada de migrar todas as MVs do hospedeiro sobrecarregado pode levar o sistema a repetidas readaptações, já que logo em seguida o hospedeiro, que antes tinha recursos e passou a ficar sobrecarregado, pode sofrer uma nova reconfiguração em função da última migração. Em nossa proposta, após a migração de uma MV é feita uma nova coleta de informações para avaliar se isso foi suficiente para equilibrar o uso dos recursos do ambiente ou se ainda é necessário efetuar novas migrações ou reconfigurações. O trabalho proposto por [2] não endereça diretamente a questão da economia de energia, não faz uso do mecanismo de ajuste de frequência dos processadores (DVFS) e desativação de hospedeiros ociosos, em contraste com a nossa proposta.

Em [3] é mostrada uma arquitetura de controle coordenada para otimização de recursos alocados para MVs. A estratégia é fazer uso de *control loops* que realizam continuamente: (i) medição do desempenho (tempo de resposta, por exemplo), (ii) cálculo dos ajustes necessários com base na diferença entre o valor medido e o valor ideal e (iii) reconfiguração do ambiente de acordo com o calculado anteriormente. O trabalho propõe o uso de dois *control loops*: um baseado no tempo de resposta de uma aplicação Web e outro baseado no consumo de energia dos hospedeiros. A arquitetura chama-se coordenada, pois os dois *control loops* são combinados de modo a se obter economia de energia sem perda de desempenho na aplicação. A proposta descarta o recurso de desativação de hospedeiros, pois os autores consideram a operação de ligar e desligar custosa (em termos de tempo e energia

consumida) se comparado com o uso do DVFS simplesmente. Em nossa proposta, utilizamos tanto DVFS quanto desativação de hospedeiros e será apresentada a estratégia adotada para minimizar o tempo de espera da reativação dos hospedeiros.

Com objetivo similar à proposta anterior, no estudo apresentado por [4] é utilizado um controlador *lookahead* que, baseado num conjunto de variáveis, tais como, quantidade de conexões e tempo de resposta médio, entre outras, ajusta o ambiente definindo a quantidade de hospedeiros que devem ser ativados e a quantidade de recursos que deve ser alocada para cada MV. A autora utilizou esse tipo de controlador com um caráter preditivo, pois, as reconfigurações do ambiente são otimizadas de acordo com a demanda do serviço provido e com as prováveis mudanças (iminente aumento de carga, por exemplo). No nosso trabalho, não são utilizados controles preditivos, pois consideramos que a inclusão dessa abordagem poderia aumentar a complexidade da arquitetura proposta dependendo do conjunto de variáveis de entrada e controles escolhidos. Entretanto, pretende-se no futuro estudar mecanismos desta natureza, colocando na entrada informações sobre consumo de energia, já que o trabalho apresentado por [4] não incluiu este item em seu controlador *lookahead*.

No trabalho apresentado por [5] é proposta a adaptação autonômica de sistemas virtualizados num ambiente composto por múltiplos domínios. Cada domínio é formado por um hospedeiro e suas MVs. A estratégia adotada foi a de alocar recursos paras as MVs ajustando-se o peso de CPU que o virtualizador atribui para cada uma delas de acordo com a demanda, assim como ajustando a quantidade de memória que a MV necessita. Caso num domínio não seja possível alocar mais recursos para as MVs, o sistema pode buscar outro domínio que possua recursos disponíveis para realizar a migração de MVs. Nesse estudo, o mecanismo de adaptação é baseado na utilização de CPU e de memória das MVs e não no tempo de resposta da aplicação ao contrário do nosso trabalho. Da mesma forma, no trabalho apresentado por [6] que também aborda alocação de recursos computacionais voltados para qualidade de serviços, o controle de adaptação baseia-se no uso de CPU e memória ao invés de considerar a aplicação.

Em [7] é apresentada uma pesquisa que utiliza um ambiente composto por hospedeiros e máquinas virtuais para prover serviços em computação em nuvem [8]. Nessa proposta é utilizado o tempo de resposta das aplicações em execução nas MVs na avaliação de reconfiguração do ambiente. Caso o tempo de resposta esteja fora das restrições, um mecanismo de adaptação é disparado para instanciar uma nova MV para que essa possa receber requisições. A estratégia de criação de novas MVs sob demanda pode causar esgotamento de recursos físicos nos hospedeiros. Em nossa proposta optou-se por

dimensionar previamente, através de testes, a quantidade máxima de requisições suportadas pelo ambiente e o número de MVs suficiente para suportar a demanda. Caso todos os recursos sejam esgotados a idéia é que o sistema possa recomendar ao administrador do ambiente que aumente os recursos disponíveis (por exemplo, novos hospedeiros ou CPUs adicionais) para atender à demanda dos serviços.

O trabalho de [9] propõe uma arquitetura para adaptação dinâmica em servidores Web considerando os aspectos de economia de energia e qualidade de serviço. Na proposta, um aglomerado de servidores Web formados por servidores físicos recebe conexões através de um balanceador de carga (*frontend*). Os servidores físicos têm suas frequências de CPU ajustadas dinamicamente e servidores ociosos são desligados, tudo de acordo com o tempo de resposta da aplicação. Embora esse estudo não aborde servidores virtualizados, o nosso trabalho se baseou na estratégia de controle de adaptação (*control loop*) usada pelo autor assim como no uso de DVFS e desativação de servidores. Outros trabalhos do mesmo grupo de pesquisa do laboratório TEMPO, da Universidade Federal Fluminense, também foram avaliados. Como exemplos, os trabalhos de [10] (mencionados nas Seções 3.4.2 e 4.1.4) e de [11], que abordam questões importantes relacionadas a QoS em arquiteturas de *clusters* Web, através de um tratamento estatístico. Além dos trabalhos avaliados, houve a interação direta com os autores mencionados e demais integrantes do TEMPO, por conta de projeto de pesquisa em comum.

2 VIRTUALIZAÇÃO

Este capítulo apresenta a virtualização em sistemas computacionais, os principais conceitos e outras questões relacionadas ao trabalho aqui proposto. Além de mostrar a tecnologia, será apresentado o virtualizador de código aberto XEN uma das soluções mais utilizadas em trabalhos acadêmicos e que também foi escolhido na implementação da proposta deste trabalho.

2.1 Tecnologia de virtualização

Nos anos 60, com o objetivo de melhor aproveitar o grande poder de processamento dos *mainframes*, foi criada uma técnica para permitir que diversas máquinas virtuais compartilhassem o mesmo *hardware* subjacente [12]. Com o surgimento dos computadores pessoais e com a redução dos custos de produção levando ao surgimento de servidores, deixou de ser interessante ou necessário utilizar técnicas de virtualização ou particionamento de sistemas computacionais.

Entretanto com o alto poder dos atuais processadores *multicore*, o uso da virtualização voltou a ser conveniente em ambientes com capacidade de multiplexação de diversos sistemas operacionais sobre um mesmo *hardware*, pois além de permitir o aproveitamento mais racional dos recursos, traz outras vantagens, tais como:

- Isolamento entre os sistemas virtuais e o *hardware*, promovendo segurança;
- Facilidade de migração de sistemas inteiros (máquinas virtuais) entre servidores distintos, provendo tolerância a falhas e otimização de recursos físicos;
- Consolidação de servidores, considerando que sistemas reais podem ser virtualizados e executados sobre uma quantidade menor de máquinas reais, gerando, assim, economia de espaço físico e de energia elétrica;
- Escalabilidade ao permitir a criação de um novo sistema virtual para ser executado sobre um mesmo servidor, podendo, inclusive, esse sistema ser um clone de um sistema virtual existente;
- Qualidade de serviço ao permitir o aumento de recursos físicos a fim de atender a uma determinada demanda de serviços e ao mesmo tempo poder diminuir tais recursos quando a demanda for menor, tornando mais eficiente o uso;

- Permitir a virtualização de dispositivos o que pode tornar possível atuar dinamicamente nestes alterando suas características de maneira transparente para o ambiente e para seus usuários.

Quanto aos conceitos utilizados em virtualização, denomina-se hospedeiro o *hardware* ou servidor utilizado para suportar a execução das máquinas virtuais hóspedes ou, simplesmente, MVs. A virtualização faz uso de uma camada de *software* chamada Monitor de Máquinas Virtuais (MMV) ou *hypervisor* para permitir a gerência e funcionamento das MVs, além de fornecer uma abstração dos recursos físicos a serem utilizados pelas mesmas. A abordagem mais comum é o MMV ser executado diretamente entre o *hardware* e as MVs (Figura 1), embora o MMV também possa ser executado entre o sistema operacional do hospedeiro e as MVs, numa abordagem conhecida como virtualização hospedada. As MVs executam sistemas operacionais independentes do sistema do hospedeiro e têm a ilusão de estarem acessando direta e exclusivamente os recursos físicos, quando, na verdade, estes são compartilhados, entre diversos sistemas virtuais.

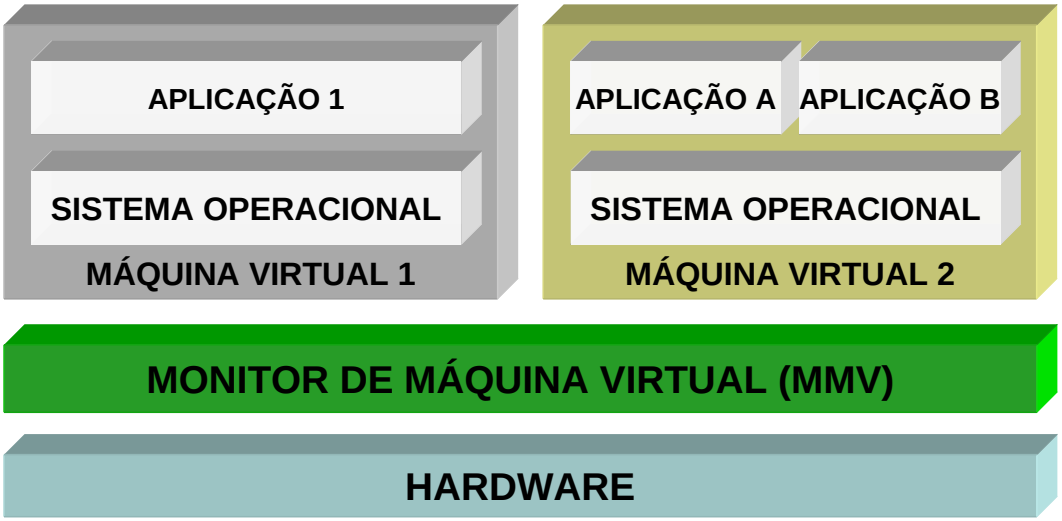


Figura 1 – Modelo de sistema com virtualização

As soluções de virtualização podem ser classificadas de acordo com o tipo de técnica de virtualização que as mesmas utilizam, a saber:

- **Virtualização completa**: nesta técnica os recursos físicos do hospedeiro são virtualmente replicados para as MVs através de abstrações o que permite que o sistema operacional e as aplicações possam ser executados exatamente como se fosse num sistema real, não necessitando alterações e de forma independente do

hardware [13]. Esta técnica tem a desvantagem de incluir uma sobretaxa na execução de instruções pela CPU, pois o MMV deve detectar instruções privilegiadas e instruções sensíveis que eventualmente a MV tente executar;

- **Paravirtualização**: nesta técnica o núcleo do sistema operacional do hospedeiro é modificado para permitir a execução concorrente com outros sistemas operacionais [13]. Diferente da técnica anterior, o MMV não precisa inspecionar as instruções despachadas pela MV para a CPU, pois as chamadas de instruções privilegiadas e sensíveis são substituídas por *hypercalls* no núcleo modificado, sendo o controle nessas chamadas passado para o MMV tratar de maneira mais adequada. Com isso não há a sobretaxa de inspeção de instruções o que torna esta técnica mais eficiente em relação à anterior;
- **Virtualização assistida por *hardware***: nesta técnica o processador provê suporte para virtualização o que suprime a sobretaxa da inspeção de instruções anteriormente citada, sendo útil quando os sistemas não podem ter seu núcleo modificado (sistemas de código fechado, por exemplo) e se deseja aproveitar o desempenho da paravirtualização. O suporte do processador também visa ao gerenciamento de memória virtual de forma mais eficiente que nas técnicas anteriores, provendo também desempenho.

Entre as diversas vantagens que o uso da virtualização apresenta, a migração de máquinas virtuais merece um destaque especial, tendo em vista que esse é um dos mecanismos utilizados no suporte da arquitetura proposta neste trabalho.

Embora em ambientes de computacionais distribuídos, como por exemplo, as grades, seja possível migrar aplicações (processos) entre nós distintos, é necessária a implementação de técnicas de *checkpointing* para garantir que a execução possa prosseguir após a migração, sem necessidade de reiniciar as aplicações. Estratégias que utilizam (i) pré-compiladores que inserem código adicional em pontos do código-fonte ou (ii) mecanismos de captura dos dados da pilha de execução das aplicações [14], requerem interferência direta na aplicação que será suportada no ambiente, o que pode ser de difícil implementação caso não se tenha acesso ao código fonte ou caso a linguagem utilizada não forneça suporte adequado.

Entretanto, o uso de virtualização suporta mais facilmente os mecanismos de migração de máquinas do que a limitada migração de processos, pois, pode permitir a movimentação de um sistema operacional completo entre servidores físicos, juntamente com as aplicações em execução, os descritores de arquivos abertos e suas conexões de rede, tudo de maneira

transparente. Isso é possível, entre outros motivos, devido ao grau de abstração e desacoplamento que o MMV oferece às MVs em relação aos recursos físicos dos hospedeiros.

Assim sendo, a MV e o seu "estado" são movimentados para o novo sistema hospedeiro através da transferência das páginas de memória virtual via rede, sem perda de contexto e sem interrupção das aplicações. Este processo é chamado de migração "ao vivo" ou *live migration*, mas também é possível fazer a migração da MV com uma pequena interrupção dos serviços (*cold migration*). Os mesmos descritores de arquivo podem ser reutilizados, pois o sistema de armazenamento é configurado para ser compartilhado pelos hospedeiros envolvidos no processo (através de NFS ou de um *storage*, por exemplo). As conexões de rede são assumidas pela placa de rede do hospedeiro de destino, através da transferência do endereço físico da MV (MAC *address*) para a nova placa.

Na próxima seção, será apresentado o XEN, solução de código aberto adotada como solução de virtualização no ambiente de testes implementado para este trabalho, conforme Capítulo 4.

2.2 O virtualizador XEN

XEN [15] é uma solução de virtualização de código aberto desenvolvida como parte do projeto XenoServers na Universidade de Cambridge. Sua primeira versão surgiu em 2003. A criação do XEN levou ao surgimento da empresa XenSource que foi adquirida pela Citrix em 2007.

XEN utiliza o modelo de paravirtualização sendo que o MMV faz uso de um sistema Linux com núcleo modificado e o conceito de domínios tanto para o sistema hospedeiro (Dom0) quanto para os sistemas hóspedes (DomU), conforme mostrado na Figura 2. Como mencionado anteriormente, o uso de um sistema operacional modificado torna a paravirtualização uma técnica mais eficiente em comparação com a virtualização total, mas em contrapartida pode impedir o seu uso em sistemas que não permitem alteração do núcleo, caso não haja suporte no *hardware* para virtualização.

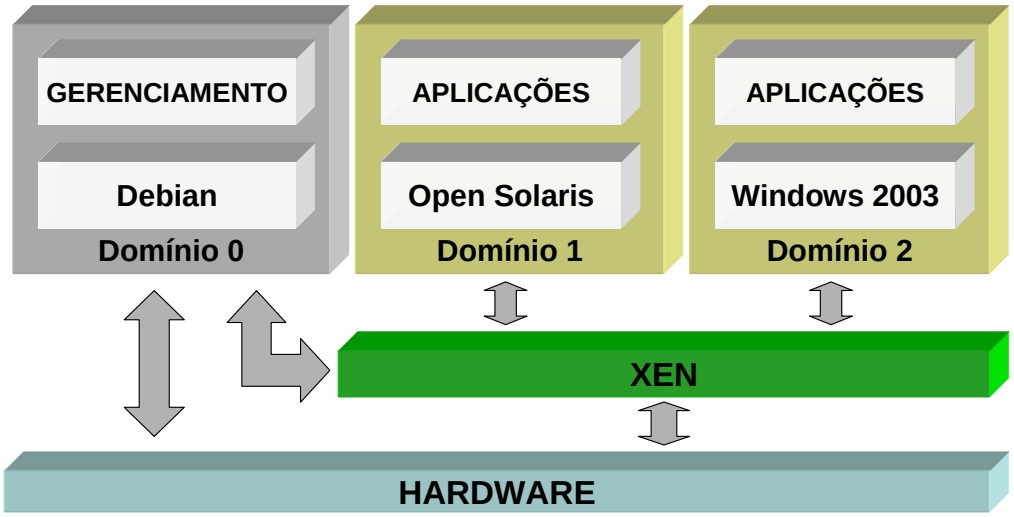


Figura 2 – Modelo com paravirtualização (XEN)

A instalação do XEN consiste na colocação de uma nova versão do núcleo do sistema Linux hospedeiro, a qual deve ser compilada com suporte ao virtualizador. Esse processo de compilação de núcleo pode trazer problemas de incompatibilidade com dispositivos (falta de *drivers*) ou problemas com o tamanho da imagem gerada devido ao excesso de opções de suporte que foram incluídas no mesmo ao invés de se utilizar módulos para prover o suporte opcional a certos dispositivos. Além do núcleo, deve ser instalado o *daemon xend* que é o responsável pelas funções de manipulação e gerência entre MMV e as MVs. Existem pacotes de instalação prontos para criar o ambiente com XEN, como, por exemplo, pacotes RPM para distribuições RedHat Enterprise Linux e Fedora ou pacotes DEB para distribuições Debian e Ubuntu, o que pode tornar um pouco mais fácil a instalação do virtualizador. Na implementação do ambiente utilizado neste trabalho, após diversas dificuldades com o processo de compilação de núcleo com suporte ao XEN na distribuição Fedora, optou-se por utilizar os pacotes DEB da distribuição Ubuntu 8.04 que suporta o XEN na versão 3.2.

Com relação à gerência do XEN, junto com a solução é disponibilizada a ferramenta XM que permite, via linha de comando, a comunicação com o MMV para criar, configurar e manipular MVs. Por se tratar de uma solução de código aberto, existem diversas ferramentas de gerência que também podem ser instaladas e utilizadas para permitir o controle do ambiente baseado no XEN, entre as quais podemos citar a Virt-Manager [16]. Esta ferramenta disponibiliza uma interface gráfica que permite a gerência centralizada dos hospedeiros e MVs, porém essa gerência é feita sobre cada hospedeiro e não sobre um conjunto de hospedeiros, ao contrário da proposta apresentada neste trabalho, que visa à gerência do ambiente como um todo. Apesar de nosso trabalho não apresentar uma ferramenta gráfica

para uso, como foi desenvolvida uma API para dar suporte à arquitetura proposta, nada impede que no futuro sejam desenvolvidas ferramentas que facilmente se integrem com a API criada.

O XEN possui uma API chamada Xen API que permite maior flexibilidade na manipulação e criação de MVs se comparada com a ferramenta XM, podendo ser utilizada em conjunto com aplicações externas. A Xen API faz uso de chamadas XML-RPC e existem bibliotecas que permitem sua utilização através das linguagens PERL, Python, C e Java. Na Seção 3.3 será discutido o uso da Xen API na implementação da API proposta neste trabalho.

3 ARQUITETURA PROPOSTA

Este capítulo apresenta a arquitetura proposta neste trabalho descrevendo cada elemento que a compõe e mostrando como eles se relacionam. Ao descrever os elementos e a própria arquitetura são discutidas as questões que levaram à concepção do modelo desenvolvido. Logo após é apresentada a API criada para a gerência dos hospedeiros e MVs, bem como os aspectos que foram considerados no seu desenvolvimento.

3.1 Apresentação da arquitetura

Na Figura 3 é apresentado o diagrama da arquitetura.

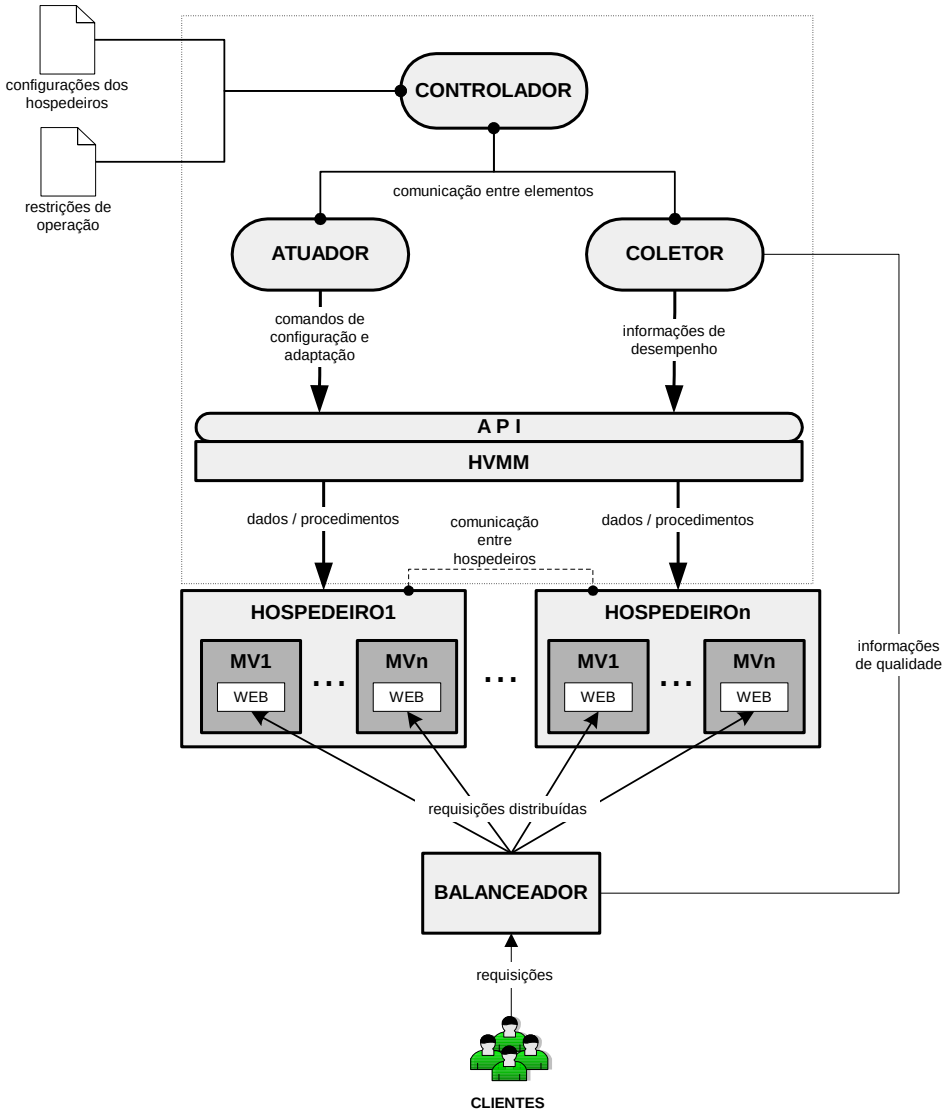


Figura 3 – Diagrama da arquitetura proposta

A arquitetura é composta por elementos, concebidos na forma de módulos com atividades bem definidas. A seguir os mesmos serão descritos.

- **Controlador.** Trata-se do elemento central da arquitetura. Inicialmente ele é carregado com as informações específicas dos hospedeiros (nome, endereço MAC, quantidade de processadores, frequência, etc.) e do Balanceador (nome, URL, etc.), além das restrições de operação da aplicação Web (tempo de resposta máximo admitido). Durante sua execução verifica periodicamente o cumprimento das restrições de operação, com base nas informações obtidas do Coletor, e aciona o Atuador para reconfigurar o ambiente adequadamente se necessário. Também determina parâmetros de operação do Coletor, tais como a quantidade de medições realizadas pelo mesmo. O Controlador é o responsável por estabelecer a conexão inicial com cada hospedeiro via API e mantém estas conexões para serem utilizadas pelo Atuador e Coletor posteriormente;
- **Atuador.** Elemento responsável por efetivamente atuar nos hospedeiros. Utiliza a API para realizar operações para ativar ou desativar hospedeiros, aumentar ou diminuir a frequência da CPU e ligar ou desligar processadores (ou núcleos). O atuador também é responsável por migrar MVs quando solicitado. Para isso, utiliza uma função que seleciona a MV a ser migrada e outra que escolhe o melhor hospedeiro de destino segundo critérios, que serão mostrados com mais detalhes na Seção 3.4.1;
- **Coletor.** Este elemento, que também utiliza a API, obtém as informações dinâmicas dos hospedeiros (utilização de CPU, frequência atual, lista e quantidade de MVs hospedadas, etc.) e do Balanceador (atual tempo de resposta da aplicação Web). O Controlador aciona o Coletor periodicamente, em ciclos de medição. Consideraremos um ciclo de medição, o período entre o início da coleta de informações dos hospedeiros e a última medição do tempo de resposta da aplicação Web, quando então o Controlador pode tomar uma decisão. Especialmente quanto ao tempo de resposta e à utilização de CPU, o Coletor faz durante um ciclo diversas medições e aplica filtros sobre os valores obtidos, a fim de evitar que oscilações temporárias (picos) provoquem reconfigurações desnecessárias do ambiente (mais informações na Seção 3.4.1);
- **API.** Fornece uma camada de alto nível para que os elementos Controlador, Atuador e Coletor possam atuar sobre os hospedeiros e MVs tanto para coleta de

informações (dados) quanto para reconfigurações (procedimentos). A próxima seção irá abordar com mais detalhes este elemento;

- **Clientes.** Representa o grupo de usuários finais do serviço provido, no caso, uma aplicação hospedada em servidores Web;
- **Balanceador.** Este elemento é o responsável por distribuir as requisições dos Clientes para os servidores Web, provendo desempenho e escalabilidade;
- **Máquinas Virtuais.** Elementos que residem nos hospedeiros. Cada MV suporta um servidor Web que recebe requisições distribuídas pelo Balanceador;
- **Hospedeiros.** Elementos responsáveis por hospedar as MVs, fornecendo de maneira compartilhada acesso aos recursos físicos (CPU, memória, disco e rede). Um desses elementos no conjunto será chamado de hospedeiro principal, no qual ficarão armazenadas as imagens de disco das MVs compartilhadas via NFS. O hospedeiro principal será o único hospedeiro que nunca será desligado para poder suportar todas as MVs do ambiente em situações de baixa utilização de recursos.

Cabe ressaltar que o Controlador, Atuador, Coletor e a API são propostos e desenvolvidos neste trabalho, enquanto que os demais são elementos que já existem prontos, sendo inclusive utilizados e apresentados em outros trabalhos e, portanto, não precisam ser descritos com muitos detalhes.

3.2 Funcionamento da arquitetura

Na Figura 4 é mostrado o fluxo de funcionamento da arquitetura, através de um diagrama de estados.

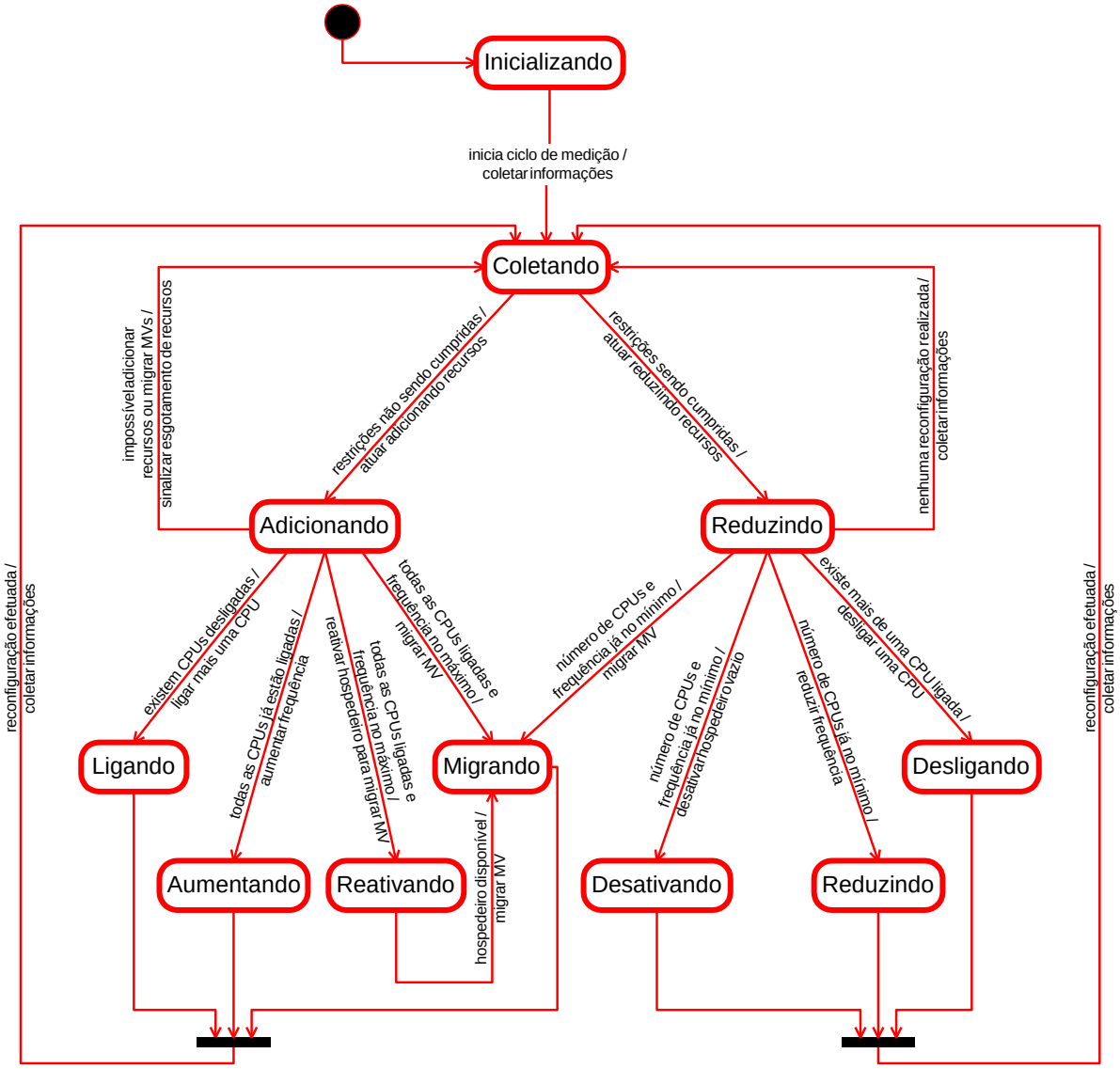


Figura 4 – Diagrama de funcionamento da arquitetura

Ao iniciar, o Controlador carrega dois arquivos XML sendo que um contém informações sobre os hospedeiros e outro as informações sobre o Balanceador, entre elas o tempo de resposta máximo admitido para a aplicação instalada nos servidores Web. O Apêndice A mostra os arquivos XML utilizados na implementação. Em seguida, usando a API, o Controlador abre as conexões com os hospedeiros gerenciados e ativa o Coletor, que inicia um ciclo de medição. Após o término do ciclo de medição, o Controlador obtém informações atualizadas sobre os hospedeiros e sobre a aplicação Web, e avalia se o tempo de resposta desta está acima ou abaixo do máximo admitido. Se a avaliação indicar que o tempo está acima, o Controlador aciona o Atuador para que este reconfigure os hospedeiros do conjunto, adicionando recursos aos mesmos ou migrando MVs entre eles. Caso o tempo de

resposta esteja abaixo, o Controlador aciona o Atuador para reconfigurar o ambiente só que, desta vez, com a finalidade de economizar energia, retirando recursos ou consolidando MVs, desde que a qualidade de serviço oferecida não seja afetada.

Quando o Atuador é acionado para adicionar recursos, a seguinte política é adotada:

1. Verificar se a utilização de CPU de cada hospedeiro está acima do máximo definido para cada um deles;
2. Para os hospedeiros que tiverem a utilização de CPU acima do máximo, o Atuador tenta aumentar a frequência de operação das CPUs, caso as mesmas ainda não estejam no máximo. Se em pelo menos um dos hospedeiros a frequência de CPU for aumentada, o Atuador retorna o controle para o Controlador;
3. Se na etapa anterior nenhum hospedeiro teve a sua frequência aumentada, o Atuador tenta, então, ligar mais núcleos nos hospedeiros que tiverem utilização alta de CPU, caso ainda existam núcleos que possam ser ligados (ativados). Caso a operação de ligar núcleos ocorra em pelo menos um dos hospedeiros, o Atuador retorna o controle para o Controlador;
4. Se nas etapas anteriores a reconfiguração de frequência e núcleos não foi executada em nenhum dos hospedeiros, o Atuador seleciona o primeiro hospedeiro com uso de CPU acima do máximo e busca outro com recursos disponíveis para fazer a migração de uma MV. Após efetuar a migração, o Atuador retorna para o Controlador;
    4.1. Se não existirem hospedeiros com recursos disponíveis para receber a MV a ser migrada no caso anterior, o Atuador, ainda, ativa hospedeiros que estiverem em estado de espera para permitir a migração. Após efetuar a migração, o Atuador retorna para o Controlador;
5. Por fim, se todos os hospedeiros estiverem ativos, em suas configurações máximas (frequência e núcleos) e sem recursos disponíveis, o que impede a migração de MVs dentro do conjunto, o Atuador sinaliza que não é possível reconfigurar o ambiente e que os recursos estão esgotados.

No caso do Atuador ser acionado para retirar recursos, um processo similar ao anteriormente descrito acontece:

1. Verificar se a utilização de CPU de cada hospedeiro está abaixo do mínimo definido para cada um deles;

2. Para os hospedeiros que tiverem a utilização de CPU abaixo do mínimo, o Atuador tenta reduzir, se possível, a frequência de operação das CPUs. Se pelo menos um dos hospedeiros teve sua frequência reduzida, o Atuador retorna o controle para o Controlador;
3. Se na etapa anterior nenhum hospedeiro teve a sua frequência reduzida, pois as mesmas já estavam no mínimo, o Atuador tenta, então, desligar núcleos nos hospedeiros que tiverem utilização baixa de CPU, caso essa operação seja possível. Se a operação de desligar núcleos ocorrer em pelo menos um dos hospedeiros, o Atuador retorna o controle para o Controlador;
4. Se nas etapas anteriores a reconfiguração de frequência e núcleos não foi executada em nenhum dos hospedeiros, o Atuador seleciona o primeiro hospedeiro com uso de CPU abaixo do mínimo e busca outro com recursos disponíveis para fazer a migração de uma MV e efetua a migração, se possível;
   4.1. Se não existirem hospedeiros com recursos disponíveis para receber a MV a ser migrada no caso anterior, nenhuma operação é realizada e o Atuador retorna para o Controlador, apenas;
5. Por fim, os hospedeiros que estiverem em suas configurações mínimas (frequência e núcleos) e sem nenhuma MV hospedada serão colocados em estado de espera (desativados), exceto o hospedeiro principal.

Ao realizar uma das etapas de reconfiguração anteriormente descritas, o Atuador devolve o controle do sistema para o Controlador. Este elemento aciona o Coletor para obter informações atualizadas para com base nessas determinar se novas reconfigurações serão necessárias ou não. A estratégia é evitar múltiplas reconfigurações de uma única vez, o que pode implicar em desperdício de recursos ou prejudicar a estabilidade do sistema como um todo. A ordem empregada na política de atuação nos hospedeiros foi escolhida em função dos testes iniciais realizados e das conclusões obtidas, conforme detalhes na Seção 4.3.2.

Segundo [17] os processadores normalmente são os responsáveis pela maior parte da energia consumida num sistema computacional. Assim sendo, neste trabalho optou-se por atuar somente na CPU dos hospedeiros e deixar o virtualizador distribuir o uso desse recurso pelas MVs, deixando fixa a configuração da CPU virtual das MVs (quantidade de VCPUs). Quanto à gerência de memória tanto do hospedeiro quanto das máquinas virtuais, também optamos por deixar que o virtualizador fizesse o gerenciamento e procuramos garantir a quantidade mínima de memória RAM para acomodar todas as MVs nos hospedeiros.

É importante destacar que a arquitetura foi concebida para ser independente de plataforma de virtualização ou sistemas operacionais específicos. A API desenvolvida foi modelada tendo em mente a transparência, viabilizando a independência citada e também dando flexibilidade à arquitetura proposta. Esta estratégia permite o uso de outras formas de implementação o que torna a arquitetura reutilizável.

Como a arquitetura é modular e cada elemento desempenha um papel específico, o modelo apresentado é flexível e consequentemente reutilizável. Se um único módulo fosse responsável por todos os papéis ou funções, seria descartada a possibilidade de reuso de elementos que não precisassem ser modificados. Como exemplo, a arquitetura pode ser adaptada para ser aplicada em um sistema computacional não-virtualizado, pois a princípio, o único elemento que deve ter funções suprimidas é o Atuador que, no caso, deixará de executar a função de migração de MVs, não aplicável a sistemas dessa natureza.

A arquitetura foi desenvolvida inicialmente para gerenciar uma única aplicação Web, pois múltiplas aplicações implicariam em utilizar políticas de controle distintas, considerando que os níveis de qualidade de serviço não são necessariamente iguais. A análise de diversas políticas para permitir o gerenciamento eficiente dos recursos poderia aumentar a complexidade do sistema e, assim, neste momento, optou-se por controlar uma única aplicação, ficando a outra abordagem para trabalhos futuros.

Se frequentemente ocorrem situações de esgotamento de recursos, conforme política anteriormente descrita, o que indica que as restrições de operação não podem ser cumpridas, isto é um indício para o administrador do ambiente que existe uma tendência de aumento de carga geral. Neste caso, as medidas para o planejamento de capacidade devem ser refeitas (da mesma forma que foi realizado inicialmente neste trabalho) já que novos recursos físicos devem ser adicionados.

3.3 A API *Host and Virtual-Machine Manager* (HVMM)

Um elemento importante no modelo apresentado é a API desenvolvida neste trabalho, batizada de HVMM, que dá suporte às operações realizadas pelos demais elementos. Através desta é possível atuar sobre os hospedeiros e MVs fazendo-se uso de funções em alto nível, tais como, migração, desativação e mudança de frequência, sem a preocupação com os aspectos de implementação pertinentes à execução desses procedimentos (baixo nível). A seguir serão discutidas a solução adotada bem como as decisões de implementação que foram tomadas.

Embora os virtualizadores disponíveis possuam suas próprias APIs, em muitos casos estas possuem características dependentes do próprio virtualizador. Um exemplo disto é a chamada de migração de MVs da API do virtualizador VMware [18] que conta com um parâmetro relacionado ao *resource pool* envolvido na operação. Um *resource pool* é um conjunto de recursos físicos do ambiente (um subconjunto de hospedeiros, por exemplo) e é uma característica presente na plataforma VMware sendo que no virtualizador XEN esse conceito não se aplica desta forma.

Existe também uma solução de API independente de plataforma chamada LIBVIRT [19] que atualmente dá suporte a diversos virtualizadores, através de uma camada abstrata para gerência de hospedeiros, MVs e outros recursos do ambiente. LIBVIRT é uma proposta de código aberto e faz uso de *drivers* específicos para cada virtualizador o que permite que suas funções sejam implementadas na solução desejada. Ao contrário do exposto anteriormente, funcionalidades dependentes do virtualizador não são visíveis na LIBVIRT, o que é uma vantagem considerando-se os aspectos de transparência e reutilização da API, mas é uma desvantagem, pois não permite dar um tratamento personalizado às funcionalidades específicas quando do desenvolvimento de sistemas de gerência. Além disso, o uso de um *driver* associado ao virtualizador pode limitar o uso desta API em futuras soluções, lembrando que nem sempre as plataformas de virtualização possuem código aberto. LIBVIRT é o que se mostra mais próximo de um padrão para APIs de gerência de ambientes de virtualização, sendo aderente inclusive ao padrão *Common Information Model* (CIM) da DMTF [20]. Entretanto, algumas funcionalidades indispensáveis para a arquitetura proposta neste trabalho não existem na LIBVIRT. Um dos exemplos é a função *hvmsetcpuson()*, desenvolvida na HVMM para determinar a quantidade de núcleos que devem ser ligados em um hospedeiro.

Como o XEN foi o virtualizador adotado na implementação deste trabalho, foi feito um estudo da API fornecida com esta solução, a Xen API, analisando-se os mecanismos de monitoramento, comunicação e atuação com os hospedeiros e MVs. Durante o estudo e testes percebeu-se que existiam algumas funções não implementadas, embora declaradas na documentação, assim como funções que necessitavam de aperfeiçoamentos ou correções. Por exemplo, a função *get_speed()*, responsável por retornar a frequência da CPU dos hospedeiros, não informava o valor atual após uma mudança de frequência. Da mesma forma que na LIBVIRT, algumas funções importantes para o desenvolvimento deste trabalho não estavam presentes na Xen API. Citando mais um exemplo neste caso, a função *hvmsuspend()*, também desenvolvida na HVMM, que coloca um hospedeiro em estado de espera (*suspend-to-RAM*).

Em resumo, existem problemas de dependência de plataforma ao adotar-se pura e simplesmente a API nativa do virtualizador e dependência de *drivers* da LIBVIRT para futuras soluções, além de, em todos os casos, haver necessidade de implementação ou correção de funções. Tendo em vista essas questões, optamos por propor uma nova API, independente pelo menos na camada de alto nível, que atendesse aos requisitos necessários.

As chamadas da HVMM foram modeladas da maneira mais genérica possível para permitir a portabilidade e reutilização da API. Por exemplo, a chamada *hvmmigratevm()* utiliza como parâmetros apenas o nome da MV a ser migrada e os objetos dos hospedeiros envolvidos (origem e destino), ou seja, a chamada independe do virtualizador usado na implementação, já que esses são os 3 elementos mínimos para uma operação de migração de MV em qualquer solução deste tipo. Os detalhes e as eventuais variações nas chamadas nativas dos virtualizadores usados na implementação podem ser tratados na implementação da biblioteca para a HVMM para a respectiva plataforma. Mais detalhes sobre a implementação serão apresentados na próxima seção.

3.4 **Detalhes de implementação**

3.4.1 Implementação da arquitetura

Os elementos da arquitetura apresentados na Seção 3.1 foram implementados como um sistema de gerência, executado no ambiente composto pelos hospedeiros, MVs e servidores Web, onde o objetivo é manter a qualidade do serviço (expressado pelo tempo de resposta de uma aplicação Web) procurando reduzir o consumo de energia. Como a API proposta neste trabalho foi implementada em forma de uma biblioteca em PERL, o sistema de gerência também foi implementado usando-se essa linguagem para facilitar a integração do código e pelas mesmas razões que levaram à escolha da PERL na implementação da API.

O Controlador é o módulo principal do sistema de gerência e o Atuador e o Coletor são módulos adicionais. Na comunicação entre os módulos é utilizado um objeto que contém as informações dos hospedeiros. Esse objeto também é passado nas chamadas da API proposta, conforme mais detalhes na próxima seção. O sistema de gerência é executado em um servidor Linux dedicado a essa função.

Como mencionado anteriormente, a cada ciclo de medição o Coletor realiza diversas medições de tempo de resposta e uso de CPU e aplica filtros sobre os valores obtidos. Isso se torna necessário, pois o ambiente pode sofrer variações temporárias que não representam a

tendência real do seu comportamento, o que causaria reconfigurações desnecessárias. Entre cada medição individual aguarda-se um intervalo de *p* segundos, pois diversas medições seguidas estariam sujeitas ao mesmo problema.

Para reduzir o efeito dos valores transientes, utilizou-se como filtro a média móvel exponencial [21]. As médias móveis tem por objetivo suavizar as variações de valores de uma série ao longo do tempo, gerando-se uma tendência. Nas médias móveis exponenciais, é aplicado um coeficiente α ($0 < \alpha \leq 1$) no valor atualmente medido (V_t) e um coeficiente ($1 - \alpha$) na média anteriormente calculada (M_{t-1}), de forma a obter uma nova média (M_t) no instante atual (*t*), conforme Equação 1.

$$M_t = \alpha \times V_t + (1 - \alpha) \times M_{t-1} \quad (1)$$

Valores de α próximos a 0 diminuem a influência dos novos valores obtidos sobre a média enquanto que valores próximos a 1 dão menor peso aos valores anteriormente medidos, ou seja, o coeficiente α determina a velocidade de reação do filtro face às medições efetuadas. Este tipo de filtro também foi utilizado no trabalho proposto por [22].

Para a implementação foi escolhido o valor de 0,5 para o coeficiente α. Para o valor de intervalo entre as medições (*p*) foi escolhido o tempo de 10 segundos, pois se espera que variações pontuais ocorridas nesse período não afetem sensivelmente o desempenho do ambiente. Para o ciclo de medição do Coletor foi escolhido o valor de 100 segundos, ou seja, são realizadas 10 medições com intervalo de 10 segundos entre as mesmas. O período de 100 segundos foi determinado em função de testes de ajuste para avaliar o tempo de reação do sistema para realizar uma reconfiguração.

A ordem de reconfiguração de recursos aplicada pelo Atuador foi mostrada na Seção 3.2. A heurística de escolha no processo de migração de MVs merece aqui um destaque especial na implementação do Atuador, pois afeta diretamente o comportamento do ambiente, seja na qualidade de serviço, seja na economia de energia. No caso do Atuador migrar MVs para distribuir carga, fará uso da função *hvmselectvm()* da API proposta que escolherá qual MV deverá ser retirada de um hospedeiro sobrecarregado. A política adotada na implementação desta função foi a de escolher a primeira MV da lista do hospedeiro, pois como a distribuição das requisições dos clientes é igual para todos os servidores Web, basta retirar a primeira MV para reduzir o uso de recursos do hospedeiro. Nada impede, entretanto, que outras políticas possam ser adotadas em trabalhos futuros, como, por exemplo, a MV que utiliza mais CPU.

Na escolha do hospedeiro de destino para a migração da MV, o Atuador faz uso da função *hvmselecthost()* da API. Esta função, quando utilizada para reduzir o consumo de

recursos, irá distribuir (balancear) a quantidade de MVs em todos os hospedeiros, escolhendo aquele que tiver menor pontuação (P), conforme a Equação 2, sendo M_a a média de utilização de CPU atual e M_b a anterior (valores obtidos diretamente do objeto que armazena as informações do hospedeiro).

$$P = M_a \times 50 + (M_a - M_b) \times 50 \quad (2)$$

Desta forma, é atribuído um peso 50 para a média de uso atual e um peso 50 para a tendência da utilização (diferença entre as médias de utilização de CPU atual e anterior). Essa forma de cálculo evita a escolha de um hospedeiro que esteja com crescimento de utilização de CPU (o que pode ser uma escolha ruim) e ao mesmo tempo privilegia a escolha de um hospedeiro que, apesar de estar com utilização alta de CPU, esteja com tendência a reduzir esse uso. O valor 50 para os pesos apresentou bons resultados nos testes realizados. Entretanto, em trabalhos futuros poderá ser estudado o efeito da mudança desses pesos no desempenho e no consumo de energia do ambiente.

Quando a função *hvmselecthost()* é utilizada pelo Atuador na escolha de hospedeiros para reduzir o consumo de energia, o mesmo esquema de pontuação é utilizado, porém não é levado em consideração o balanceamento de MVs já que o objetivo é reduzir a quantidade de hospedeiros e não distribuir carga.

Quanto ao elemento Balanceador, o mesmo foi implementando utilizando o módulo *mod_proxy_balancer* do servidor Web Apache (vide Seção 4.1.4) em um servidor Linux. O módulo foi configurado para distribuir igualmente as requisições HTTP dos clientes para os servidores Web em execução nas MVs. Uma das premissas do Balanceador foi não usar políticas de balanceamento específicas do *mod_proxy_balancer* a fim de permitir o uso de qualquer solução similar que distribua requisições da mesma forma, como, por exemplo, o servidor Web NGIX [23] ou um *switch* de camada 7.

Os elementos hospedeiros são servidores Linux que utilizam o virtualizador XEN e as MVs são servidores Linux comuns configuradas para executar cada uma o servidor Web Apache. Mais detalhes sobre a implementação e configuração destes e dos demais elementos podem ser vistos no Capítulo 4.

3.4.2 Implementação da API HVMM

Optou-se por utilizar PERL na implementação da API HVMM, tendo em vista a facilidade e a robustez que essa linguagem apresentou durante o estudo e testes com a Xen API, além da disponibilidade de *scripts* prontos exemplificando o uso. Também existe suporte

para implementações com a Xen API em Java, Python e C. Desta forma, a API proposta pode ser utilizada por outras linguagens, desde que seja implementada uma biblioteca correspondente à linguagem desejada.

Na HVMM é feito o uso de um objeto que contém em seus atributos as diversas informações dos hospedeiros (nome, endereço MAC, uso de CPU, lista de MVs, frequências disponíveis, etc) sendo este objeto utilizado em todas as chamadas mantendo-se consistente e persistente o estado dos hospedeiros. No Apêndice B é mostrada a descrição dos atributos do objeto usado na HVMM e o Apêndice C mostra a relação de todas as funções.

Para complementar o suporte necessário à implementação da HVMM, foram feitas alterações na Xen API incluindo-se as funções necessárias para a arquitetura proposta e corrigindo-se algumas funções existentes, para permitir a execução de operações importantes e também como forma de contribuição ao virtualizador XEN (o Apêndice D destaca todas as alterações efetuadas no código da Xen API). As chamadas da Xen API foram encapsuladas dentro das chamadas da HVMM, escondendo-se os detalhes inerentes à plataforma XEN o que permite portar a API proposta para outros virtualizadores, preservando-se as premissas da interface. Quanto à comunicação com os elementos gerenciados via HVMM, aproveitou-se o mecanismo da Xen API baseado em XML-RPC.

A Figura 5 mostra como exemplo a função *hvmsetspeed()* implementada na HVMM.

```
1  sub hvmsetspeed
2  {
3    my ($_hvm) = $_[0];         # parâmetro hospedeiro
4    my $_new_speed = $_[1];     # parâmetro novo clock
5    my $_r = "";                # retorno da chamada
6
7    $_new_speed *= 1000;        # ajusta valor de MHz para Hz
8    $_r = xvalue($_hvm->{conn}->simple_request
9    ("host_cpu.set_speed",$_hvm->{session},$_new_speed));
10
11   if ($_r eq "") {
12     return -1
13   }
14   else {
15     return 0
16   }
17 }  # hvmsetspeed
```

Figura 5 – Exemplo de uso da função da HVMM

A função mostrada no exemplo serve para alterar a frequência de operação da CPU de um hospedeiro. A linha 4 mostra que o objeto que contém informações sobre o hospedeiro é recebido como um parâmetro dentro da função e a linha 5 indica o novo valor de frequência a

ser configurado. Os atributos *conn* e *session* do objeto *$_hvm* foram previamente preenchidos com o uso das funções *hvmconnect()* e *hvmsession()* da HVMM. Nas linhas 9 e 10 é mostrada a chamada da Xen API para alterar a frequência. Entre as linhas 12 e 17 é feito o tratamento do retorno da chamada da Xen API para determinar se o retorno da função *hvmsetspeed()* será bem sucedido (0) ou não (-1). Como observação, destacamos que a função *set_speed* (linha 10) não existia na Xen API original e teve que ser também implementada e adicionada.

Com a finalidade de reduzir o consumo de energia, porém sem prejudicar a qualidade de serviço, ao invés de desligar os hospedeiros ociosos (sem MVs), adotou-se a estratégia de colocá-los em estado de espera. Para isso foi implementada a função *hvmsuspend()* que utiliza chamadas ao sistema operacional para colocar o hospedeiro no estado ACPI S3 (s*uspend-to-RAM*) [24].

Testes realizados em nosso ambiente (conforme Seção 4.3.3) mostraram que essa estratégia é vantajosa, pois em estado de espera o consumo de energia do hospedeiro é baixo e o tempo de reativação é bem menor que o tempo de inicialização (*boot*), caso o hospedeiro fosse desligado. Os trabalhos apresentados por [9] e [10] também mostram essa forma de implementação como uma boa solução. A reativação do hospedeiro é feita através do mecanismo *wakeonlan* [25] através da função *hvmwakeup()*.

3.4.3 Implementação de funções adicionais

Além das funções implementadas na API HVMM, foi desenvolvida a função *getrt()* para obter o tempo de resposta da aplicação Web. Esta função é executada sobre o balanceador que é o elemento responsável por distribuir as requisições dos clientes para os servidores Web.

A função *getrt()* recebe como parâmetro uma URL alvo de onde pode ser obtido o tempo de resposta atual da aplicação Web. Na implementação, a URI que contém essa informação é a "*/resp_time.html*" que fica hospedada no balanceador. Essa URI armazena o tempo médio de acesso em milissegundos o qual é atualizado à medida que novas requisições são recebidas. Essa atualização é possível graças a um programa em C que foi feito com base no programa *rotatelogs* (disponibilizado com o Apache para rotação de *logs* do servidor). Através do redirecionamento do *log* do Apache para esse programa (conforme Seção 4.1.2), os tempos de acesso à aplicação Web são contabilizados com uso de média móvel exponencial cujo valor final é gravado no arquivo *resp_time.html* no diretório-raiz do servidor. A solução aqui apresentada pode permitir no futuro que a função *getrt()* possa

informar o tempo de resposta de diversas aplicações no mesmo servidor Web, além de permitir a implementação com outros servidores Web que tenham suporte para personalização e redirecionamento de seus *logs*.

Outra função implementada e que não faz parte da API foi a *getapcpower()*. Através de uma conexão USB entre o *nobreak* utilizado no ambiente e um dos servidores (Seção 4.2) e fazendo-se uso de um utilitário chamado *apcupsd* [26] são obtidas diversas informações, tais como a tensão e nível de carga da bateria, tempo de autonomia, tensão de entrada, entre outras. A função *getapcpower()* retorna, então, a potência em VA fornecida pelo *nobreak* aos hospedeiros. Desta forma, é possível verificar se com o uso da arquitetura proposta é possível reduzir o consumo de energia do ambiente.

Ambas as funções apresentadas nesta seção foram desenvolvidas em PERL pelas mesas razões expostas nas seções anteriores.

4 AVALIAÇÃO DE DESEMPENHO

Neste capítulo é mostrada uma avaliação de desempenho da arquitetura proposta, através da implementação de um sistema de gerência para o ambiente suportando uma aplicação Web. Também são descritos neste capítulo as ferramentas (*softwares*) usadas no trabalho, o ambiente utilizado, os testes iniciais e os testes de desempenho realizados, assim como a análise dos resultados obtidos. No final, são feitas considerações gerais sobre algumas questões encontradas.

4.1 Softwares utilizados

4.1.1 HTTPERF

HTTPERF [27] é uma ferramenta de código aberto que é usada em testes de avaliação de desempenho de servidores Web. Sua primeira versão foi desenvolvida em 2000 no *Hewlett-Packard Research Laboratories*, recebendo em suas versões posteriores contribuições de estudantes da Universidade de Calgary e de outros.

HTTPERF gera requisições HTTP concorrentes, simulando pedidos de clientes, avaliando os resultados além de medir o tempo de resposta médio das requisições enviadas. A ferramenta pode ser configurada para gerar determinada taxa de requisições enviadas, quantidade de conexões e duração do teste, porém existem configurações avançadas que podem ser utilizadas para simular tráfego real baseado em um *log* de acesso de um servidor Web extraindo-se desse *log* as diversas URLs a serem disparadas contra o servidor alvo da avaliação de desempenho. Outra forma de uso da ferramenta para simular tráfego real é analisar o *log* do servidor Web, contabilizar a quantidade de requisições em dados intervalos de tempo e definir parâmetros na HTTPERF que simulem a mesma carga na mesma proporção. Esta forma será utilizada neste trabalho, conforme metodologia de avaliação adotada na Seção 4.4.2.

Ao término de uma sessão de teste da HTTPERF é exibido um relatório contendo informações sobre as taxas de conexões e requisições, os tempos de resposta, os resultados (*status*) dos pedidos HTTP, além de informações sobre uso de recursos locais do cliente (CPU e rede) e erros. Na Figura 6 há um exemplo de relatório.

Esta ferramenta foi escolhida entre outras similares por ser de simples utilização e que atende à necessidade deste trabalho, além de ser bastante utilizada em avaliações de

desempenho desse gênero. HTTPERF é utilizada em conjunto com outra ferramenta que será mostrada na próxima seção.

```
Total: connections 100 requests 100 replies 100 test-duration 0.991 s

Connection rate: 100.9 conn/s (9.9 ms/conn, <=1 concurrent connections)
Connection time [ms]: min 0.3 avg 0.4 max 0.6 median 0.5 stddev 0.0
Connection time [ms]: connect 0.1
Connection length [replies/conn]: 1.000

Request rate: 100.9 req/s (9.9 ms/req)
Request size [B]: 62.0

Reply rate [replies/s]: min 0.0 avg 0.0 max 0.0 stddev 0.0 (0 samples)
Reply time [ms]: response 0.3 transfer 0.0
Reply size [B]: header 198.0 content 295.0 footer 0.0 (total 493.0)
Reply status: 1xx=0 2xx=0 3xx=100 4xx=0 5xx=0

CPU time [s]: user 0.24 system 0.75 (user 24.2% system 75.5% total
99.7%)
Net I/O: 54.7 KB/s (0.4*10^6 bps)

Errors: total 0 client-timo 0 socket-timo 0 connrefused 0 connreset 0
Errors: fd-unavail 0 addrunavail 0 ftab-full 0 other 0
```

Figura 6 – Exemplo de relatório da HTTPERF

4.1.2 Autobench

Criada por [28], Autobench é uma ferramenta desenvolvida em PERL que utiliza HTTPERF para avaliações de desempenho de servidores Web automatizadas.

Em avaliações de desempenho onde são utilizadas diferentes taxas de requisição HTTP dentro de um mesmo *workload* (carga de trabalho), para cada configuração individual é necessário definir uma linha específica de chamada da HTTPERF, colocando-se essas linhas em sequência num arquivo *shell script*, por exemplo. Entretanto ao se utilizar Autobench basta definir quais são as taxas inicial e final, além da variação e a cada iteração a taxa é automaticamente incrementada, preservando-se os parâmetros gerais tais como servidor alvo e tempo de duração do teste. A cada iteração Autobench obtém os relatórios de resultados da HTTPERF e mostra o seu próprio relatório de maneira mais resumida. A Figura 7 mostra um exemplo de relatório com algumas informações suprimidas para facilitar a apresentação.

dem_req_rate	con_rate	min_rep_rate	avg_rep_rate	max_rep_rate	stddev_resp_time	resp_time	net_io
10	1.2	12	12	12	0.0	31.75	3.2
20	2.2	20	20	20	0.0	31.55	5.8
30	3.2	32	32	32	0.1	31.6	8.4
40	4	40	40	40	0.1	31.8	11.0
50	5	50.4	50.4	50.4	0.1	33.3	13.6
60	6	59.8	59.9	60	0.6	34.35	16.2
70	6.4	62.6	63	63.4	2.0	145.1	17.1
80	6.4	62.2	62.5	62.8	2.5	296.05	17.2

Figura 7 – Exemplo de relatório da Autobench

Estes relatórios podem ser utilizados para geração de gráficos ou para serem manipulados por outros programas. Neste trabalho foi implementado um mecanismo para inserção das informações geradas pelo Autobench na solução de monitoramento CACTI (a ser apresentada na próxima seção), sendo possível acompanhar através de gráficos em tempo real os resultados dos testes de desempenho.

Autobench foi escolhida pela sua capacidade de automatização de testes o que é uma característica interessante para a avaliação de desempenho mostrada neste trabalho. Na versão original a ferramenta gera somente requisições com taxas crescentes. Por ser de código aberto, foi acrescentada uma funcionalidade na Autobench que permite gerar requisições com taxas decrescentes para geração de *workloads* em forma de rampa (com taxas de requisição crescentes e logo depois decrescentes) que é outro formato de avaliação apresentado na Seção 4.4.1.

4.1.3 CACTI

CACTI [29] trata-se de uma solução de monitoramento de recursos que faz uso da ferramenta RRDTOOL [30] para armazenamento dos dados e geração de gráficos. Com CACTI é possível monitorar diversos recursos computacionais tais como CPU, memória, rede e disco. Através da adição de *plugins* é possível monitorar informações estatísticas de acesso a servidores de aplicação, temperatura e potência consumida pelos equipamentos, quantidade de elementos em filas de mensagens ou processos, estatísticas de servidores de banco de dados e informações de uso de equipamentos de rede. É possível, ainda, desenvolver *plugins* específicos de acordo com as necessidades desde que respeitados os padrões para inserção desses elementos dentro da solução.

O acesso ao CACTI tanto para visualização como para configuração é todo feito via Web, pois o mesmo foi desenvolvido em linguagem PHP. CACTI coleta periodicamente em

intervalos de 5 minutos os dados dos nós monitorados, usa a ferramenta RRDTOOL para armazenar as informações coletadas (permitindo a geração de gráficos de utilização) e guarda outras informações em um banco de dados. Neste banco de dados também são mantidas as informações de configuração dos nós além de outras configurações da solução. Na visualização dos gráficos são mostrados períodos históricos por hora, dia, semana, mês e ano, mas também podem ser personalizados intervalos específicos (das 10:00 às 12:00 horas, por exemplo). As informações mostradas nos gráficos podem ser exportadas para arquivo, permitindo-se, assim, o tratamento através de planilhas ou outras aplicações. Na Figura 8 é mostrado um exemplo de gráfico exibindo a variação de carga em VA do *nobreak* utilizado neste trabalho, durante um período aproximado de 30 dias.

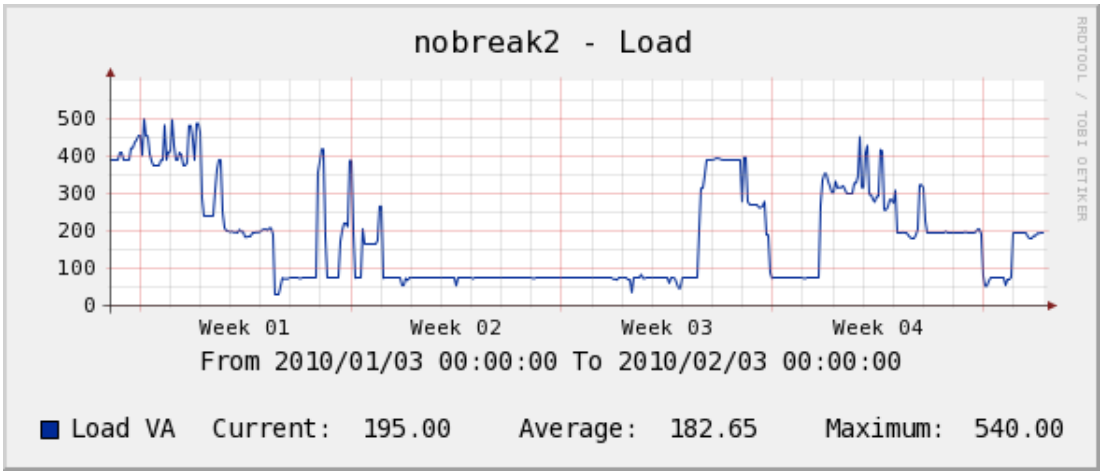


Figura 8 – Exemplo de gráfico gerado por CACTI

A coleta de dados é feita através do protocolo SNMP o que permite a solução obter dados gerais de qualquer equipamento ou aplicação que utilize esse protocolo. Através de SNMP é possível ainda implementar monitores específicos, encapsulando-se dados gerados pela saída de comandos externos ou *scripts* executados no nó monitorado, permitindo, assim, coletar informações fora do padrão SNMP.

CACTI foi escolhida neste trabalho devido à dinâmica que a ferramenta proporciona no acompanhamento dos resultados dos testes, monitorando informações como utilização de CPU dos hospedeiros e energia consumida pelos mesmos. Como estas informações também são coletadas através do próprio sistema de gerência e do monitor de informações de desempenho (Seção 4.1.6), utilizando funções da API HVMM e funções complementares, CACTI foi utilizada para confrontar os valores obtidos.

Foram coletadas informações sobre tempos de resposta, uso de CPU, memória e potência consumida pelos equipamentos (esta última exibida na Figura 8) para observar a utilização destes recursos nos testes antes e depois do uso da arquitetura proposta. Além da configuração padrão da solução, foram inseridos *plugins* para coleta de informações do virtualizador XEN e do *nobreak* APC, além do desenvolvimento de um *plugin* para coleta das informações geradas pela Autobench, permitindo a visualização em tempo de real dos tempos de resposta durante as avaliações de desempenho. A Figura 9 mostra um exemplo de tela do CACTI contendo múltiplos gráficos de desempenho.

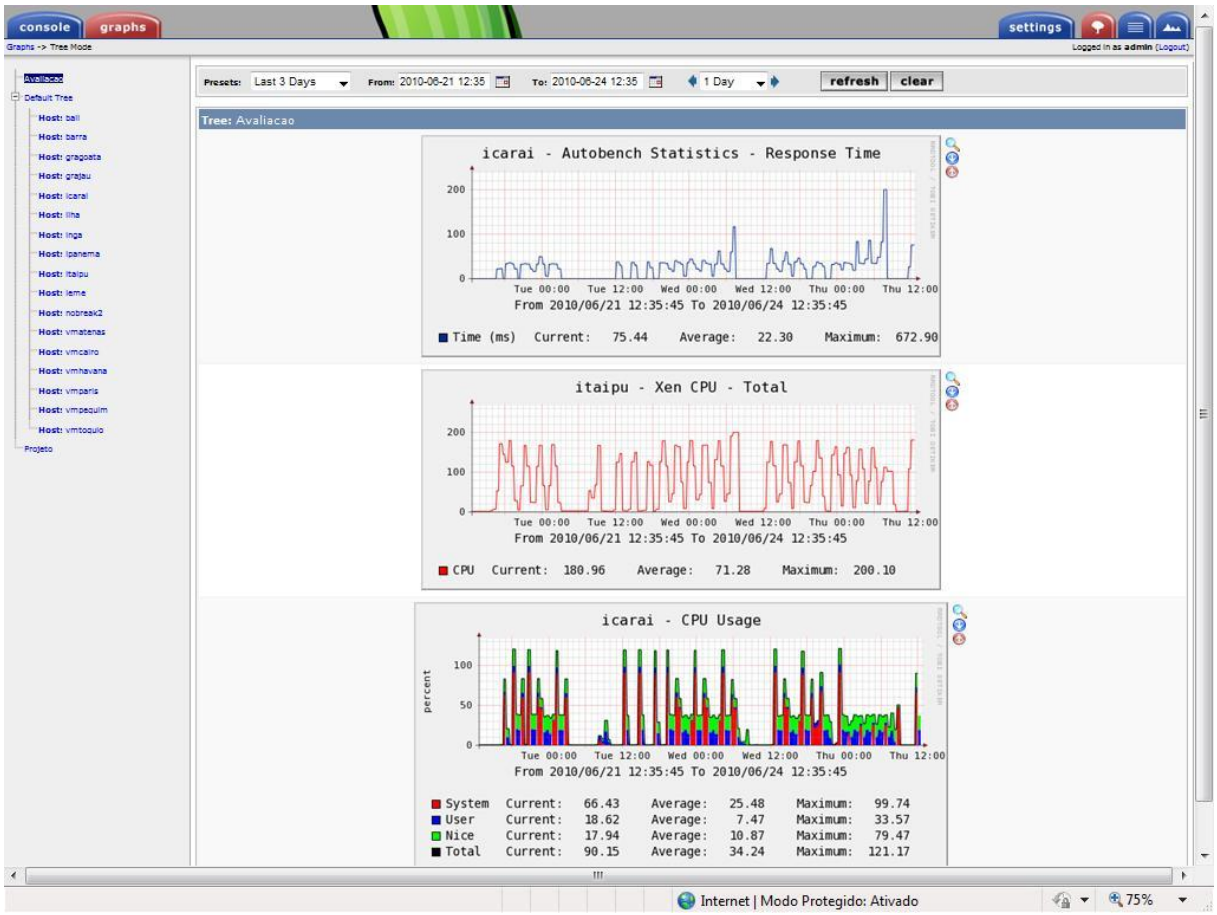


Figura 9 – Tela do CACTI com múltiplos gráficos

4.1.4 Apache

Já mencionado diversas vezes neste trabalho, o servidor Web Apache é um projeto de desenvolvimento de um servidor de código aberto multiplataforma, criado inicialmente por Robert McCool em 1995 e atualmente mantido pela *Apache Software Foundation* [31].

O servidor Web Apache recebe conexões TCP/IP de clientes, normalmente na porta 80/TCP, que representa a porta padrão para o recebimento de requisições HTTP. Através dessas conexões os clientes podem fazer pedidos HTTP enviando e recebendo conteúdo estático em HTML ou dinâmico gerado por aplicações que também geram conteúdo HTML.

Através de módulos que podem ser carregados no servidor, é possível estender as funcionalidades do Apache, permitindo suporte a conexões com bancos de dados, autenticação, fornecimento de conteúdo criptografado (HTTPS), suporte a diversas linguagens no desenvolvimento das aplicações dinâmicas, entre outras diversas funcionalidades. O uso de módulos dá uma razoável flexibilidade ao Apache, pois pode permitir suporte a tecnologias que podem ser desenvolvidas no futuro, sem necessidade de se reescrever o servidor inteiro.

Neste trabalho foi utilizado o módulo *mod_proxy_balancer* que permite que o Apache distribua requisições HTTP para outros servidores Web (*workers*), conforme apresentado na Seção 3.4.1. O *mod_proxy_balancer* possui políticas de balanceamento por quantidade de requisições, quantidade de tráfego ou por quantidade de requisições pendentes. A política por quantidade de requisições foi escolhida por ser a mais simples e que atende às necessidades da avaliação de desempenho. Basicamente, com esta política as requisições recebidas dos clientes são distribuídas de maneira igual para os *workers*. Como foge ao escopo deste trabalho discutir os detalhes das outras políticas de balanceamento, mais informações podem ser obtidas em [10] e em [31].

Através de arquivos contendo diretivas de configuração é possível definir parâmetros de operação do Apache, tais como, a porta TCP para recebimento de conexões, o diretório-raiz para armazenamento das páginas HTML, localização e formato dos arquivos de *log*, entre outros. Para permitir que todos os servidores Apache utilizados na avaliação de desempenho aceitassem uma quantidade de requisições acima da configuração padrão, as diretivas *ServerLimit* e *MaxClients* tiveram seus valores modificados para 10000. Além disso, a diretiva *KeepAlive* teve seu valor modificado para “On” permitindo conexões persistentes (numa única conexão podem ser feitas diversas requisições) aumentando o desempenho dos servidores através da redução da sobretaxa de conexão.

Para que os tempos de resposta da aplicação pudessem ser coletados remota e dinamicamente, foi adicionado na configuração do formato do *log*, o tempo de resposta de cada requisição. Além disso, o *log* foi direcionado para um programa que contabiliza e disponibiliza o tempo médio de resposta da aplicação Web, conforme apresentado na Seção 3.4.3.

O servidor Web Apache foi escolhido neste trabalho por ser capaz de atender os requisitos necessários na avaliação de desempenho e por ser um servidor Web bastante utilizado em trabalhos similares.

4.1.5 Aplicação Web

Citada diversas vezes durante este trabalho, a aplicação Web trata, na verdade, de um pequeno *script* em PHP utilizado para testes de desempenho em serviços de hospedagem de páginas Web, disponibilizado por [32]. O *script* efetua algumas operações matemáticas simples e operações de manipulação de *strings*, de maneira repetitiva o que força a utilização de CPU do servidor Web onde o código é executado.

Caso fosse utilizada apenas uma página HTML estática como alvo das requisições HTTP, a saturação dos recursos físicos ficaria restrita à limitação da rede e do disco (operações de E/S) já que o acesso a conteúdo estático provoca proporcionalmente mais tráfego do que processamento, lembrando que o objetivo é provocar utilização de CPU o que consequentemente provoca um aumento no consumo de energia.

4.1.6 Monitor de informações de desempenho

Nos testes iniciais e nas avaliações de desempenho são medidos o tempo de resposta da aplicação Web e o consumo de energia dos hospedeiros, fatores que são os alvos do estudo aqui apresentado, além da utilização de CPU.

Utilizando-se as funções *hvmgetcpu()* da API HVMM e as funções complementares *getapcpower()* e *getrt()* é possível obter o uso de CPU, o consumo de energia dos hospedeiros e o tempo de resposta da aplicação Web, respectivamente.

Assim sendo, foi desenvolvido em PERL um sistema monitor de informações de desempenho que, utilizando as funções anteriormente citadas, coleta os dados e os registra em um arquivo (*log*), permitindo, assim, a posterior análise dos resultados e a geração de gráficos. O sistema é executado de maneira cíclica durante as sessões de testes e através do registro do horário em que as informações são obtidas, é possível confrontar os valores medidos com a fase correspondente do teste (taxa de requisições corrente, por exemplo). Além de coletar os valores, o sistema calcula as médias e também registra essa informação no *log*.

O sistema foi utilizado nos testes iniciais e nos testes de desempenho (Seções 4.3 e 4.4). Nos testes de desempenho também foram utilizadas algumas informações do *log* do sistema de gerência (operações de reconfiguração realizadas).

4.2 Ambiente utilizado

A topologia do ambiente de testes é apresentada na Figura 10 e o mesmo foi montado no Laboratório de Redes e Sistemas Distribuídos da Faculdade de Engenharia (FEN).

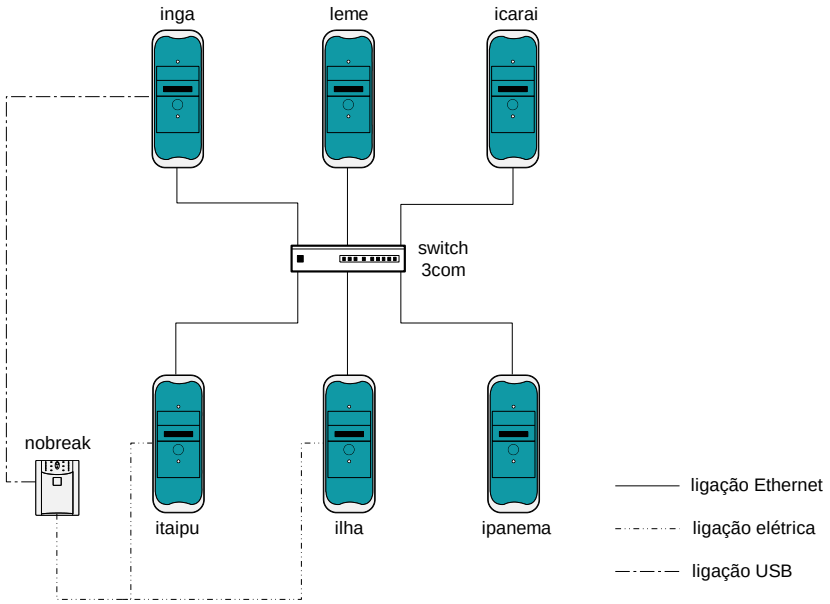


Figura 10 – Topologia do ambiente utilizado

É composto por 6 servidores Linux, um *switch* Gigabit Ethernet da marca 3COM com 8 portas e um *nobreak* da marca APC com capacidade de 1500 VA e fator de potência aproximado de 0.6. O *nobreak* alimenta somente os hospedeiros já que estes elementos são os que serão diretamente reconfigurados provocando a variação no consumo de energia (aumento ou redução, dependendo da reconfiguração realizada), lembrando que a economia de energia é um dos objetivos do trabalho. O nobreak tem sua alimentação elétrica (entrada) mantida durante os testes, mas fornece via interface USB as informações de consumo dos hospedeiros, conforme descrito na Seção 3.4.3.

Na Tabela 1 são mostradas as informações detalhadas sobre a configuração dos servidores utilizados.

Tabela 1 – Detalhamento dos servidores utilizados

Nome	Função	Configuração	Sistema
Inga	gerenciador/ monitor	Intel(R) Pentium(R) 4 2.40 GHz 1 GB de RAM e 40 GB de disco	Linux Ubuntu 8.04 Kernel 2.6.24
Icarai	cliente	Intel(R) Pentium(R) 4 3.20 GHz 2,5 GB de RAM e 80 GB de disco	Linux Ubuntu 8.04 Kernel 2.6.24
Ipanema	balanceador	Intel(R) Pentium(R) D 3.00 GHz 4 GB de RAM e 80 GB de disco	Linux Fedora Core 5 Kernel 2.6.20
Itaipu	hospedeiro principal	Intel(R) Core(TM) 2 Duo 3.00 GHz 4 GB de RAM e 160 GB de disco	Linux Ubuntu 8.04 Kernel 2.6.24-xen
Ilha	hospedeiro	Intel(R) Core(TM) 2 Duo 3.00 GHz 4 GB de RAM e 160 GB de disco	Linux Ubuntu 8.04 Kernel 2.6.24-xen
Leme	monitor (CACTI)	Intel(R) Pentium(R) D 3.00 GHz 2 GB de RAM e 80 GB de disco	Linux Fedora Core 5 Kernel 2.6.20

O ambiente conta ainda com 4 MVs que podem ser alocadas em quaisquer dos 2 hospedeiros. Todas as MVs possuem a mesma configuração (1 CPU virtual, 256 MB de RAM, 2 GB de disco e sistema Linux Fedora Core 9 com *kernel* 2.6.24-xen).

A configuração da quantidade de MVs e a quantidade de CPUs virtuais alocadas foi escolhida após testes, onde se concluiu que o desempenho de duas MVs com uma CPU virtual, cada uma, é melhor do que uma única MV com duas CPUs virtuais alocadas, lembrando que cada hospedeiro possui um processador com dois núcleos.

A quantidade de memória alocada para cada MV também foi determinada com base nos testes efetuados onde se observou o uso médio de memória de cada MV, sendo definida a quantidade mínima suficiente para garantir o seu funcionamento. Além disso, a velocidade de migração de MVs é influenciada pela quantidade de memória alocada para as mesmas, já que o contexto e as páginas de memória devem ser transferidos através da rede de um hospedeiro para o outro.

O gerenciador Inga executa o sistema de gerência e o monitor de informações de desempenho. O cliente Icarai simula solicitações de clientes Web utilizando o gerador de carga HTTPERF e a ferramenta Autobench, criando, assim, requisições HTTP com diversas taxas. O balanceador Ipanema, por sua vez, é o alvo das requisições geradas e executa o servidor Web Apache com o módulo *mod_proxy_balancer* para distribuí-las entre os servidores Apache em execução nas MVs. Os hospedeiros Itaipu e Ilha executam o virtualizador XEN para hospedar as MVs. O monitor Leme executa a ferramenta CACTI para coletar em tempo real as informações de desempenho de todos os equipamentos. O *nobreak*

APC é conectado via USB ao servidor Ingá, já que este executa as funções de gerenciador e monitor.

O objetivo deste ambiente é simular o funcionamento de um conjunto de servidores Web sendo submetidos a requisições HTTP geradas por clientes e distribuídas por um balanceador. Deverão ser medidos tanto o desempenho quanto o consumo de energia no conjunto para avaliação de resultados e validação da arquitetura proposta.

4.3 Testes iniciais

Nesta seção são apresentados alguns testes iniciais que foram realizados para obtenção de informações importantes para a implementação da arquitetura (Seção 3.4) e para a avaliação de desempenho (Seção 4.4). Este conjunto de informações basicamente determinou (i) a estratégia de reconfiguração dos hospedeiros e (ii) a quantidade máxima de requisições suportadas pelo conjunto. Além disso, foram feitos pequenos testes para avaliar os procedimentos de *suspend-to-RAM* e migração de MVs do ponto de vista de eficiência em relação ao tempo e consumo de energia.

Em todos os testes foi usado o mesmo ambiente descrito na seção anterior, composto por MVs nos hospedeiros, cada uma contendo um servidor Web e estes submetidos a diversos *workloads* de requisições HTTP geradas pelo cliente e distribuídas pelo balanceador.

Nas medições do tempo de resposta, uso de CPU e consumo de energia foi utilizado o monitor de informações de desempenho apresentado anteriormente. Durante o funcionamento do monitor são efetuadas 20 medições com intervalo de 15 segundos entre as mesmas (formando períodos de 300 segundos). Isso visa facilitar a análise dos resultados, pois as taxas de requisição HTTP serão enviadas em “degraus” de 5 minutos durante os *workloads*.

Cada *workload* foi executado 5 vezes para permitir o cálculo da média e do desvio padrão dos valores medidos. Para apresentação dos resultados, foi utilizado um intervalo de confiança de 90% calculado com auxílio de uma distribuição *t de Student* [33] com grau de liberdade igual a 4. Essa distribuição deve ser utilizada quando a quantidade de amostras é menor do que 30.

4.3.1 Quantidade máxima de requisições suportadas

Para este teste submeteu-se o conjunto de hospedeiros contendo cada um uma MV com um servidor Web a um *workload* de requisições HTTP em um único cenário, com o

objetivo de avaliar a quantidade máxima de requisições suportadas, analisando-se a variação no tempo de resposta da aplicação Web e a utilização de CPU dos hospedeiros. O consumo de energia dos hospedeiros é medido nesse teste apenas para fins de observação, já que não há preocupação em medir o consumo máximo de energia, lembrando que a qualidade de serviço é a principal variável. A quantidade máxima de requisições determinada é usada como um dos parâmetros para a avaliação de desempenho deste trabalho a fim de evitar que o envio de taxas de requisições acima do máximo suportado gere conclusões errôneas ou deixe o ambiente instável. A taxa de requisições varia entre 5 e 115 requisições por segundo, com incremento de 10 requisições a cada intervalo de 5 minutos. Nas Figuras 11 e 12 são mostrados os gráficos com os resultados quanto aos tempos de resposta médio 7e utilização de CPU.

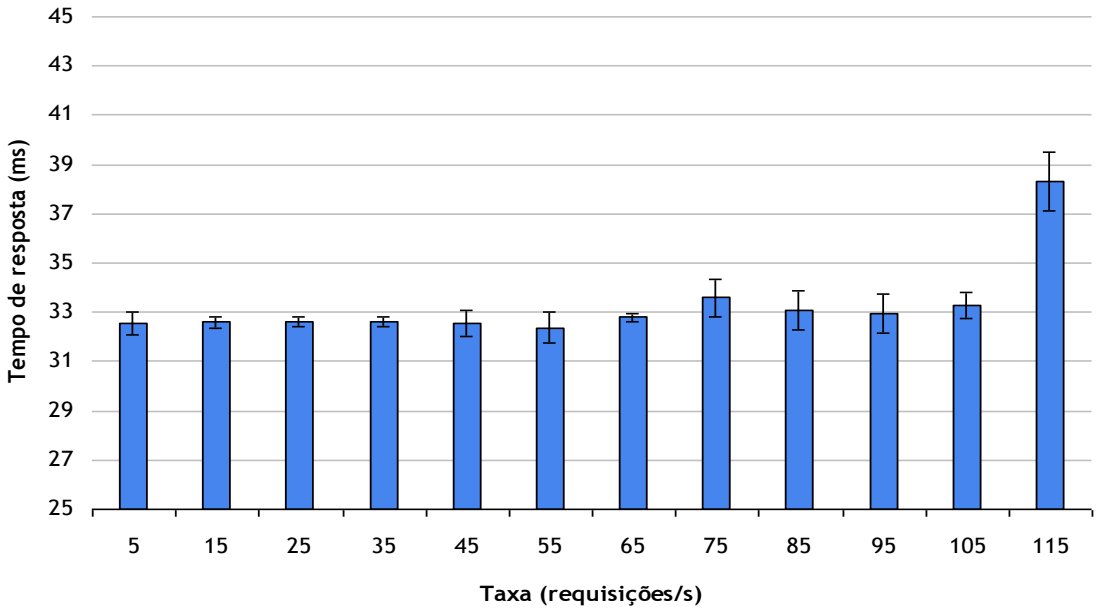


Figura 11 – Quantidade de requisições X tempo de resposta

Ao observarmos os resultados, nota-se que a partir de 115 requisições por segundo, o tempo de resposta aumenta em relação aos valores médios anteriores. Além disso, ao analisar o gráfico da Figura 12, nota-se que o uso de CPU dos hospedeiros Ilha e Itaipu está próximo ao máximo, o que confirma a dificuldade de se manter o tempo de resposta em valores mais baixos. A utilização máxima de CPU foi considerada como 200% em função dos hospedeiros possuírem processadores com dois núcleos.

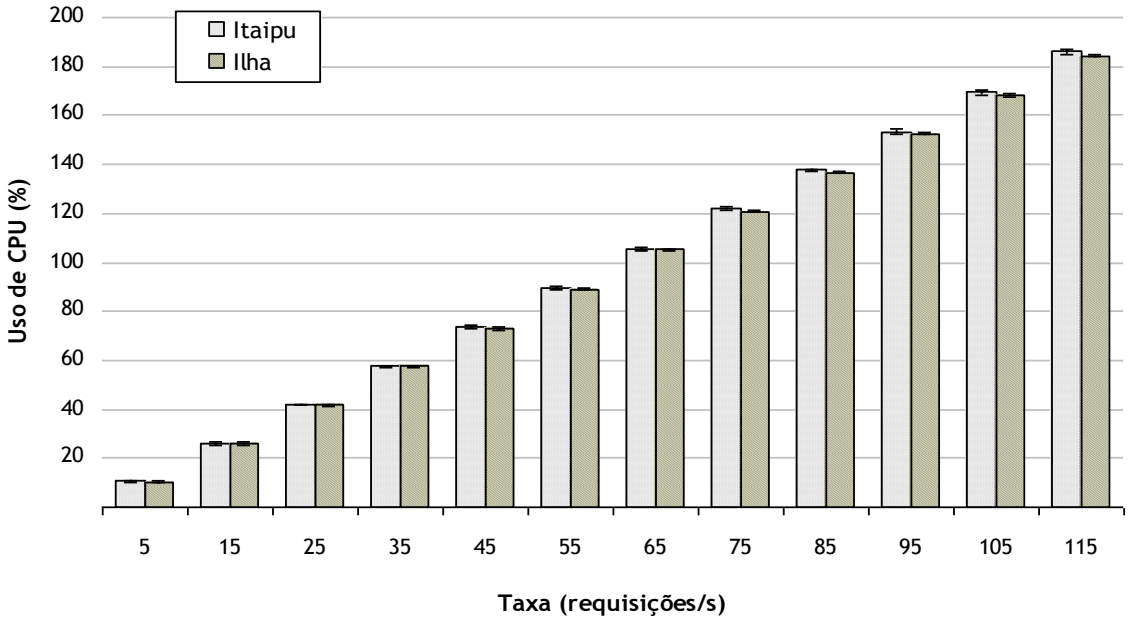


Figura 12 – Quantidade de requisições X uso de CPU

Ainda que fosse possível submeter o conjunto a taxas de requisições superiores a 115, é importante que existam recursos disponíveis para o próprio virtualizador, sob pena de comprometer a gerência das MVs (operações de migração, inclusive). Assim sendo, não foram consideradas para fins de avaliação taxas iguais ou maiores que 115 requisições por segundo. Este tipo de informação é importante em qualquer infraestrutura para dimensionarem-se os recursos adequadamente de acordo com a demanda esperada.

Pode ser observado ainda na Figura 12 que o uso de CPU é praticamente o mesmo nos dois hospedeiros. Isto indica que o balanceador distribui de maneira uniforme as requisições HTTP recebidas do cliente.

A variação do tempo de resposta da aplicação Web observada nos resultados também foi útil para determinar o valor para a restrição de operação (tempo de máximo admitido).

Na Figura 13 é mostrado o consumo de energia dos dois hospedeiros, com caráter meramente informativo, conforme citado anteriormente. Os valores medidos são apresentados em *Watt*, através da conversão dos valores em VA obtidos através da função *getapcpower()* (Seção 3.4.3), usando-se o fator de potência igual a 0,6 do *nobreak* APC.

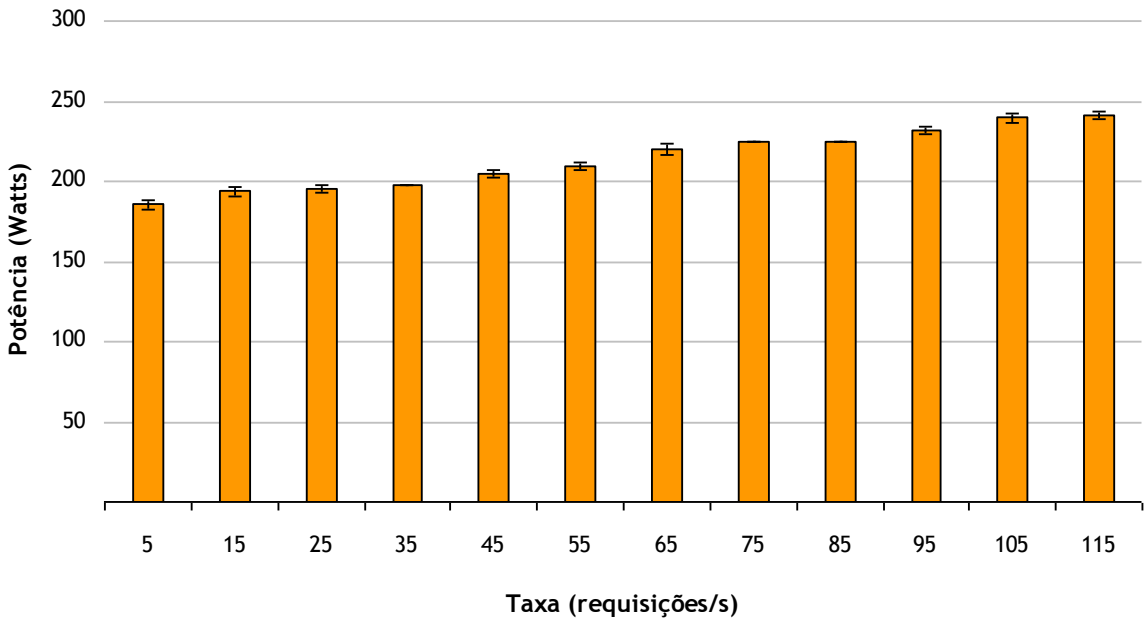


Figura 13 – Quantidade de requisições X consumo de energia

4.3.2 Reconfiguração de hospedeiros

Para avaliar qual é a melhor estratégia de atuação nos recursos dos hospedeiros, de forma a manter qualidade de serviço com economia de energia, foi feito um teste submetendo o conjunto a um único *workload* em (i) um cenário onde os hospedeiros tem apenas um núcleo de CPU ativado na maior frequência e (ii) em outro cenário onde os hospedeiros tem dois núcleos ativados na menor frequência de operação. Durante o funcionamento do sistema, quando os hospedeiros estão com seus recursos configurados no mínimo, os mesmos ficam com apenas um núcleo ativado na menor frequência de operação da CPU.

O objetivo é avaliar a ordem da política de reconfiguração dos hospedeiros (primeiro ligar/desligar ou aumentar/diminuir a frequência da CPU). Em cada cenário foram analisados o tempo de resposta da aplicação Web e o consumo de energia. A taxa de requisições enviadas é fixada em 110 requisições por segundo, a qual deixa os hospedeiros com utilização alta de CPU, conforme observado nos testes da seção anterior.

No cenário (i) o tempo de resposta médio obtido foi de 32,61 ms (IC ± 0,06) enquanto que no cenário (ii) foi de 34,09 ms (IC ± 0,87), o que mostra que, nas condições aqui

apresentadas, é mais vantajoso do ponto de vista de qualidade de serviço, primeiro aumentar a frequência de operação e depois ligar mais núcleos da CPU.

Quanto ao consumo de energia, não foi observada diferença entre os dois cenários nos testes. Entretanto, com o uso de outras ferramentas para sobrecarregar a CPU (*stress tools*), observamos que a ativação de um núcleo gera um pequeno acréscimo no consumo de energia, provavelmente em função da ligação de mais circuitos da CPU.

Assim sendo, optou-se pela política de aumentar a frequência de operação antes de ativar mais núcleos, na implementação do Atuador (Seção 3.2), tendo em vista que é a política que prioriza a qualidade de serviço e que ainda é mais conservativa do ponto de vista de consumo de energia.

4.3.3 Atuação em MVs e hospedeiros

Para medir o tempo necessário para migrar uma MV de hospedeiro para outro, o que é um fator que influencia no tempo de reconfiguração do ambiente, foi realizado um teste simples submetendo-se um servidor Web com duas MVs a uma taxa de 60 requisições por segundo distribuídas pelo balanceador somente para as duas MVs. Esta taxa deixa o hospedeiro com alta utilização de CPU e foi estimada com base no teste apresentado na Seção 4.3.1. Ao fazer a migração de uma das MVs citadas para outro hospedeiro (vazio), obteve-se um tempo médio de migração de 3,4 segundos (IC ± 0,55).

Conforme discutido na Seção 3.4.2, optou-se por colocar os hospedeiros ociosos em estado de espera (*suspend-to-RAM*) ao invés de desligá-los totalmente (*shutdown*). Testes realizados mostraram que os hospedeiros levam em média 95 segundos (IC ± 2,8) para serem inicializados (estando inicialmente desligados), enquanto que se estiverem em estado de espera o tempo de ativação médio é de 6 segundos (IC ± 1,2). Em termos de consumo de energia, os hospedeiros apresentam pouca diferença quando desligados e quando em estado de espera [34]. Assim sendo, a estratégia de suspender os hospedeiros ao invés de desligá-los se mostrou muito vantajosa, já que o tempo de reativação é bem menor e o consumo de energia é praticamente o mesmo entre as duas formas.

4.4 Testes de desempenho

Nesta seção serão mostrados os testes de desempenho realizados com a arquitetura proposta neste trabalho, implementada sob a forma de um sistema de gerência, onde o ambiente descrito na Seção 4.2 é submetido a dois cenários com distintos *workloads* de requisições HTTP. As requisições são geradas no cliente e são distribuídas pelo balanceador para os servidores Web em execução nas MVs.

Foram medidos os tempos de resposta dos servidores Web (através do balanceador), a utilização de CPU dos hospedeiros e o consumo de energia elétrica destes usando-se o monitor de informações de desempenho com os mesmos parâmetros usados nos testes iniciais. Além das medições citadas, através da análise dos registros (*logs*) do sistema foram contabilizadas as transições de frequência dos hospedeiros, as operações de desligamento e religação de núcleos (CPUs), a suspensão e reativação de hospedeiros e as migrações de MVs, para que fosse possível analisar a atuação do sistema e o comportamento geral do ambiente.

Cada um dos cenários foi testado com e sem a utilização do sistema de gerência para avaliar a eficiência deste, tanto do ponto de vista da qualidade de serviço, quanto da economia de energia. Os *workloads* foram executados 5 vezes e foi utilizado um intervalo de confiança de 90% calculado com auxílio da distribuição *t de Student* com grau de liberdade igual a 4.

A restrição de operação de operação (tempo de resposta máximo admitido para a aplicação Web) foi definida como 33,5 ms.

4.4.1 Cenário com *workload* em rampa

Neste cenário é gerado um *workload* com taxa de requisições HTTP crescente, variando entre 5 e 115 requisições por segundo, com incremento de 10 requisições a cada intervalo de 5 minutos. Logo em seguida a taxa decresce na mesma razão, variando entre 105 e 5 requisições por segundo, formando-se uma “rampa”, conforme Figura 14.

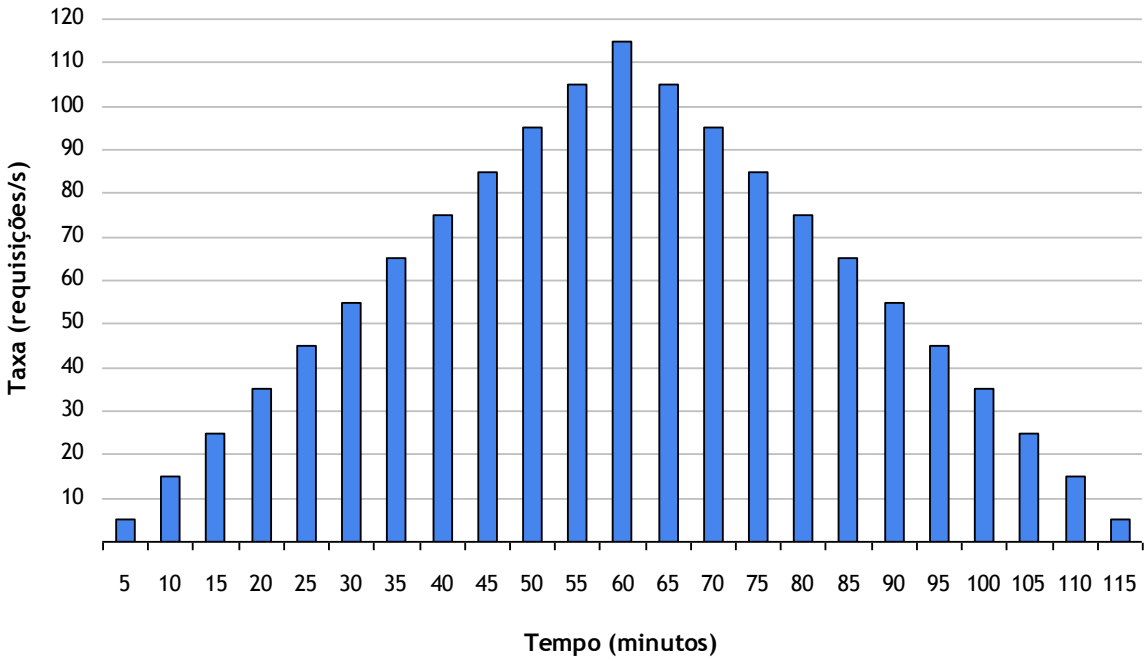


Figura 14 – Variação da taxa de requisições no *workload* em rampa

O objetivo é avaliar a arquitetura proposta quando a mesma atua num ambiente submetido a uma carga com comportamento definido (caracterizado pelo *workload* em rampa). Ao mesmo tempo em que se busca mostrar a eficiência do sistema de gerência, pode-se avaliar as reconfigurações do ambiente validando, assim, o modelo proposto. A seguir são mostrados os resultados obtidos assim como as observações sobre os mesmos.

Na Figura 15 é mostrada a comparação do tempo de resposta médio da aplicação Web antes e depois do uso da arquitetura (sistema de gerência).

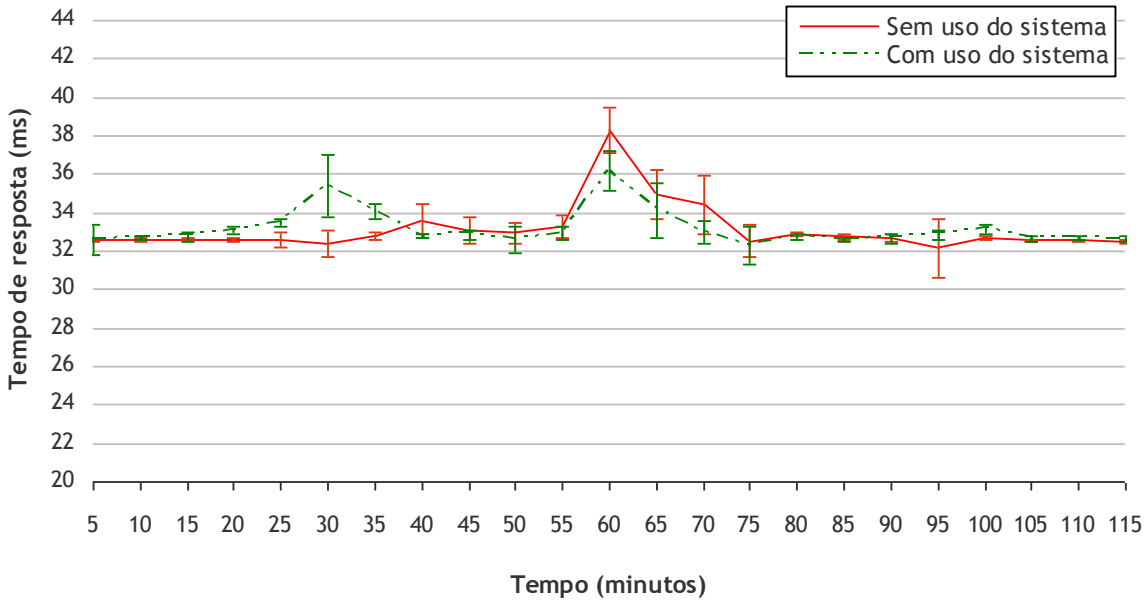


Figura 15 – Tempos de resposta (*workload* em rampa)

O gráfico mostra que os tempos médios de resposta, nas situações com e sem o uso do sistema de gerência, apresentam diferenças apenas no intervalo entre 20 e 35 minutos. Nos demais períodos, os tempos de resposta são compatíveis entre si. Apesar da diferença encontrada, nos intervalos de 20 e 25 minutos, o tempo de resposta encontra-se dentro da restrição de operação (abaixo de 33,5 ms) o que mostra que o sistema apresentou bom desempenho mesmo nessa situação. Analisando-se o *log*, foram identificadas reconfigurações para adicionar recursos ao ambiente entre 25 e 30 minutos (aumento do número de CPUs do host Itaipu, reativação do host Ilha e migração de duas MVs de Itaipu para Ilha). As operações de reconfiguração associadas ao tempo de reação do sistema (100 segundos) explicam a elevação do tempo médio para cerca de 35 ms no período em questão.

Aos 60 minutos, nota-se que o tempo de resposta fica fora da restrição de operação nas duas situações. Neste ponto, a taxa é de 115 requisições por segundo que, conforme apresentado nos testes iniciais, não será considerada para fins de avaliação. Ainda assim, o *workload* incluiu tal taxa para demonstrar que o sistema se mantém estável mesmo nesta situação atípica, sendo registrada no *log*, a informação de que os recursos físicos estão esgotados, de acordo com o que foi apresentado no Capítulo 3.

Numa avaliação geral, do ponto de vista de cumprimento de restrições de operação, o sistema apresentou um bom desempenho. Num total de 22 taxas de requisição submetidas ao

ambiente (desconsiderando a taxa igual a 115) em apenas duas o tempo de resposta ficou acima do máximo admitido, o que representa cerca de 9% de diferença entre as situações com e sem uso do sistema de gerência. Na Figura 16 é mostrado o consumo de energia durante a avaliação de desempenho.

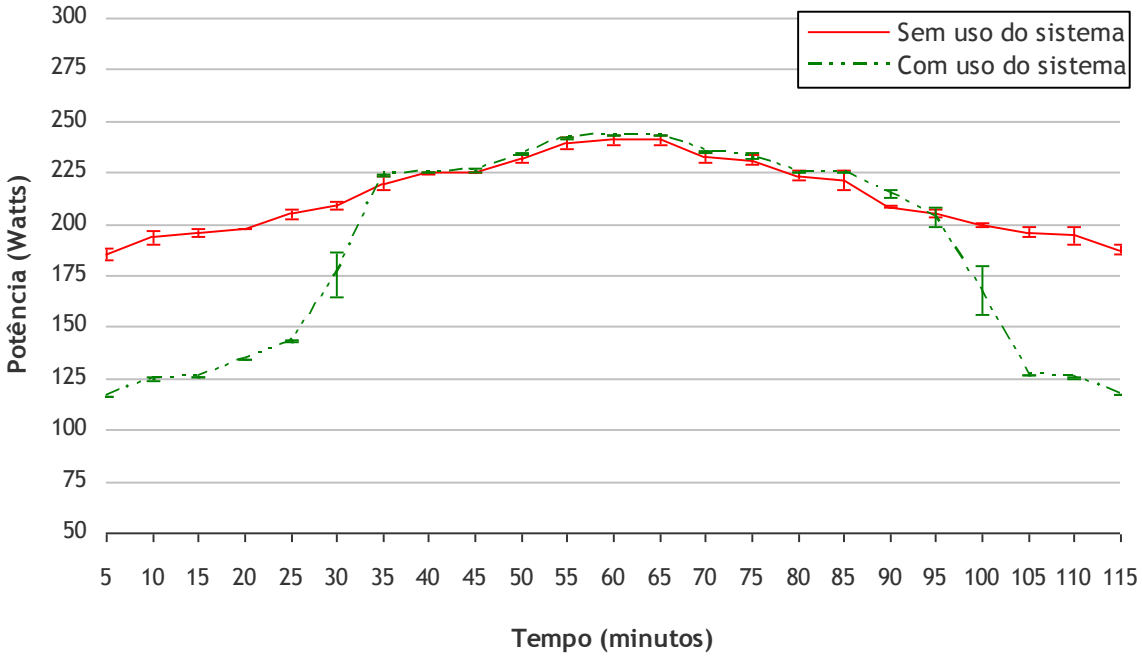


Figura 16 – Consumo de energia (*workload* em rampa)

Analisando os valores do consumo de energia nos testes executados, observa-se que no período entre 5 e 30 minutos, houve em média uma economia de 31,08% com o uso do sistema de gerência. Conforme demonstrado anteriormente, entre 25 e 30 minutos ocorrem reconfigurações no ambiente já que a restrição de operação não está sendo cumprida o que explica a repentina elevação no consumo de energia nesse período.

No período entre 95 e 115 minutos também pode se observar uma queda no consumo de energia. Mais uma vez, através da análise do *log* do sistema de gerência, foi identificado que essa diminuição ocorre em função da reconfiguração do ambiente com redução de recursos (núcleos e frequência), migração de MVs e desativação de um hospedeiro ocioso. Nesse caso a economia média é de 25,04%. Durante todo o período de teste, foram encontradas reduções no consumo de até 37,49% em determinados pontos.

Para uma comparação global em termos de potência, os valores médios de consumo foram transformados em KWh, considerando o período total de duração da avaliação de

desempenho. Quando o ambiente foi submetido ao teste sem o uso do sistema de gerência o consumo global foi de 0,409 KWh enquanto que utilizando o sistema o consumo foi de 0,361 KWh. Isso representa uma economia de 11,75%.

Cabe ressaltar que o comportamento do *workload* influencia diretamente o consumo de energia, ou seja, outros tipos de carga podem permitir uma economia maior ou menor. Entretanto, nos momentos de baixa utilização, o sistema se mostrou eficiente, pois o desperdício de recursos foi evitado com pouco prejuízo do tempo de resposta.

Na Figura 17 é mostrado o gráfico com a utilização de CPU dos hospedeiros Itaipu e Ilha durante o teste.

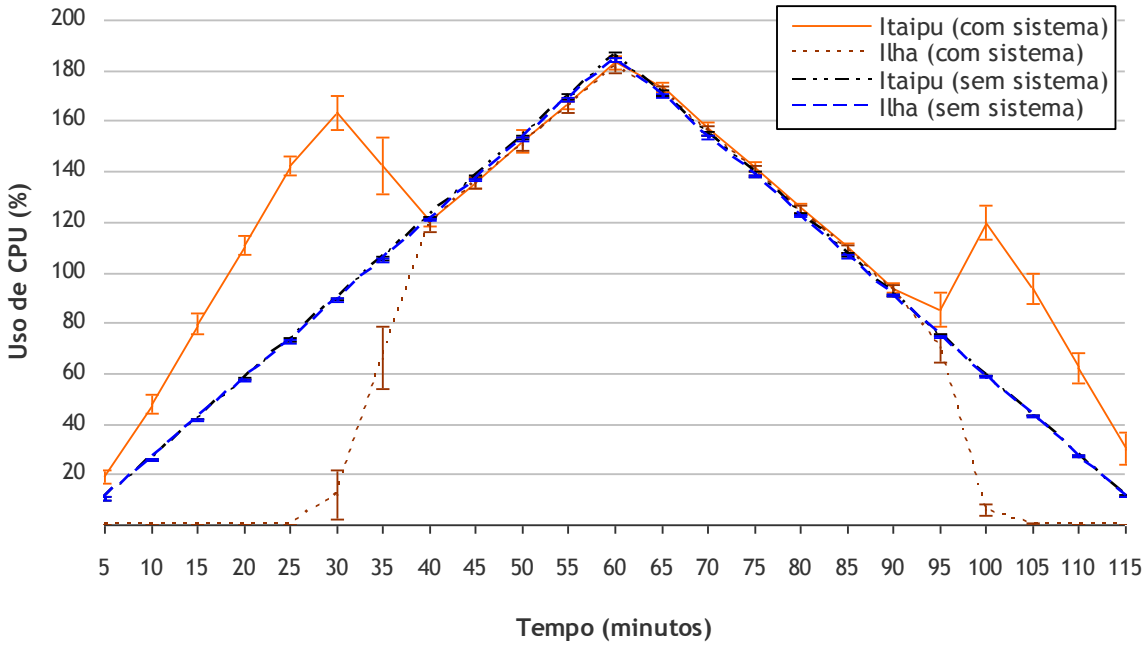


Figura 17 – Utilização de CPU (*workload* em rampa)

Analisando o gráfico em conjunto com as informações do *log* do sistema de gerência, observa-se que a ausência de valores de utilização de CPU do hospedeiro Ilha indica, na verdade, que o mesmo está desativado. Inicialmente só o hospedeiro Itaipu está ativo, suportando todas as MVs.

Após os 25 minutos, tendo em vista que o tempo de resposta sai da restrição de operação (conforme gráfico na Figura 15), que o uso de CPU de Itaipu está elevado e que este já se encontra com todos os seus recursos ativados e no máximo, Ilha é reativado para receber uma das MVs de Itaipu. Após 30 minutos, nota-se que a utilização de CPU de Itaipu sofre

redução em função da migração de mais uma MV para Ilha. A partir, então, dos 40 minutos, a carga dos dois hospedeiros é igual, pois ambos possuem a mesma quantidade de MVs.

A partir dos 60 minutos a taxa de requisições começa a ser reduzida gradativamente e, consequentemente a utilização de CPU também sofre redução. Após 90 minutos, o processo de consolidação começa, migrando uma MV de Ilha para Itaipu. Observa-se que o uso de CPU de Itaipu se eleva, mas não compromete o tempo de resposta (gráfico da Figura 15). Próximo a 95 minutos, mais uma MV é migrada e após 100 minutos, o hospedeiro Ilha é desativado, já que o mesmo fica ocioso (vazio).

Ainda no gráfico da Figura 17 foi incluída a utilização de CPU dos hospedeiros sem o uso do sistema de gerência. Analisando o período entre 5 e 30 minutos, observa-se que o uso dos recursos de Itaipu são maximizados antes de reativar o outro hospedeiro (Ilha). Se comparadas as situações com e sem uso do sistema, isso representa uma utilização mais eficiente de recursos. Entre 95 e 115 minutos foi possível consolidar as MVs num único hospedeiro sem comprometer a restrição de operação da aplicação.

Em resumo, mostramos que a arquitetura propicia um uso mais racional dos recursos, além de garantir a qualidade do serviço e economizar energia.

4.4.2 Cenário com *workload* variável

Diferente do cenário anterior, neste caso o *workload* utilizado não possui um comportamento definido com relação à variação da taxa requisições HTTP enviadas ao longo do tempo. O mesmo foi construído a partir de uma parcela dos *logs* de acesso ao *site* oficial da Copa do Mundo de 1998, obtidos em [35]. Neste evento, foram registrados cerca de 1,35 bilhões de acessos ao *site*.

Desta vez, o objetivo é avaliar o funcionamento do sistema baseado na arquitetura proposta, ao atuar num ambiente cuja carga de requisições submetida ao mesmo apresenta diversas variações ao longo do tempo, validando, assim, o modelo para outros tipos de *workload*, do ponto de vista de qualidade de serviço e economia de recursos.

A partir da análise apresentada por [36] foi escolhido entre os diversos *logs* disponíveis o período compreendido entre o meio-dia do dia 04 de julho de 1998 e as 23h59min do mesmo dia, totalizando 12 horas de registros de requisição.

Para que o tempo do teste fosse reduzido, viabilizando a repetição do mesmo para obtenção de média e desvio padrão mais apurados, o período total de 12 horas foi reduzido para 4 horas. Cada período de 15 minutos do *log* original foi considerado como um período de

5 minutos no novo *workload* gerado. As quantidades de requisição contabilizadas em cada período foram normalizadas para que ficassem compatíveis com a capacidade (taxa) suportada pelo ambiente. As taxas de requisição no *workload* variam entre 21 e 83 requisições por segundo, com mudança de taxa a cada intervalo de 5 minutos.

Nas Figuras 18 e 19 são mostrados os gráficos com as variações das requisições ao longo do tempo, antes e depois da transformação descrita, onde se observa que o comportamento original da carga foi preservado.

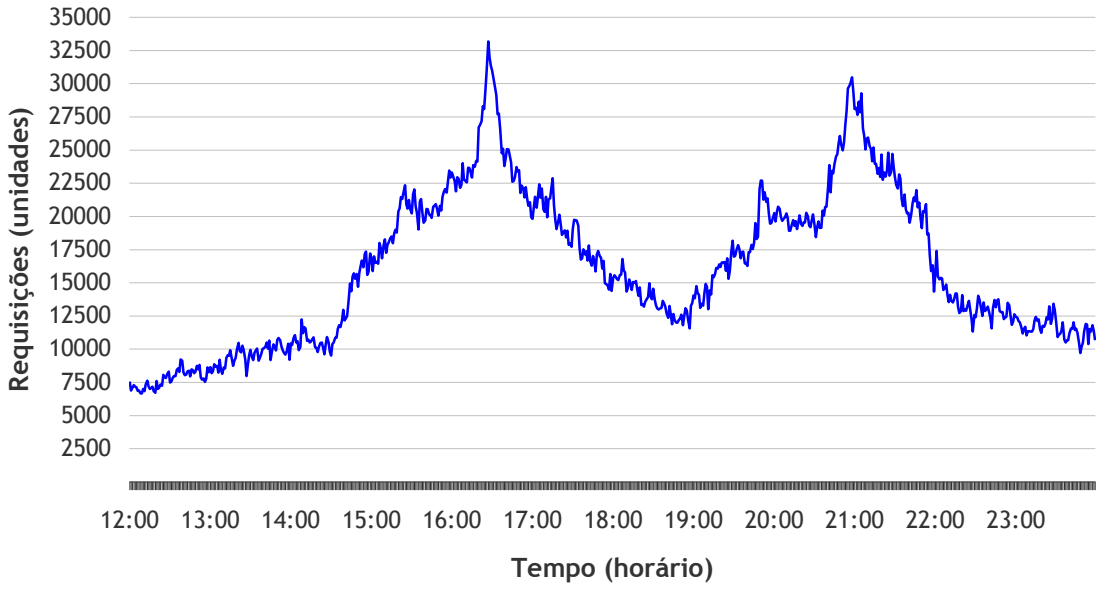


Figura 18 – Variação do volume de requisições (*log* da Copa original)

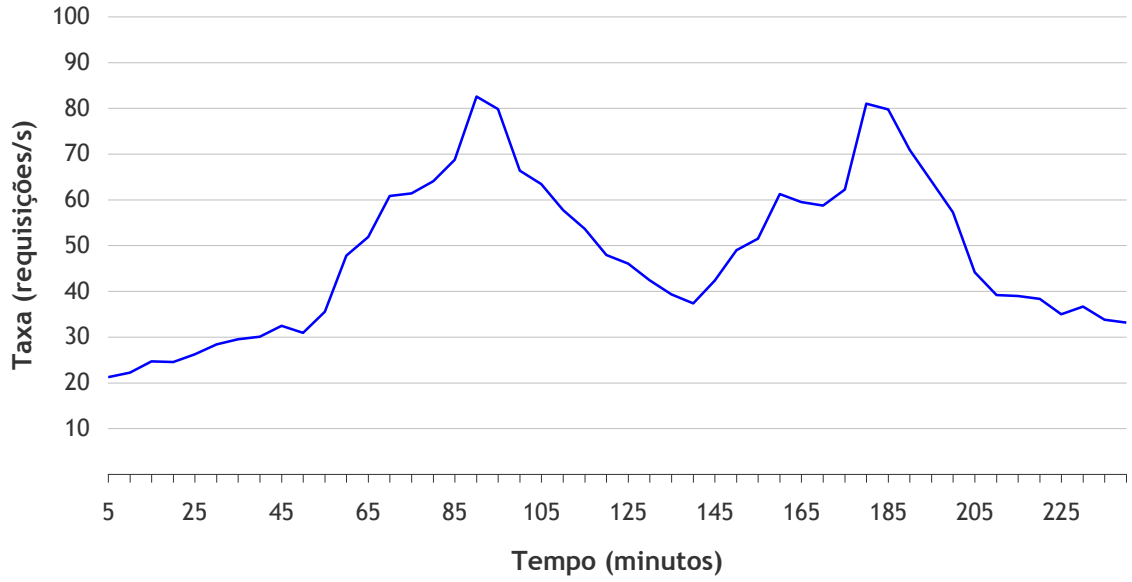


Figura 19 – Variação da taxa de requisições (*log* da Copa normalizado)

A seguir (Figura 20) é mostrado o gráfico com os tempos de resposta médios da aplicação Web, antes e depois do uso do sistema e, logo após, a análise sobre resultados.

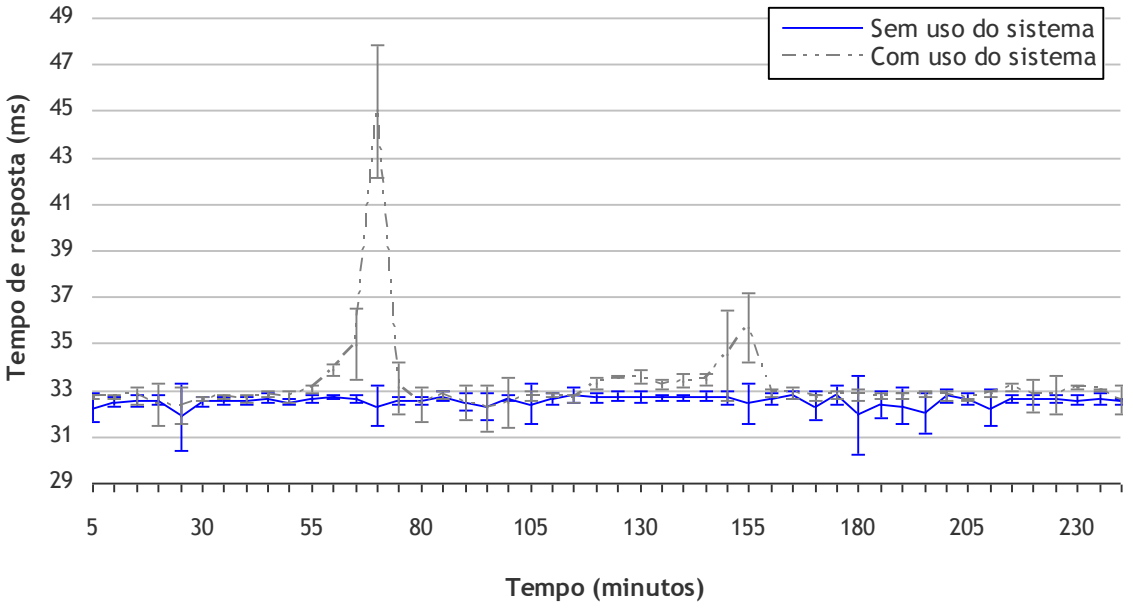


Figura 20 – Tempos de resposta (*workload* variável)

Analisando o gráfico, observa-se que os tempos médios de resposta, nas situações com e sem o uso do sistema de gerência, são diferentes entre si nos intervalos entre 55 e 70 minutos e entre 120 a 155 minutos, sendo compatíveis nos demais intervalos. Dentre os períodos citados, apenas nos intervalos de 60, 65, 70 e 155 minutos o tempo de resposta ficou acima da restrição de operação (33,5 ms), o que mostra que o sistema apresentou um desempenho razoável, apesar das diferenças encontradas em relação aos tempos obtidos sem o seu uso.

De maneira análoga ao cenário em rampa, ao realizar a análise do *log* do sistema de gerência, foram identificadas operações de reconfiguração para aumentar a frequência da CPU e a quantidade de núcleos do hospedeiro Itaipu entre 55 e 60 minutos, ocorrendo em seguida (entre 65 e 70 minutos) a reativação do hospedeiro Ilha para receber MVs de Itaipu através de migração. Desta forma, mostramos que o sistema reagiu em função do tempo de resposta estar acima do máximo admitido e que as operações de reconfiguração reduziram este tempo, conforme se observa no período após 70 minutos.

Entre o período de 120 a 150 minutos, em função da carga de requisições ser menor (conforme gráfico da Figura 19), o sistema reduz a quantidade de recursos utilizados no ambiente. Porém, após 150 minutos como taxa de requisições, que voltou a crescer, não é mais suportada pelo conjunto, novas reconfigurações são realizadas para adicionar recursos, o justifica a elevação do tempo de resposta aos 155 minutos. Mais uma vez o sistema atua e reduz o tempo de resposta da aplicação Web para níveis aceitáveis.

Assim sendo, demonstramos que o sistema é capaz de reagir a diversas situações de acordo com a variação do *workload* aplicado ao ambiente, realizando as operações de reconfiguração necessárias, de acordo com a demanda de requisições, o que reflete diretamente na qualidade do serviço oferecida.

De um total de 240 taxas de requisição submetidas ao ambiente, apenas em 4 a restrição de operação não foi cumprida, o que representa menos de 2% de diferença entre as situações com e sem uso do sistema de gerência. Na Figura 21 é mostrado o consumo de energia durante a avaliação de desempenho.

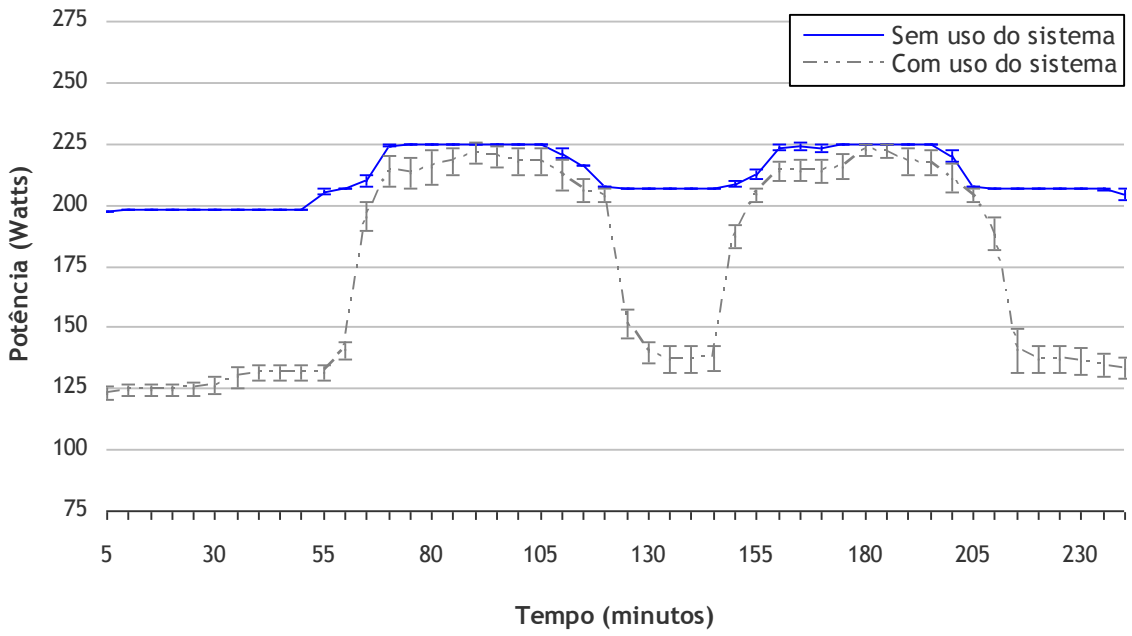


Figura 21 – Consumo de energia (*workload* variável)

Durante todo o período do teste, houve uma economia de energia de em média 18,56% ao se comparar as duas situações (antes e depois do uso do sistema de gerência), sendo o consumo, de um modo geral, igual ou inferior em todos os pontos. Nos períodos entre 5 e 60, 125 e 150 e, ainda, entre 215 e 240 minutos, onde as diferenças no consumo de energia são maiores, a economia média é de 33,35%. Em determinados pontos, foram encontradas reduções de até 37,44% no consumo.

Da mesma forma que no cenário em rampa, a economia de energia ocorreu em função das reconfigurações efetuadas no ambiente (ajuste de frequência, de quantidade de núcleos e desativação de hospedeiros) de acordo com a carga (*workload*).

Em termos de KWh, sem a utilização do sistema de gerência o consumo global foi de 0,848 KWh enquanto que utilizando o sistema o consumo foi de 0,7 KWh, o que representa uma economia de 17,5%. Mais uma vez, cabe ressaltar que o comportamento do *workload* influencia diretamente o consumo de energia. No *workload* da seção anterior obteve-se 11,75% de economia.

Nas Figuras 22 e 23 são mostrados os gráficos com a utilização de CPU dos hospedeiros Itaipu e Ilha durante o teste.

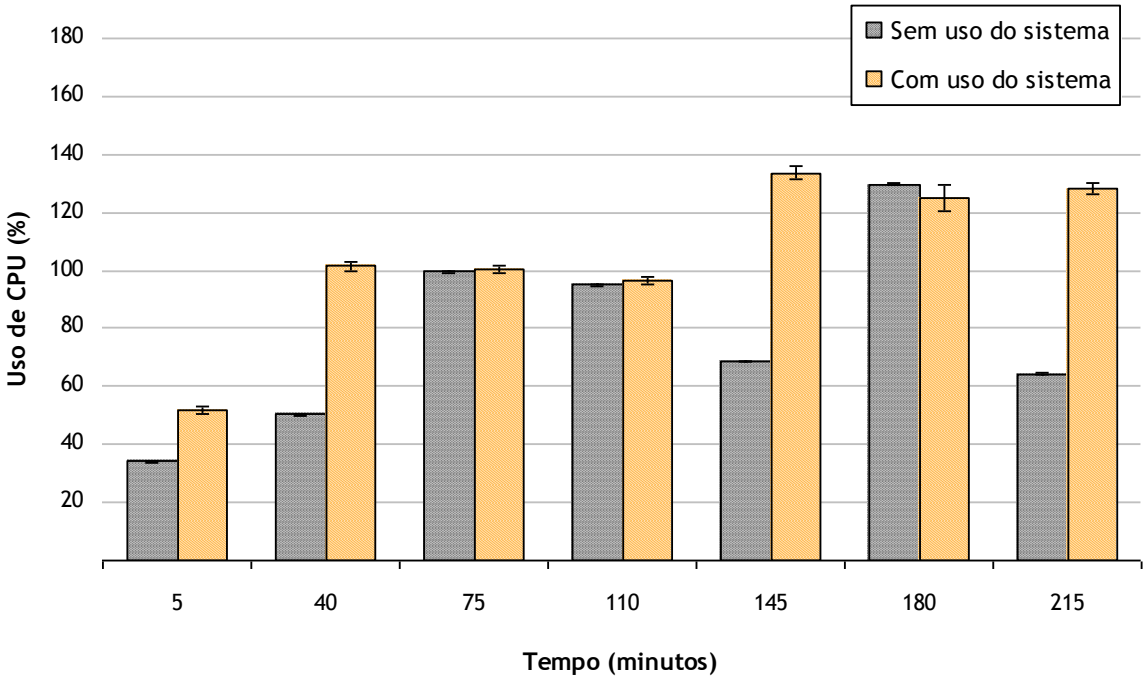


Figura 22 – Utilização de CPU (hospedeiro Itaipu)

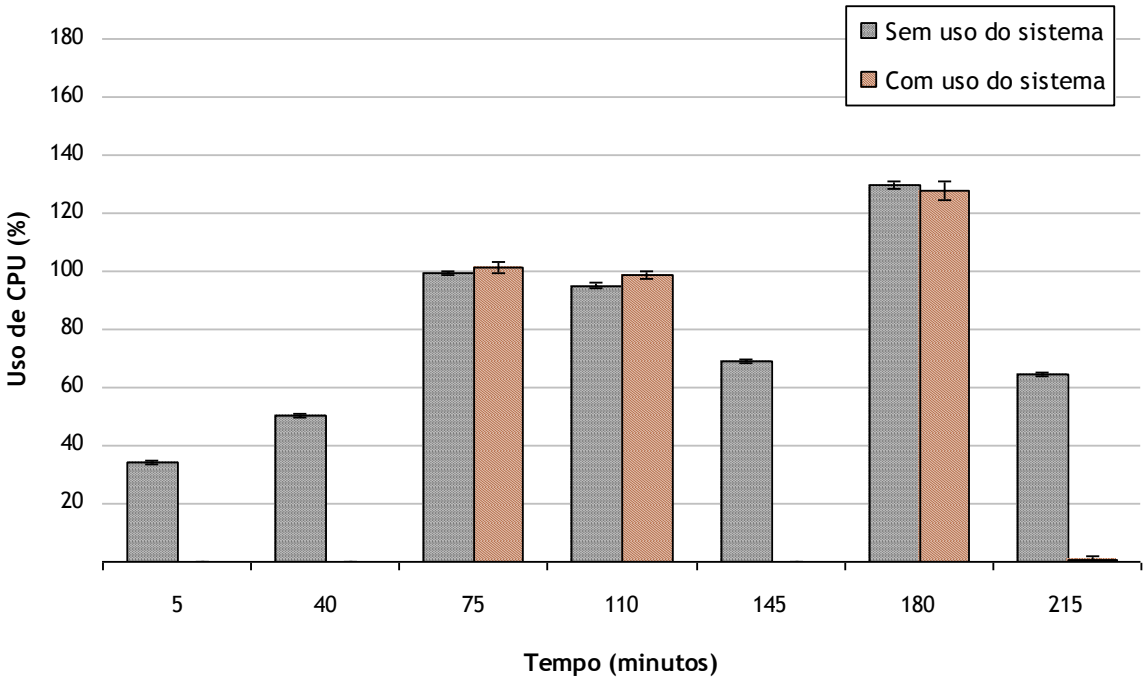


Figura 23 – Utilização de CPU (hospedeiro Ilha)

Para permitir uma melhor visualização e comparação dos resultados, foram selecionados 7 períodos de medição, reduzindo a quantidade de valores a serem exibidos nos gráficos. Apesar das perdas (medições intermediárias), é possível observar o aproveitamento dos recursos (CPU) de ambos os hospedeiros, com e sem a utilização do sistema de gerência.

Na Figura 22, observa-se que a utilização de CPU do hospedeiro Itaipu é sempre igual ou maior com o uso do sistema do que sem. Isto demonstra um melhor aproveitamento dos recursos ao se utilizar a arquitetura proposta, de maneira similar ao cenário com o *workload* em rampa (seção anterior).

Na Figura 23, nos períodos de 5, 40, 145 e 215 minutos o hospedeiro Ilha está desativado e por isso não consta no gráfico sua utilização de CPU com o uso do sistema, nos intervalos citados. Já nos períodos de 75, 110 e 180 minutos, o hospedeiro Ilha está ativo e nota-se que sua utilização de CPU é igual ou maior quando utilizado o sistema de gerência do que sem o mesmo. Mais uma vez, demonstramos que há uma utilização mais racional dos recursos (maximização), com pouco prejuízo da qualidade de serviço, conforme mostrado nos gráficos e análises anteriores.

4.5 Considerações

Quanto à escabilidade da arquitetura para permitir o controle de diversas aplicações, considerando que todas irão compartilhar o mesmo ambiente (uma nuvem, por exemplo), talvez seja necessário fazer adaptações no modelo para que políticas diferenciadas possam coexistir, de forma a atender às restrições de operação individuais de cada aplicação. Esse tipo de abordagem está relacionado à oferta de serviços diferenciados, que podem ser adquiridos pelos clientes de acordo com o nível de qualidade de serviço requerido e custo correspondente. De maneira análoga, serviços de hospedagem de *sites* contam com essa abordagem para conseguir atender seus clientes com redução de custos.

Na medição do consumo de energia dos hospedeiros observamos que os valores de potência ficaram restritos ao “degrau” de 1% (15 VA) do *nobreak* APC utilizado. Ainda assim, os resultados mostraram que a proposta é funcional, embora os valores possam ter sido truncados em função da limitação do equipamento.

Uma vez que os servidores utilizados como hospedeiros só permitem o uso de duas frequências de CPU no mecanismo de DVFS, a variação no consumo de energia assim como o tempo de resposta da aplicação podem ter sofrido pequenas interferências, pois a otimização do uso dos recursos ficou limitada às frequências disponíveis. Uma gama maior de

frequências poderia propiciar um refinamento dos valores medidos, embora essa limitação não invalide o modelo apresentado.

De maneira análoga, a economia de energia em função da desativação de hospedeiros ficou restrita à quantidade de equipamentos disponíveis na implementação (2). Certamente em ambientes com maior número de servidores os percentuais de economia de energia podem ser ainda maiores.

5 CONCLUSÕES

Neste trabalho foi apresentada uma arquitetura reutilizável para gerência de uma infraestrutura usando máquinas virtuais, para prover suporte a aplicações Web, visando à qualidade de serviço com economia de energia. Para atingir estes objetivos, a estratégia foi fazer um uso mais racional dos recursos físicos, atuando nos parâmetros de operação das CPUs dos hospedeiros, desativando os que estiverem ociosos e redistribuindo dinamicamente as MVs, utilizando migração "ao vivo". As operações são executadas de acordo com a demanda de serviços, tendo as restrições de operação da aplicação como principal parâmetro.

Conforme mostrado no desenvolvimento do trabalho, a arquitetura é reutilizável, pois foi desenhada de maneira modular, dividindo as função de controle, coleta de informações e reconfiguração (atuação) em módulos individuais que se relacionam entre si. Desta forma, se o modelo apresentado for utilizado em outras soluções, como, por exemplo, servidores sem virtualização, basta adaptar o módulo de atuação, preservando a arquitetura e as políticas propostas. Como não existe dependência de plataforma, a proposta também pode ser implementada com outras soluções de virtualização ou sistemas operacionais, bastando portar a biblioteca da API HVMM, desenvolvida para dar suporte às interações com o virtualizador e com o ambiente, através de uma camada de alto nível.

Na avaliação de desempenho, mostramos que a arquitetura é funcional através do uso de um sistema implementado em PERL submetido a testes com envio de requisições Web em diversos cenários. Além disso, mostramos que com o uso do sistema foi possível obter redução no consumo de energia, nos cenários apresentados, mantendo a qualidade de serviço da aplicação em níveis aceitáveis.

Como trabalhos futuros, sugere-se uma avaliação de outras política de reconfiguração diferentes da utilizada (Seção 3.4) de forma a otimizar os parâmetros de operação, sem prejuízos da qualidade de serviço. Por exemplo, pode ser avaliada a mudança dinâmica da ordem das operações de reconfiguração, em função do tipo ou comportamento da carga (requisições), usando previsores de carga, como o proposto por [10]. O uso de previsores de carga permitiria também antecipar o acréscimo de recursos ao ambiente, reduzindo, assim, as eventuais violações das restrições de operação, da mesma forma que seria possível economizar mais energia ao desativar recursos que não serão necessários, em função da demanda futura. Outra sugestão é diminuir dinamicamente o tempo de espera entre as operações de reconfiguração, aumentando assim a velocidade de reação do sistema, em função do aumento de utilização (carga).

Devido à transparência que as funções *hvmselecthost()* e *hvmselectvm()* apresentam no contexto da arquitetura, conforme apresentado na Seção 3.4.1, outras estratégias de escolha de MVs a serem migradas e de hospedeiros candidatos também podem ser avaliadas, buscando otimização do processo.

Na verificação do cumprimento de restrições de operação pelo elemento Controlador, futuramente serão considerados dois limiares de comparação (mínimo e máximo) de forma a suavizar o disparo de reconfigurações, já que no trabalho aqui apresentado, só é considerando um limiar (máximo) o que deixa pouca tolerância no teste com os valores obtidos.

Outro campo que pode ser explorado é o uso de restrições de operação ligadas à energia ou potência consumida. Pode ser feita uma análise na correlação entre o uso de CPU e o comportamento das aplicações, de acordo com a variação da carga e tipo de processamento. Embora os processadores sejam os responsáveis pela maior parte do gasto de energia em sistemas computacionais, o consumo dos outros dispositivos (memória, discos, etc) e dos circuitos de apoio não pode ser totalmente desprezado, principalmente quando se tratam de processos que não sejam do tipo *CPU bound* [37].

E, por fim, devem ser feitos mais testes com a proposta apresentada para que seja avaliada a escalabilidade quando adicionados mais hospedeiros, uma vez que optamos por centralizar a gerência e as medições de desempenho num único nó ao invés de usar mecanismos como agentes de recursos ou federação, como proposto, por exemplo, por [38]. Além disso, os testes realizados consideraram uma única aplicação Web. Há de se considerar, em trabalhos futuros, como a arquitetura pode ser configurada para dar suporte a múltiplas aplicações, cada qual com suas restrições de operação.

REFERÊNCIAS

[1] GOVINDAN, S.; CHOI, J.; URGAONKAR, B.; SIVASUBRAMANIAM, A.; BALDINI, A. Statistical profiling-based techniques for effective power provisioning in data centers. In: Proceedings of the 4th ACM European conference on Computer systems. New York, NY, USA: ACM, março de 2009. p. 317-330.

[2] MEVÅG, I. Towards Automatic Management and Live Migration of Virtual Machines. 2007. 107 p. Master's thesis - Oslo University College (UiO / HiO), University of Oslo, Norway, maio de 2007.

[3] WANG, X.; WANG, Y. Co-con: Coordinated control of power and application performance for virtualized server clusters. In: Proceedings of the 17th IEEE International Workshop on Quality of Service. Charleston, USA: [s.n.], julho de 2009. p. 1-9.

[4] KUSIC, D.; KEPHART, J. O.; HANSON, J. E.; KANDASAMY, N.; JIANG, G. Power and performance management of virtualized computing environments via lookahead control. In: Proceedings of the 2008 International Conference on Autonomic Computing. Washington, DC, USA: IEEE Computer Society, junho de 2008. p. 3-12.

[5] RUTH, P.; RHEE, J.; XU, D.; KENNELL, R.; GOASGUEN, S. Autonomic Live Adaptation of Virtual Computational Environments in a Multi-Domain Infrastructure. In: Proceedings of the 2006 IEEE International Conference on Autonomic Computing. Washington, DC, USA: IEEE Computer Society, junho de 2006. p. 5-14.

[6] GOIRI, I.; JULIA, F.; EJARQUE, J.; DE PALOL, M.; BADIA, R. M.; GUITART, J.; TORRES, J. Introducing Virtual Execution Environments for Application Lifecycle Management and SLA-Driven Resource Distribution within Service Providers. In: Proceedings of the 8th IEEE International Symposium on Network Computing and Applications. Washington, DC, USA: IEEE Computer Society, julho de 2009. p. 211-218.

[7] IQBAL, W. Service Level Agreement Driven Adaptive Resource Management For Web Applications on Heterogeneous Compute Clouds. 2009. 46 p. Master's thesis (Master in Information Technology) - Barcelona School of Informatics, University of Catalunya, Spain, setembro de 2009.

[8] HAYES, B. Cloud Computing. Communications of the ACM, v. 51, n.7, p. 9-11, julho de 2008.

[9] PETRUCCI, V. A framework for supporting dynamic adaptation of power-aware web server clusters. 2008. 61 p. Master's thesis (Master in Science) - Instituto de Computação, Universidade Federal Fluminense, Niterói, Brasil, dezembro de 2008.

[10] SANT´ANA, C. H. S. Previsão de Carga em Aglomerado de Servidores Web Aplicada a Economia de Energia. 2010. 76 p. Dissertação (Mestrado em Ciências) - Instituto de Computação, Universidade Federal Fluminense, Niterói, Brasil, março de 2010.

[11] BERTINI, L.; LEITE, J. C. B.; MOSSÉ, D. Statistical QoS Guarantee and Energy-efficiency in Web Server Clusters. In: 19th Euromicro Conference on Real-Time Systems. Washington, DC, USA: IEEE Computer Society, julho de 2007. p. 83-92.

[12] GOLDBERG, R. Survey of virtual machine research. IEEE Computer, v. 7, n. 6, p. 34–45, junho de 1974.

[13] ROSE, R. Survey of System Virtualization Techniques. 2004. 15 p. Disponível em <http://www.robertwrose.com/vita/rose-virtualization.pdf>. Acesso em: maio de 2010.

[14] CAMARGO, R. Y.; GOLDCHLEGER, A.; KON, F.; GOLDMAN, A. Checkpointing BSP parallel applications on the InteGrade Grid middleware. Concurrency and Computation: Practice and Experience, v. 18, n. 6, p. 567–579, maio de 2006.

[15] BARHAM, P.; DRAGOVIC, B.; FRASER, K.; HAND, S.; HARRIS, T.; HO, A.; NEUGEBAUER, R.; PRATT, I.; WARFIELD, A. Xen and the Art of Virtualization. In: Proceedings of 19th ACM Symposium on Operating Systems Principles. New York, NY, USA: ACM, outubro de 2003. v. 37, p. 164-177.

[16] REDHAT Emerging Technology Project (2010b). Virt-Manager: RedHat Virtual Machine Building Tool. Disponível em: <http://virt-manager.et.redhat.com/>. Acesso em: fevereiro de 2010.

[17] BOHRER, P.; ELNOZAHY, E. N.; KELLER, T.; KISTLER, M.; LEFURGY, C.; MCDOWELL, C.; RAJAMONY, R. The case for power management in web servers. Power Aware Computing, p. 261-289, 2002.

[18] VMWARE. Página da empresa VMware Inc. Disponível em: <http://www.vmware.com>. Acesso em: março de 2010.

[19] REDHAT Emerging Technology Project (2010a). LIBVIRT: the virtualization API. Disponível em: <http://libvirt.org/>. Acesso em: janeiro de 2010.

[20] DMTF. Distributed Management Task Force. CIM: Common Information Model. Disponível em: <http://www.dmtf.org/standards/cim/>. Acesso em: abril de 2010.

[21] NIST. Introduction to time series analysis. Section 6.4 of NIST/SEMATECH e-handbook of statistical methods. Disponível em: <http://www.itl.nist.gov/div898/handbook/>. Acesso em: março de 2010.

[22] OLIVEIRA, C.; PETRUCCI, V.; LOQUES, O. Impact of server dynamic allocation on the response time for energy-efficient virtualized web clusters. In: Proceedings of the 12th Brazilian Workshop on Real-Time and Embedded Systems. Rio Grande do Sul, Brasil: SBC, maio de 2010. p. 39-48.

[23] SYSOEV, I. Nginx: a free, open-source, high-performance HTTP server and reverse proxy. Disponível em: <http://nginx.net/>. Acesso em: abril de 2010.

[24] ACPI. Advanced Configuration and Power Interface: an open industry specification co-developed by Hewlett-Packard Corporation, Intel Corporation, Microsoft Corporation, Phoenix Technologies Ltd. and Toshiba Corporation. Disponível em: <http://www.acpi.info/>. Acesso em: fevereiro de 2010.

[25] LIEBERMAN, P. Wake-on-LAN technology. 2006. 9 p. Disponível em: <http://www.liebsoft.com/index.cfm/whitepapers/Wake_On_LAN>. Acesso em: janeiro de 2010.

[26] FACCHETTI, R. Monitoring Your UPS with apcupsd. Linux Journal, v. 2000, p. 1-8, nov. 2000.

[27] MOSBERGER, D.; Jin, T. HTTPERF: A tool for measuring web server performance. ACM SIGMETRICS Performance Evaluation Review, v. 26, n. 3, p. 31-37, dezembro de 1998.

[28] MIDGLEY, J. T. J. Autobench: a PERL wrapper around httperf for automating benchmarking. Disponível em: <http://www.xenoclast.org/autobench>. Acesso em: fevereiro de 2010.

[29] CACTI. The complete rrdtool-based graphing solution. Disponível em: <http://www.cacti.net/>. Acesso em: janeiro de 2010.

[30] OETIKER, T. RRDtool: the OpenSource industry standard, high performance data logging and graphing system for time series data. Disponível em: <http://oss.oetiker.ch/rrdtool/>. Acesso em: janeiro de 2010.

[31] APACHE Software Foundation. Apache HTTP Server Project. Disponível em: <http://httpd.apache.org>. Acesso em: fevereiro de 2010.

[32] FREE-WEBHOSTS.com. Free PHP Benchmark Performance Script. Disponível em: <http://www.free-webhosts.com/php-benchmark-script.php>. Acesso em: janeiro de 2010.

[33] ZABELL, S. L. On Student's 1908 Article "The Probable Error of a Mean". Journal of the American Statistical Association. v. 103, n. 481, p. 1-7, março de 2008.

[34] BERTINI, L., LEITE, J. C. B.; MOSSÉ, D. Optimal Dynamic Configuration in Webserver Clusters. RT-1/08. Niterói, Brasil: IC-UFF, 2008. 27 p. Relatório Técnico.

[35] DANZIG, P.; MOGUL, J.; PAXSON, V.; SCHWARTZ, M. The Internet Traffic Archive. Disponível em: <http://ita.ee.lbl.gov/>. Acesso em: junho de 2010.

[36] ARLITT, M.; JIN, T. Workload characterization of the 1998 World Cup Web site. IEEE Network, v. 14, n. 3, p. 30-37, maio de 2000.

[37] SILBERSCHATZ, A.; GALVIN, P. B. Operating System Concepts. 3th ed. Boston, USA: Addison-Wesley Longman Publishing Co., Inc, 1993. p. 104.

[38] RODRIGUES, A. L. B. Uma Infra-estrutura para Monitoramento de Sistemas Cientes de Contexto. 2009. 137 p. Dissertação (Mestrado em Ciências e Engenharia Eletrônica) - Faculdade de Engenharia, Universidade do Estado do Rio de Janeiro, Brasil, agosto de 2009.

APÊNDICE A – Arquivos appsrv.xml e hvms.xml

appsrv.xml

```
<?xml version='1.0'?>
<appsrv>
        <hostname>ipanema</hostname>
        <maxrt>33.5</maxrt>
        <rturl>http://ipanema/resp_time.html</rturl>
</appsrv>
```

hvms.xml

```
<?xml version='1.0'?>
<hvms>
        <hvm>
                <hostname>ilha</hostname>
                <cpus>2</cpus>
                <freqs>1998,2997</freqs>
                <mac>00:1C:C0:71:77:1A</mac>
                <maxcpu>0.70</maxcpu>
                <mincpu>0.45</mincpu>
                <nfssrv>0</nfssrv>
                <url>http://ilha:9363</url>
        </hvm>
        <hvm>
                <hostname>itaipu</hostname>
                <cpus>2</cpus>
                <freqs>1998,2997</freqs>
                <mac>00:1C:C0:6C:92:82</mac>
                <maxcpu>0.70</maxcpu>
                <mincpu>0.45</mincpu>
                <nfssrv>1</nfssrv>
                <url>http://itaipu:9363</url>
        </hvm>
</hvms>
```

APÊNDICE B – Objeto utilizado na API HVMM

```
hvm = object
        conn: ponteiro;       # conexão com o host
        cpus: inteiro;        # número total de CPUs do host
        cpuson: inteiro;      # número de CPUs ligadas
        cpuavg1: real;        # média de uso da CPU (atual)
        cpuavg2: real;        # média de uso da CPU (anterior)
        freq: inteiro;        # frequência atual da CPU (MHz)
        freqs: inteiro;       # frequências de CPU disponíveis (MHz)
        hostname: string;     # nome do host
        mac: string;          # endereço MAC do host
        maxcpu: real;         # uso máximo de CPU
        mincpu: real;         # uso mínimo de CPU
        nfssrv: inteiro;      # indica se o host é servidor NFS
        status: string;       # status do host (ativo ou inativo)
        session: ponteiro;    # sessão com o host
        url: string;          # URL para conexão
        user: string;         # usuário para conexão via Xen API
        passwd: string;       # senha para conexão via Xen API
        vmcount: inteiro;     # quantidade de MVs do host
        vmlist: string;       # lista de MVs do host
      end;
```

APÊNDICE C – Funções da API HVMM

Função: hvmconnect
Funcionalidade: abre conexão com um hospedeiro
Parâmetros: objeto hvm do hospedeiro alvo
Retorno: (ponteiro) conexão para o hospedeiro ou nulo em caso de erro

Função: hvmgetcpu
Funcionalidade: obtém utilização de CPU de um hospedeiro
Parâmetros: objeto hvm do hospedeiro alvo
Retorno: (real) valor da utilização da CPU ou -1 em caso de erro

Função: hvmgetcpuson
Funcionalidade: obtém quantidade de CPUs ou núcleos ligados em um hospedeiro
Parâmetros: objeto hvm do hospedeiro alvo
Retorno: (inteiro) quantidade de CPUs ligadas ou -1 em caso de erro

Função: hvmgetvmcount
Funcionalidade: obtém quantidade de MVs em um hospedeiro
Parâmetros: objeto hvm do hospedeiro alvo
Retorno: (inteiro) quantidade de MVs

Função: hvmgetvmlist
Funcionalidade: obtém lista das MVs de um hospedeiro
Parâmetros: objeto hvm do hospedeiro alvo
Retorno: (string) lista das MVs ou nulo em caso de erro

Função: hvmgetspeed
Funcionalidade: obtém frequência atual de operação da CPU de um hospedeiro
Parâmetros: objeto hvm do hospedeiro alvo
Retorno: (inteiro) frequência atual em MHz ou -1 em caso de erro

Função: hvmmigratevm
Funcionalidade: faz migração ao vivo de uma MV entre dois hospedeiros
Parâmetros: nome da MV
objeto hvm do hospedeiro origem
objeto hvm do hospedeiro destino
Retorno: (inteiro) 0 em caso de sucesso ou -1 em caso de erro

Função: hvmselecthost
Funcionalidade: seleciona um hospedeiro destino para migração de MV
Parâmetros: lista de objetos hvm dos hospedeiros
número do hospedeiro exceção
tipo de seleção (1 = balanceamento / 2 = consolidação)
Retorno: (inteiro) número do hospedeiro escolhido ou -1 em caso de erro

Função: hvmselectvm
Funcionalidade: seleciona uma MV do hospedeiro para efetuar migração
Parâmetros: objeto hvm do hospedeiro alvo
Retorno: (string) nome da MV a ser migrada ou -1 em caso de erro

Função: hvmsession
Funcionalidade: abre sessão com um hospedeiro (após uso de hvmconnect)
Parâmetros: objeto hvm do hospedeiro alvo
Retorno: (ponteiro) sessão com o hospedeiro ou nulo em caso de erro

Função: hvmsetspeed
Funcionalidade: define frequência de operação da CPU de um hospedeiro
Parâmetros: objeto hvm do hospedeiro alvo
frequência em MHz
Retorno: (inteiro) 0 em caso de sucesso ou -1 em caso de erro

Função: hvmsetcpuson
Funcionalidade: determina a quantidade de CPUs ou núcleos que devem estar ligados
Parâmetros: objeto hvm do hospedeiro alvo
quantidade de CPUs ou núcleos
Retorno: (inteiro) 0 em caso de sucesso ou -1 em caso de erro

Função: hvmsuspend
Funcionalidade: desativa um hospedeiro colocando-o em estado de espera
Parâmetros: objeto hvm do hospedeiro alvo
Retorno: (inteiro) 0 em caso de sucesso ou -1 em caso de erro

Função: hvmwakeup
Funcionalidade: seleciona um hospedeiro inativo e reativa o mesmo
Parâmetros: lista de objetos hvm dos hospedeiros
Retorno: (inteiro) número do hospedeiro disponível ou -1 em caso de erro

APÊNDICE D – Alterações realizadas na Xen API

XendAPI.py

```
905     host_funcs = [('get_by_name_label', None),
906                   ('list_methods', None),
907                   ('suspend', None)]

991     def host_suspend(self, session, host_ref):
992         if not XendDomain.instance().allow_new_domains():
993             return xen_api_error(XEND_ERROR_HOST_RUNNING)
994         return xen_api_success(XendNode.instance().suspend())

1070    host_cpu_funcs  =  [('set_speed', None),
1071                        ('set_status', None),
1072                        ('get_status', None)]

1114    def host_cpu_set_speed(self, _, new_speed):
1115        return xen_api_success(XendNode.instance().set_host_cpu_speed(new_speed))
1116    def host_cpu_get_status(self, _, cpu):
1117        return xen_api_success(XendNode.instance().get_host_cpu_status(cpu))
1118    def host_cpu_set_status(self, _, cpu, status):
1119        return xen_api_success(XendNode.instance().set_host_cpu_status(cpu, status))
```

XendNode.py

```
315     def suspend(self):
316         try:
317             f = file('/sys/power/state', 'w')
318             f.write("mem")
319             f.close()
320             return 0
321         except:
322             raise XendError('Error Suspending Host')
```

```
455     def get_host_cpu_field(self, ref, field):
456         try:
457             if field == "speed":
458                 return int(float(parse_proc_cpuinfo()[0]['cpu MHz']))
459             else:
460                 return self.cpus[ref][field]
461         except KeyError:
462             raise XendError('Invalid CPU Reference')
500     def set_host_cpu_speed(self, new_speed):
501         try:
502             f = file('/sys/devices/system/cpu/cpu0/cpufreq/scaling_setspeed', 'w')
503             print >> f, new_speed
504             f.close()
505             return 0
506         except:
507             raise XendError('Error Setting CPU Speed')

510     def get_host_cpu_status(self, cpu):
511         try:
512             f = file('/sys/devices/system/cpu/cpu' + str(cpu) + '/online', 'r')
513             return f.read(1)
514         except:
515             raise XendError('Error Getting CPU Status')

518     def set_host_cpu_status(self, cpu, status):
519         try:
520             f = file('/sys/devices/system/cpu/cpu' + str(cpu) + '/online', 'w')
521             f.write(str(status))
522             return 0
523         except:
524             raise XendError('Error Setting CPU Status')
```