Universidade Federal do Ceará
Departamento de Engenharia de Teleinformática
Programa de Pós Graduação em Engenharia de Teleinformática

Deborah Maria Vieira Magalhães

Uma Estratégia de Migração Dinâmica de Máquinas Virtuais para Economia de Energia em Ambientes Computacionais Virtualizados

Fortaleza – Ceará
Março - 2012

Uma Estratégia de Migração Dinâmica de Máquinas Virtuais para Economia de Energia em Ambientes Computacionais Virtualizados

Autor:
Deborah Maria Vieira Magalhães

Orientador:
Prof. Danielo Gonçalves Gomes, Dr.

Coorientador:
Prof. José Marques Soares, Dr.

Dissertação de Mestrado apresentada à Coordenação do Curso de Pós-Graduação em Engenharia de Teleinformática da Universidade Federal do Ceará como parte dos requisitos para obtenção do grau de **Mestre em Engenharia de Teleinformática**.

Fortaleza – Ceará
Março - 2012

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Biblioteca de Pós-Graduação em Engenharia - BPGE

M165e Magalhães, Deborah Maria Vieira.
Uma Estratégia de migração dinâmica de máquinas virtuais para economia de energia em ambientes computacionais virtualizados / Deborah Maria Vieira Magalhães. – 2012.
111 f. : il.

Dissertação (mestrado) – Universidade Federal do Ceará, Centro de Tecnologia, Departamento de Engenharia de Teleinformática, Programa de Pós-Graduação em Engenharia de Teleinformática, Fortaleza, 2012.
Área de concentração: Sinais e Sistemas.
Orientação: Prof. Dr. Danielo Gonçalves Gomes.
Coorientação: Prof. Dr. José Marques Soares.

1. Teleinformática. 2. Alocação de recursos. I. Título.

CDD 621.38

DEBORAH MARIA VIEIRA MAGALHÃES

UMA ESTRATÉGIA DE MIGRAÇÃO DINÂMICA DE MÁQUINAS VIRTUAIS PARA ECONOMIA DE ENERGIA EM AMBIENTES COMPUTACIONAIS VIRTUALIZADOS

Dissertação submetida à Coordenação do Programa de Pós-Graduação em Engenharia de Teleinformática, da Universidade Federal do Ceará, como requisito parcial para a obtenção do grau de Mestre em Engenharia de Teleinformática.
Área de concentração: Eletromagnetismo Aplicado.

Aprovada em 01/03/2012.

BANCA EXAMINADORA

Prof. Dr. Danielo Gonçalves Gomes (Orientador)
Universidade Federal do Ceará - UFC

Prof. Dr. José Marques Soares
Universidade Federal do Ceará - UFC

Prof. Dr. José Neuman de Souza
Universidade Federal do Ceará - UFC

Prof. Dr. Bruno Richard Schulze
Laboratório Nacional de Computação Científica - LNCC

Resumo

Em clusters e data centers virtualizados, os recursos precisam ser gerenciados com eficácia na busca de um trade-off entre a garantia de um atendimento satisfatório à demanda por Qualidade de Serviço (QoS) e a redução dos custos operacionais por parte dos provedores. Este trabalho propõe uma estratégia para alocação dinâmica de recursos em ambientes computacionais virtualizados com vistas à redução do consumo de energia, sem promover sobrecargas que podem comprometer o desempenho dos serviços ofertados. Os algoritmos propostos, baseados em heurísticas clássicas, realizam migração de máquinas virtuais entre servidores distintos conforme variação na demanda por recursos. Estes algoritmos foram verificados e validados por medições em um ambiente real composto por servidores virtualizados heterogêneos. O desempenho da proposta é avaliado em quatro cenários distintos a partir das métricas utilização de CPU, utilização de memória, número de migrações e consumo de energia. Os resultados mostraram que os algoritmos responsáveis pela consolidação e distribuição das máquinas virtuais são capazes de reduzir o consumo de energia e dissipar os pontos de ócio e sobrecarga do ambiente.

Palavras-chave: Alocação Dinâmica de Recursos, Virtualização, Migração de Máquinas Virtuais, Economia de Energia, Gerenciamento de Recursos.

Abstract

In clusters and virtualized data centers, the resources must be managed effectively by maximising the SLA fulfilment while minimising the cost. This work proposes a strategy for dynamic resource allocation in virtualized computing environments in order to reduce energy consumption without compromising performance requirements concerning availability and SLA violation. The proposed algorithms, based on classical heuristics, perform virtual machines migration between distinct hosts according to the variation in resources demand. These algorithms were evaluated from measurements in a real environment composed by heterogeneous virtualized hosts. We evaluate their performance in four different scenarios based on the CPU and memory utilization, number of migrations and energy consumption. In general, the results show that the algorithms responsible for consolidation and distribution of virtual machines between hosts are able to reduce energy consumption and dissipate the idle and overload points.

Keywords: Dynamic Resource Allocation, Virtualization, Virtual Machine Migration, Energy-aware Computing, Resource Management.

Dedico esta dissertação
aos meus pais e amigos.

Agradecimentos

Aos meus pais, Olga e Sérgio, que me ensinaram o valor da disciplina e do comprometimento e contribuíram para a construção de quem sou hoje. Agradeço em especial Tiago Vieira, pela compreensão e amizade que o diferencia como pessoa e que me faz querer pôr a descoberto o melhor de mim. Agradeço, ainda, pelo amor, carinho, dedicação, paciência, apoio e conselhos que foram compartilhados comigo durante nosso caminhar.

Aos amigos que me acompanharam, desde a graduação. Agradeço ao meu amigo Filipe, pelo apoio, dedicação e por estar sempre ao meu lado nos momentos difíceis. Às minhas amigas Sandra e Andrea pelo convívio farto de reflexões, comemorações e risadas, e, principalmente, pela amizade.

Ao meu orientador, professor Danielo, pela confiança em mim depositada. Agradeço, particularmente, a paciência dedicada durante esses anos de formação acadêmica. Agradeço também pelos desafios que me foram colocados. Eles me ajudaram a amadurecer pessoal e profissionalmente, tornando-me mais independente.

Aos professores Neuman e Bruno Schulze pela disponibilidade em participar da banca examinadora. Suas críticas e sugestões certamente contribuirão para melhoria deste trabalho.

Aos professores Marques e Cidcley, pelas sugestões e críticas valiosas que muito contribuiram para o amadurecimento deste trabalho. Ao professor Serra, pelo apoio e conselhos nos momentos decisivos.

Aos colegas, Thiago Sá e Mauro Gadelha, pela parceria que rendeu discussões

úteis e muito trabalho, contribuindo de forma determinante para esta dissertação.

Agradeço aos professores e funcionários do PPGETI, pelo conhecimento compartilhado e suporte oferecido. Agradeço, ainda, aos servidores do GREat, pela atenção e apoio às minhas atividades.

Por fim, agradeço à CAPES pelo auxílio financeiro que me permitiu dedicação integral ao mestrado e à minha pesquisa.

"A vida virtuosa é aquela inspirada pelo amor e guiada pelo conhecimento"

Bertrand Russell

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Siglas

API	*Application Programming Interface*
BF	*Best-Fit*
BFM	*Best-Fit Modified*
CPU	*Central Processing Unit*
DCD	*Dynamic Component Deactivation*
DPM	*Dynamic Power Management*
DPS	*Dynamic Performance Scaling*
DVFS	*Dynamic Voltage and Frequency Scaling*
FF	*First-Fit*
FFD	*First-Fit Decreasing*
GNU	*GNU is Not Unix*
GPL	*General Public License*
IaaS	*Infrastructure as a Service*
MTM	*Menor Tempo de Migração*
MVC	*Máximo Volume de Carga*
NFS	*Network File System*

SLA	*Service Level Agreement*
SO	Sistema Operacional
SPM	*Static Power Management*
TIC	Tecnologia da Informação e Comunicação
VM	*Virtual Machine*
VMM	*Virtual Machine Monitor*
WF	*Worst-Fit*
WFM	*Worst-Fit Modified*
XCP	*Xen Cloud Platform*

Capítulo 1

Introdução

Esta dissertação apresenta uma estratégia de alocação dinâmica de recursos capaz de identificar e mitigar pontos de sobrecarga e/ou ócio presentes no ambiente computacional virtualizado. Tal ambiente consiste num conjunto de servidores físicos heterogêneos, que fazem uso de tecnologia de virtualização para permitir a execução de múltiplas máquinas virtuais em um único servidor físico. O objetivo principal da proposta é o uso efetivo dos recursos a fim de economizar energia sem comprometer os requisitos de desempenho. Para tanto, os algoritmos que compõem essa estratégia, baseados em heurísticas clássicas, realizam consolidação e distribuição de máquinas virtuais entre servidores distintos conforme variação na demanda por recursos.

Na Seção 1.1, é apresentado o contexto em que a virtualização tornou-se uma importante ferramenta para o gerenciamento de recursos em ambientes computacionais em larga escala. Na Seção 1.2 são mostrados os problemas encontrados em tais ambientes que impulsionaram o desenvolvimento da solução proposta. Em seguida, a Seção 1.3 expõe os principais objetivos desta dissertação, sendo o método utilizado para alcançá-los discutida na Seção 1.4. E, finalmente, na Seção 1.5, é apresentada a estrutura dos capítulos desta dissertação.

1.1 Contextualização

Tradicionalmente, o foco principal da indústria tem sido na melhoria do desempenho de sistemas computacionais, através de projetos mais eficientes e aumento da densidade dos seus componentes. O aumento do poder computacional

de tais sistemas associado ao crescimento exponencial do tamanho dos dados em simulação/instrumentação científica, armazenamento e publicações na Internet impulsionaram o investimento dos grandes provedores em *data centers*, cada vez mais robustos, para hospedar aplicações que vão desde redes sociais até *workflows* científicos (ALENCAR *et al.*, 2011). Nesse contexto, a Computação em Nuvem despontou no cenário da Tecnologia da Informação e Comunicação (TIC) como uma solução interessante para problemas de processamento e de armazenamento despertando, assim, o interesse de governos, laboratórios de pesquisa e, principalmente, dos gigantes de TIC (e.g. Google, Amazon, IBM, Microsoft, HP) (BUYYA *et al.*, 2009).

Uma vez que tais ambientes computacionais em larga escala necessitam de uma infraestrutura dinâmica, capaz de suportar a natureza elástica da alocação de recursos conforme a demanda dos usuários, as tecnologias de virtualização têm sido uma alternativa amplamente adotada devido a sua capacidade de isolamento, consolidação e migração da carga de trabalho (VOORSLUYS *et al.*, 2009). Através da virtualização, as cargas de trabalho podem ser encapsuladas em máquinas virtuais (VMs) e transferidas para um servidor físico mais adequado, conforme aumento ou redução da demanda por recursos. Essa migração de máquinas virtuais entre servidores permite aumentar a flexibilidade e reduzir a complexidade de gerenciamento dos recursos através de políticas de balanceamento de carga, tolerância a falhas e economia de energia.

1.2 Motivação

Ambientes computacionais consomem grandes quantidades de energia implicando em problemas de ordem monetária, de desempenho e ambiental. A principal razão da ineficiência energética em *data centers* é a baixa utilização média dos recursos (BELOGLAZOV *et al.*, 2011). Conforme dito na seção anterior, esses ambientes necessitam de uma infraestrutura computacional capaz de lidar com a natureza dinâmica da demanda por recursos. Assim, quando a demanda diminui, os servidores físicos ficam subutilizados ou ociosos e quando excede a capacidade de provisionamento do sistema, ocorrem sobrecargas que comprometem o desempenho dos serviços ofertados, implicando em violações de SLA (*Service Level Agreement*) ou até mesmo na indisponibilidade dos mesmos. A fim de lidar com estas sobrecargas, o planejamento de capacidade da infraestrutura computacional é construída para

suportar uma carga máxima teórica, que raramente ocorre na prática. Entretanto, o custo necessário ao super provisionamento (*overprovisioning*) é bastante significativo e inclui despesas com arrefecimento, unidade de distribuição de potência, geradores, instalações de fornecimento de energia, fontes de alimentação, entre outras. Todos estes fatores impactam no aumento do custo financeiro total, seja a curto, médio ou longo prazos (compra, manutenção e expansão do ambiente computacional).

Além das despesas com a infraestrutura, há de se considerar os custos com a energia elétrica consumida nos *data centers* durante o período da operação, que é o principal componente dos custos operacionais. Este consumo não diz respeito apenas à energia utilizada pelos componentes do ambiente computacional, mas também ao sistema de refrigeração. Com o crescimento da densidade de componentes do computador, mais calor é dissipado por metro quadrado, impactando negativamente no desempenho do sistema e na vida útil de tais componentes. Outro problema causado pelo crescente consumo de energia são as elevadas emissões de dióxido de carbono (CO2) as quais contribuem para o aquecimento global. Assim, a economia de energia tornou-se um objetivo de primeira ordem para os sistemas de computação modernos. Portanto, é fundamental que os recursos virtualizados sejam gerenciados de forma eficaz de modo a atender às expectativas de QoS dos usuários e, ainda, reduzir os custos operacionais dos provedores e o impacto sofrido pelos usuários.

Diante disso, várias abordagens têm sido utilizadas buscando a economia de energia, tais como: melhoria dos algoritmos das aplicações, *hardware* energeticamente eficiente e DVFS (*Dynamic Voltage and Frequency Scaling*) (SEMERARO *et al.*, 2002). Outros trabalhos recentes propuseram a virtualização como alternativa comumente adotada em *clusters* e *data centers*, mais especificamente mostrando a viabilidade da migração de máquinas virtuais para a eliminação rápida de pontos de sobrecarga/ócio nestes ambientes (BELOGLAZOV; BUYYA, 2010; PETRUCCI; LOQUES; MOSSÉ, 2010; WOOD *et al.*, 2009; LIAO; HU; JIN, 2010). Todavia, o processo migratório provoca uma degradação no desempenho das aplicações executadas nas máquinas virtuais, degradação esta que também deve ser levada em consideração (CLARK *et al.*, 2005; MAGALHÃES; SOARES; GOMES, 2011).

Nesta dissertação, é proposta uma solução para estes desafios de forma conjunta. Assim, os recursos são gerenciados de modo a economizar energia sem desprezar as sobrecargas que comprometem o desempenho dos serviços ofertados podendo

provocar violações de SLA. Além disso, a solução busca reduzir o impacto negativo promovido pelas migrações de máquinas virtuais no desempenho do sistema.

1.3 Objetivos

1.3.1 Objetivo geral

O principal objetivo deste trabalho é desenvolver uma estratégia para alocação dinâmica de recursos em ambientes computacionais virtualizados com vistas à redução do consumo de energia, sem promover sobrecargas que podem comprometer requisitos de desempenho relativos à disponibilidade e quebra de SLA dos serviços ofertados.

1.3.2 Objetivos específicos

Para cumprir o objetivo geral deste trabalho, os seguintes objetivos específicos devem ser alcançados:

i. levar em consideração o *trade-off* entre economia de energia e desempenho das aplicações que executam no ambiente;

ii. determinar quais VMs, quando e para onde migrá-las de modo a minimizar o consumo de energia ou distribuir a carga do sistema;

iii. minimizar o impacto do processo de migração exercido no desempenho dos serviços a fim de não comprometer totalmente a capacidade de resposta e disponibilidade dos aplicativos.

1.4 Método

O método empregado na concepção e desenvolvimento da solução proposta nesta dissertação pode ser resumida como segue.

i. Revisão Bibliográfica e Caracterização do Problema (Capítulo 2)
Para delineamento do problema a ser resolvido, primeiro, realizaram-se estudos abordando as tecnologias de virtualização, conceitos de Computação em Nuvem, elementos necessários para montar um ambiente de experimentação e ferramentas que suportam a simulação de uma infraestrutura computacional

virtualizada. Em seguida, foi efetuada uma revisão bibliográfica a respeito das principais técnicas de migração de máquinas virtuais, abordagens para economia de energia em *clusters* e *data centers* virtualizados, algoritmos utilizados para alocação dinâmica de recursos virtuais e estratégias para minimizar o *overhead* gerado pela migração.

ii. Concepção e Desenvolvimento da Proposta (Capítulo 3)
Após a configuração do ambiente de experimentação, definimos os principais módulos que compõem a arquitetura da solução, bem como o conjunto de políticas e heurísticas que atuam em cada módulo: 1) Módulo de monitoramento, conta com as políticas de limiar simples e duplo para identificar pontos de sobrecarga/ócio no sistema; e 2) Módulo de Mitigação, faz uso das políticas de seleção de VMs e heurísticas de alocação de recursos, para determinar quais VMs devem ser migradas e para onde migrá-las, respectivamente. Em seguida, concebemos os pseudocódigos dos algoritmos que formam a solução e, por fim, implementamos a proposta no ambiente de experimentação.

iii. Configuração do Ambiente de Experimentação (Seção 4.1)
Para o desenvolvimento da proposta, foram configurados 5 servidores físicos heterogêneos de acordo com 3 papéis desempenhados no ambiente de experimentação: 1) Frontend, responsável por gerenciar o conjunto de recursos; 2) Servidores XCP (*Xen Cloud Platform*) 1.1 (XCP, 2011), onde serão instanciadas as VMs e suas respectivas aplicações; e 3) Servidor NFS (*Network File System*), onde residem as imagens das VMs e os discos virtuais. Foi estudada a API (*Application Programming Interface*) do hipervisor XCP utilizada na implementação da estratégia proposta.

iv. Análise dos Resultados (Seções 4.2)
Esta etapa contou com a definição das métricas e parâmetros relevantes para avaliação, cenários de experimentação e cargas de trabalho utilizadas. De posse dessas informações, os testes foram efetuados e os resultados analisados.

1.5 Contribuições

A principal contribuição deste trabalho é a concepção de uma estratégia de alocação dinâmica de VMs que possibilita ambientes computacionais virtualizados,

como *clusters* e *data centers*, lidar com a natureza elástica da alocação dos recursos conforme a demanda dos usuários, considerando o balanceamento entre os dois dos principais problemas encontrados em tais ambientes. Estes problemas são a necessidade de economizar energia e o combate às sobrecargas que comprometem o desempenho dos serviços ofertados, podendo provocar violações de SLA.

Outras contribuições desta dissertação:

- A maior parte dos trabalhos presentes na literatura considera apenas a CPU nas decisões de realocação das VMs a fim de economizar energia. No entanto, outros recursos como memória, interfaces de rede e discos, também contribuem para o consumo total de energia. Em nossa estratégia, as decisões de realocação são tomadas com base no volume de carga que capta o grau de sobrecarga/ócio ao longo de múltiplas dimensões, levando em consideração memória e processador;
- Neste trabalho, consideramos o impacto do processo migratório, tanto quando definimos uma política de alocação de VMs para reduzir o número de migrações como também quando definimos uma política de seleção de VMs que minimiza o tempo total de migração;
- Avaliação do impacto das abordagens stop-and-copy e pré-cópia. Estas duas técnicas de migração de máquinas virtuais, amplamente referenciadas na literatura e disponibilizadas pelo hipervisor Xen, foram analisadas em um ambiente de experimentação real no trabalho (MAGALHÃES; SOARES; GOMES, 2011);
- Modificações em heurísticas clássicas. Estas modificações permitem que os algoritmos da nossa estratégia levem em conta o custo energético de um hospedeiro no momento da definição do par origem/destino envolvidos no processo de realocação das máquinas virtuais;

1.6 Estrutura do documento

No Capítulo 2 é realizado um apanhado dos conceitos, técnicas e desafios relevantes ao contexto da economia de energia em ambiente computacionais virtualizados, tais como, *clusters*, *data centers* e Nuvens. Apresentamos uma taxonomia para o gerenciamento eficiente de energia em sistemas computacionais,

dando enfoque às técnicas de gerenciamento de energia dinâmicas aplicadas em múltiplos servidores, *data centers* e Nuvens. Discutimos trabalhos que se propõem a gerenciar recursos computacionais virtualizados de modo a economizar energia e/ou dissipar sobrecargas que comprometem o desempenho do sistema. Também elencamos as principais abordagens de migração de máquinas virtuais, destacando suas vantagens e impactos no desempenho do sistema no qual são empregadas. E, por fim, com base nos apresentamos como o problema de alocação de VMs em servidores físicos pode ser caracterizado, destacando suas principais variáveis e o modelo de potência utilizado pela nossa proposta para determinar o consumo de energia de um servidor.

No Capítulo 3, apresentamos uma visão geral da estratégia de alocação dinâmica de VMs proposta, com seus componentes principais e respectivas funcionalidades. Em seguida, descrevemos as heurísticas e políticas utilizadas pela mesma. Detalhamos a abordagem de monitoramento e classificação dos servidores. Por fim, mostramos como os algoritmos de consolidação e distribuição atuam no ambiente para economizar energia sem desprezar as sobrecargas que podem comprometer o desempenho do sistema.

No Capítulo 4, definimos as métricas utilizadas para avaliar o desempenho da proposta e das políticas e heurísticas utilizadas. Em seguida, apresentamos em detalhes o ambiente de experimentação utilizado para realizar os testes de validação da mesma. Finalmente, descrevemos experimentos realizados com medições reais.

No Capítulo 5, fazemos uma síntese das principais conclusões, e contribuições, apontando perspectivas de trabalhos futuros. No Apêndice A, são encontrados todos os pseudocódigos utilizados no desenvolvimento da proposta.

Capítulo 2

Revisão Bibliográfica e Caracterização do Problema

Neste capítulo realizamos uma revisão de literatura para enfatizar conceitos, técnicas e desafios relevantes ao contexto da economia de energia em ambiente computacionais virtualizados, tais como, *clusters*, *data centers* e Nuvens. Na Seção 2.1, discutimos sobre uma taxonomia para o gerenciamento eficiente de energia em sistemas computacionais, dando enfoque às técnicas dinâmicas de gerenciamento de energia aplicadas em múltiplos servidores, *data centers* e Nuvens. Visto que nossa estratégia de alocação dinâmica de VMs baseia-se na migração das mesmas, a Seção 2.2 discute as principais abordagens de migração de máquinas virtuais, destacando suas vantagens e impactos no desempenho do sistema no qual são empregadas. E, por fim, com base nos trabalhos correlatos tratados neste capítulo, a Seção 2.3 apresenta como o problema de alocação de VMs em servidores físicos pode ser caracterizado, destacando suas principais variáveis e o modelo de potência utilizado pea nossa proposta para determinar o consumo de energia de um servidor.

2.1 Taxonomia de Gerenciamento de Energia

A eficiência energética emergiu como um dos requisitos mais importantes para sistemas de computação, em particular os de larga escala, tais como *data centers* e Nuvens, uma vez que os mesmos consomem enormes quantidades de energia elétrica, promovendo altos custos operacionais e significativas emissões de dióxido de carbono no ambiente. Diante disso, existem iniciativas da indústria para tornar os recursos

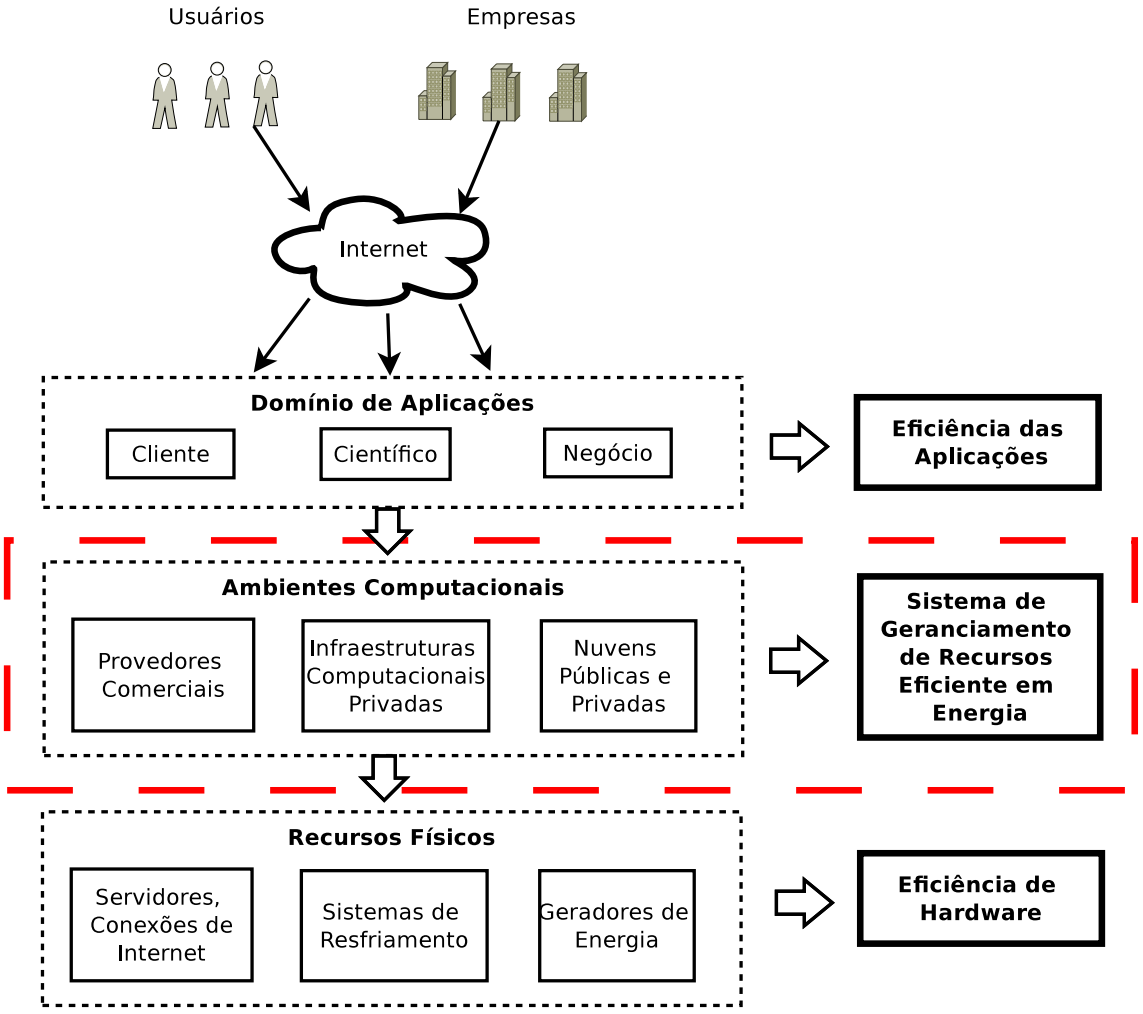


Figura 2.1: Consumo de energia em diferentes níveis dos sistemas computacionais. Adaptado de (BELOGLAZOV *et al.*, 2011)

físicos mais eficientes no que diz respeito ao consumo de energia. No entanto, este consumo não é determinado apenas pela eficiência dos recursos físicos, mas também é dependente do sistema de gerenciamento de recursos implantados na infraestrutura e eficiência dos aplicativos em execução no sistema. A relação entre consumo de energia em diferentes níveis dos sistemas computacionais pode ser vista na Figura 2.1. Conforme em destaque na mesma, este trabalho atua em nível de ambiente computacional.

Beloglazov *et al.* (2011) propuseram uma taxonomia para o gerenciamento eficiente de energia em sistemas computacionais que cobre diversos níveis, como apresentado na Figura 2.2, em que as técnicas de gerenciamento de energia de alto nível são divididas em estáticas e dinâmicas. Do ponto de vista do *hardware*, o SPM (*Static Power Management*) contém todos os métodos de otimização que são

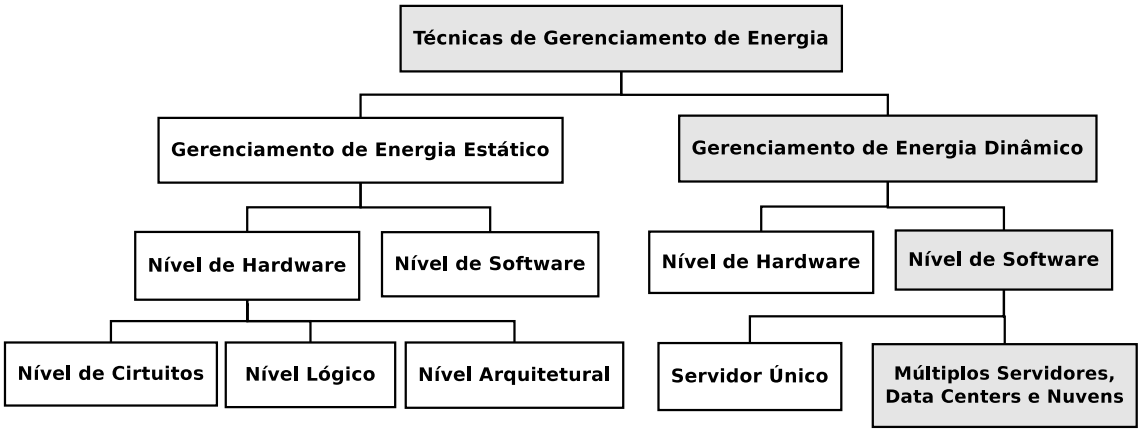


Figura 2.2: Taxonomia para o gerenciamento eficiente de energia em sistemas computacionais. Adaptado de (BELOGLAZOV *et al.*, 2011)

aplicados no momento de projetar os níveis de circuito, lógico, arquitetura e sistema dos sistemas computacionais. Do ponto de vista do *software*, é importante considerar o consumo energético na implementação dos programas que devem executar no sistema pois, mesmo com *hardware* otimizado, *softwares* mau projetados podem levar a perdas de energia.

Esta dissertação foca nas técnicas de Gerenciamento Dinâmico de Energia, DPM (*Dynamic Power Management*) as quais incluem métodos e estratégias para adaptação, em tempo de execução, do comportamento de um sistema de acordo com as necessidades de recursos atuais ou qualquer outra característica dinâmica do estado do sistema. Ainda de acordo com a Figura 2.2, as técnicas DPM pode ser distinguidas pelo nível em que são aplicadas: *hardware* ou *software*. O problema do gerenciamento eficiente de recursos em termos energéticos tem sido investigado em diferentes contextos, tais como: gerenciamento através de Sistema Operacional (SO) de servidores virtualizados e não virtualizados, seguido por sistemas de múltiplos nós, entre eles, *clusters* homogêneos e heterogêneos, *data centers* e Nuvens.

2.1.1 *Dynamic Voltage and Frequency Scaling* (DVFS)

As técnicas DPM aplicadas em nível de *hardware* podem ser classificadas em DCD (*Dynamic Component Deactivation*) e DPS (*Dynamic Performance Scaling*). As técnicas DCD são aplicadas em componentes do computador que não suportam escalonamento de desempenho, ou seja, esses componentes somente podem ser desativados em períodos de inatividade. Em contraste, as técnicas DPS são aplicadas em componentes que suportam ajuste dinâmico de seu desempenho como, por

exemplo a CPU (*Central Processing Unit*), a qual permite ajustes graduais em sua frequência de *clock* juntamente com o ajuste da tensão de alimentação. Desse modo, tais componentes podem assumir estados ativos de baixa potência em períodos de ócio, evitando o *overhead* provocado pela mudança entre os estados ativo e inativo.

A técnica DVFS cria uma ampla faixa dinâmica de potência para a CPU, permitindo modos ativos de baixa potência. Essa flexibilidade levou à sua ampla adoção e surgimento de muitas políticas que escalam o desempenho da CPU de acordo com os requerimentos atuais do sistema (PALLIPADI; STARIKOVSKIY, 2006). Posteriormente, essa técnica foi extrapolada para sistemas de múltiplos servidores provendo um bom desempenho energético entre eles (ELNOZAHY; KISTLER; RAJAMONY, 2003).

Verma, Ahuja e Neogi (2008) propuseram um controlador de alocação dinâmica de aplicações que minimiza o consumo de energia e o custo de migração, com garantias de desempenho em clusters virtualizados. Essa solução é composta por três gerentes e um árbitro, que coordena as ações e toma decisões de alocação. O gerente de desempenho monitora o comportamento das aplicações e redimensiona VMs de acordo com as necessidades de recursos atuais e do SLA. O gerente de energia se encarrega de ajustar os estados de energia do *hardware* através de DVFS. O gerente de migração realiza as instruções para a migração de máquinas virtuais, a fim de consolidar a carga de trabalho. Por fim, o árbitro tem uma visão global do sistema e toma decisões sobre novos posicionamentos de VMs e determina quais VMs e para quais nós devem ser migradas.

Ainda em (VERMA; AHUJA; NEOGI, 2008), o problema de alocação das VMs em servidores físicos é tratado como um problema de empacotamento binário, com caixas de diferentes valores e tamanhos. Para solucionar o problema, eles adaptaram o algoritmo *First-Fit Decreasing* (FFD) para trabalhar para caixas de tamanhos diferentes e itens que dependem de uma função custo. O problema foi dividido em dois sub-problemas: na primeira parte, os valores de nova utilização são determinados para cada servidor com base nas funções de custo e desempenho exigidos; na segunda parte, as VMs são hospedades em servidores até alcançar taxa de utilização objetivo. A cada período de tempo, o posicionamento VM deve ser otimizado para minimizar o consumo de energia e maximizar o desempenho. Os autores partem de várias hipóteses para resolver o problema, tais como: i) isolamento de desempenho, significa que uma VM pode ser vista por um aplicativo em execução

como um servidor físico dedicado com características iguais aos parâmetros de própria VM; ii) a duração de uma migração ao vivo de máquinas virtuais não depende da carga que executa em *background* e o custo de migração pode ser estimado a priori com base no tamanho VM; e iii) o algoritmo de minimização de energia pode minimizar o consumo sem saber a quantidade real de potência consumida pela aplicação. No entanto, em (CLARK *et al.*, 2005), experimentos mostram que o tempo total de migração e, consequentemente, seu custo dependem do tipo de carga que está sendo executada na VM. Nesta dissertação, realizamos testes a fim de analisar o impacto da migração de máquinas e os resultados encontrados corroboram essa dependência (MAGALHÃES; SOARES; GOMES, 2011).

Petrucci, Loques e Mossé (2010) propuseram uma solução de otimização para gerenciamento de energia e desempenho em *clusters* virtualizados. O problema tratado consiste em selecionar a configuração mais eficiente em energia em tempo de execução, mapeando as múltiplas aplicações executando nas máquinas virtuais para os servidores físicos. A decisão de otimização inclui selecionar a melhor combinação frequência/tensão elétrica para cada servidor físico. A estratégia proposta possibilita o sistema de servidores virtualizados reagir a variações na carga e adaptar a configuração de acordo com as mesmas. Apesar da proposta garantir soluções ótimas, ela se torna crítica principalmente em ambientes computacionais em larga escala, onde há constante variação na demanda por recursos. Em resumo, DVFS pode proporcionar uma substancial economia de energia, no entanto, deve ser aplicada com critério, pois o resultado pode variar significativamente para diferentes arquiteturas de sistemas de *hardware* e *software*.

Neste trabalho também utilizamos a migração de máquinas virtuais a fim de desligar os hospedeiros ociosos. Todavia, não fazemos uso da técnica DVFS. É importante destacar que reduzir a potência do processador enquanto ele encontra-se executando uma tarefa não necessariamente trará economia de energia, uma vez que o mesmo poderá demorar mais tempo para executá-la. A DVFS reduz o número de instruções que um processador pode entregar em um determinado período de tempo, reduzindo assim seu desempenho. Este, por sua vez, aumenta o tempo de execução para os segmentos de programa que são orientados a CPU. Logo, as estratégias que visam economizar energia modificando a frequência e/ou tensão de alimentação das CPUs devem garantir que uma tarefa não utilize grande parte do poder de

processamento. Além disso, DVFS não se aplica a outros recursos computacionais de um sistema além da CPU. Detalhes sobre: (i) problema de alocação dinâmica de VMs em servidores físicos serão discutidos na Seção 2.3; (ii) as abordagens de migração de VMs e seus custos associados serão apresentados na Seção 2.2.

2.1.2 Virtualização

Assim como a DVFS contribui com as técnicas DPM aplicadas em nível de *hardware*, a virtualização de recursos computacionais pode ser utilizada com as técnicas DPM aplicadas em nivel de *software* no gerenciamento de recursos visando reduzir o consumo de energia. O conceito de virtualização originou-se com os sistemas operacionais *mainframe* da IBM na década de 1960. Atualmente, várias empresas e projetos de código aberto oferecem pacotes de *software* para permitir uma transição para a computação virtual. A Intel Corporation e a AMD têm desenvolvido melhorias de virtualização proprietárias sobre o conjunto de instruções x86 em cada uma de suas linhas de produtos de CPU, a fim de facilitar a computação virtualizada (BELOGLAZOV *et al.*, 2011).

A ampla adoção da virtualização em ambientes computacionais modernos, tais como *data centers* e Nuvens, ocorreu principalmente devido a sua capacidade de isolamento e migração da carga de trabalho (VOORSLUYS *et al.*, 2009). As tecnologias de virtualização permitem criar diversas VMs em um único servidor físico e, portanto, reduzir a quantidade de *hardware* em uso e melhorar a utilização dos recursos. Essas VMs estão isoladas no que diz respeito ao desempenho, ou seja, se um usuário promover uma falha em sua VM, essa falha não impacta no desempenho das outras VMs contidas no mesmo servidor físico. A virtualização também permite deslocar, ou seja, migrar as VMs de um servidor físico para outro mais adequado no sistema, conforme a demanda por recursos varia ao longo do tempo.

Esta migração de VMs através de servidores físicos distintos traz diversas vantagens, tais como: (i) balanceamento de carga - as VMs podem ser rearranjadas em nós físicos do ambiente computacional para aliviar a carga sobre hospedeiros saturados, (ii) manutenção online e tolerância a falhas - uma máquina física pode precisar de *upgrade* ou manutenção do sistema devido a falhas, de forma que um administrador pode migrar as VMs em execução para uma máquina alternativa, liberando a máquina original para manutenção e, (iii) gerenciamento de energia - quando as VMs podem ser consolidadas em um número reduzido de servidores,

enquanto nós ociosos podem ser desligados reduzindo assim o custo de energia. Tais vantagens permitem aumentar a flexibilidade e reduzir a complexidade de gerenciamento dos recursos físicos em ambientes virtualizados. Todavia, a migração provoca uma degradação no desempenho do sistema uma vez que para realizá-la é necessário parar a execução da VM, provocando sua indisponibilidade e, por conseguinte, das aplicações que nela são executadas. Além disso, há um consumo extra de ciclos de CPU e de largura de banda no processo de transferência da VM de um hospedeiro para outro, impactando nos recursos disponibilizados para os serviços providos pela mesma. Nesse contexto, uma melhor compreensão do impacto da migração no desempenho do sistema é fundamental para concepção de estratégias aptas a utilizar os recursos de maneira eficiente sem comprometer a capacidade de resposta e disponibilidade dos serviços ofertados. Uma discussão mais ampla a esse respeito é realizada na Seção 2.2.

Essa possibilidade de remapeamento de recursos físicos em servidores virtuais a fim de lidar com carga de trabalho dinâmica consiste em uma das principais vantagens da virtualização. Isso é especialmente útil em Computação em Nuvem (MELL; GRANCE, 2011), uma vez que tais ambientes necessitam de uma infraestrutura computacional dinâmica, capaz de suportar a natureza elástica da alocação de recursos conforme a demanda dos usuários. O paradigma de Computação em Nuvem apoia-se fortemente no uso da virtualização e oferece a capacidade de fornecimento de recursos sob demanda em um modelo no qual os usuários pagam somente pelos recursos que consomem. Assim, as empresas podem terceirizar suas necessidades de computação para a Nuvem, eliminando a necessidade de manter uma infraestrutura computacional própria. A utilização da virtualização confere à Computação em Nuvem as seguintes características: i) melhor utilização dos recursos; ii) independência de localização - as VMs pode ser movidas para um lugar onde a energia é mais barato, de forma transparente para o usuário; iii) redimensionamento de VMs - o uso de recursos podem ser ajustados às exigências atuais do sistema e, por fim, iv) gestão eficiente dos recursos por parte do provedor da Nuvem.

Apesar das diversas vantagens inerentes à utilização das tecnologias de virtualização, a consolidação agressiva de máquinas virtuais e a variabilidade da carga de trabalho de algumas VMs podem provocar perdas de desempenho em termos do incremento do tempo de resposta, *time outs* ou falhas. Portanto, os

provedores de Nuvem tem de lidar com o *trade-off* entre economia de energia e desempenho.

2.1.2.1 Tecnologias de Virtualização

A camada de *software* que oferece a virtualização é chamada VMM (*Virtual Machine Monitor*) ou hipervisor e fica acima dos recursos da máquina física, conforme ilustrado na Figura 2.3. O hipervisor virtualiza tais recursos permitindo, assim, a execução de múltiplas máquinas virtuais, onde cada uma delas executa seu próprio SO, ou seja, o VMM permite a execução de vários sistemas operacionais, simultaneamente, sobre um mesmo *hardware*. Essa característica é extremamente útil para provedores de Computação em Nuvem que fornecem IaaS (*Infrastructure as a Service*). Assim, os usuários podem instanciar sua própria VM e definir o SO e aplicações que desejarem. Isso torna o serviço muito mais flexível e personalizável. O SO que executa dentro de uma VM é referido como um sistema operacional convidado. O VMM normalmente mapeia o estado de suas VMs em um sistema de arquivos local do hospedeiro físico. Quando uma VM é suspensa os arquivos correspondentes são atualizados. Se tais arquivos são copiados para um hospedeiro remoto com arquitetura de *hardware* semelhante, um VMM presente nesse hospedeiro pode retomar a execução da VM, ocorrendo, desse modo, sua migração.

Dentre as tecnologias de virtualização mais populares, temos: Xen Hypervisor (BARHAM *et al.*, 2003), VMware Solutions (VMWARE..., 2011) e KVM (KVM, 2011). Nesta dissertação, discutiremos o Xen, tecnologia na qual a solução proposta baseia-se. O Xen é uma camada de *software* rodando diretamente no *hardware* do computador que substitui o SO, permitindo assim, que o *hardware* do computador execute vários SOs convidados simultaneamente. O suporte aos processadores x86, x86-64, Itanium, Power PC, e ARM permitem ao Xen executar uma ampla variedade de dispositivos de computação e SOs, entre eles: Linux, NetBSD, FreeBSD, Solaris, Windows e outros sistemas operacionais como convidados executando sobre o hipervisor. A comunidade Xen.org desenvolve e mantém o Xen Hypervisor como uma solução livre licenciado sob a GNU (*GNU is Not Unix*) GPL (*General Public License*) e atualmente suporta vários projetos entre eles: o XCP que consiste na plataforma de Computação em Nuvem provida pelo Xen. Sua versão atual, v1.0, foi derivada do XenServer 5.6. No entanto, ele é totalmente gratuito e *open source*. O XCP atende às necessidades dos provedores de Nuvem, hospedagem de serviços e

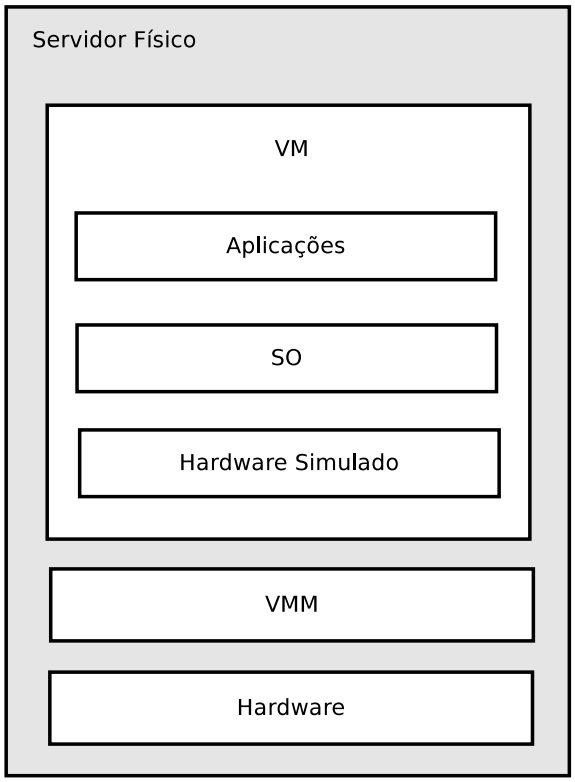


Figura 2.3: Arquitetura Virtualizada

data centers. Ele consolida as cargas de trabalho do servidor, permitindo economia de energia, contribuindo assim para computação ecologicamente sustentável.

O computador que executa o hipervisor Xen contém três componentes (SPECTOR, 2010): i) o próprio hipervisor Xen - é executado diretamente sobre o *hardware* e torna-se a *interface* para todas as solicitações de *hardware* (CPU, I/O e disco) dos sistemas operacionais convidados. Ao separar os SOs convidados do *hardware*, o Xen é capaz de executar vários sistemas operacionais de forma segura e independente; ii) Domínio 0, domínio privilegiado (Dom0) - cliente privilegiado em execução no hipervisor com acesso direto ao *hardware*. Este domínio é lançado pelo hipervisor durante a inicialização do sistema e possui é privilégios que lhe permitem administrar todos os aspectos dos domínios convidados como, por exemplo, iniciar/parar pedidos I/O; e iii) Múltiplos Domínios U, domínio convidado desprivilegiado (DomU) - cliente em execução hipervisor que não possui acesso direto ao *hardware*(memória, disco, etc). Esses componentes são ilustrados na Figura 2.4.

O Xen suporta as aborgens de migração de máquinas virtuais *live* e *non-live*, que podem ser aproveitadas por algoritmos eficientes em energia para consolidação

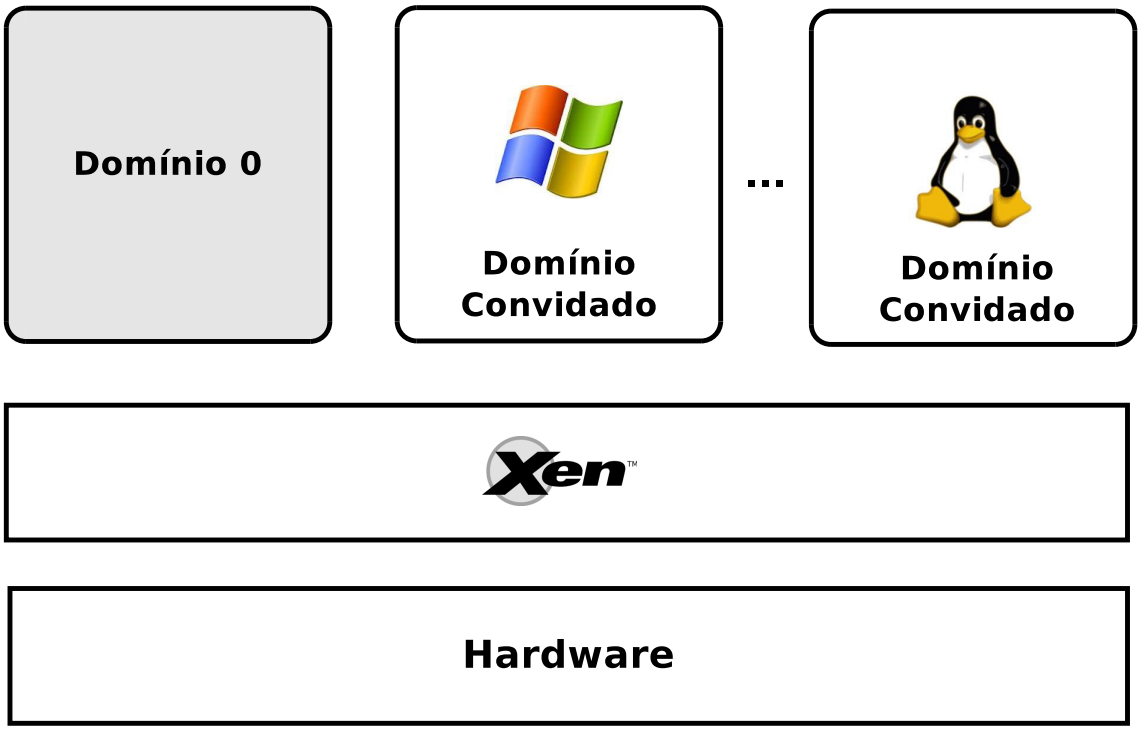


Figura 2.4: Hipervisor Xen. Adaptado de (SPECTOR, 2010)

dinâmica de VMs. A migração é utilizada para transferir uma VM entre servidores físicos. Para executar uma migração, ambos os hospedeiros envolvidos devem estar executando o Xen e o servidor de destino deve ter recursos suficientes (por exemplo, capacidade de memória) para acomodar a VM após a migração. Uma vez que o processo migratório é finalizado, uma nova instância da VM é iniciada no hospedeiro de destino. Para minimizar a sobrecarga de migração, os servidores são ligados a um servidor NFS ou solução de armazenamento similar, que elimina a necessidade de copiar o conteúdo da VM diretamente do disco do hospedeiro de origem. Mais detalhes a respeito desses duas abordagens de migração são apresentados na Seção 2.2.

2.1.3 Múltiplos servidores, *data centers* e Nuvens

Tendo em vista o impacto econômico e ambiental promovido pelo alto consumo de energia em ambientes computacionais em larga escala, diversos trabalhos na literatura propuseram soluções que lidam com o gerenciamento de recursos a fim de economizar energia e minimizar a perda de desempenho a nível de clusters, data centers e Nuvens. Nesta seção, elencamos algumas dessas abordagens propostas recentemente. As características utilizadas para classificar as abordagens são

apresentadas na Figura 2.5, onde as características da solução proposta nesta dissertação estão em destaque.

Quando a demanda por recursos excede a capacidade de provisionamento de um sistema, ocorrem sobrecargas que podem comprometer até mesmo a disponibilidade dos serviços oferecidos. Diante desse desafio, Wood *et al.* (2009) propuseram um sistema chamado Sandpiper, responsável por automatizar as tarefas de detecção e mitigação de sobrecargas, determinando um novo mapeamento de recursos físicos para virtuais, através do redimensionando das máquinas virtuais e/ou migrações. A solução implementa duas abordagens distintas de monitoramento que podem atuar separadas ou em conjunto. A abordagem conhecida como black-box monitora o uso dos recursos (CPU, memória e rede) de cada servidor virtual em um intervalo de medição e gera relatórios com essas estatísticas para tomada de decisões. A vantagem dessa abordagem reside na decisão ser baseada na aferição da utilização dos recursos apenas a partir de observações externas e independente do sistema operacional residente dentro de cada máquina virtual. Entretanto, a abordagem black-box limita-se a uma tomada de decisão reativa e não pró-ativa. Em contrapartida, a abordagem gray-box assume o acesso a uma pequena quantidade de estatísticas a nível de sistema operacional e aplicações que executam no interior da máquina virtual. Apesar de mais intrusiva, esta abordagem permite informar melhor o algoritmo de provisionamento, melhorando potencialmente a qualidade do processo decisório. Os resultados mostraram que a migração de máquinas virtuais é uma técnica viável para a eliminação rápida de sobrecargas em ambientes de datacenter. Em nosso trabalho, utizamos uma abordagem de monitoramento menos intrusiva. Nossa solução não se concentra em aplicações específicas e pode ser aplicado a qualquer tipo de carga de trabalho e SO. Além disso, nossa abordagem pode atuar tanto em Nuvens privadas quanto em Nuvens públicas diferente da abordagem gray-box, que se restringe a Nuvens privadas.

Tarighi, Motamedi e Sharifian (2010) propuseram uma solução cujo objetivo também consiste em equilibrar a carga do ambiente computacional e evitar a inatividade dos serviços disponibilizados. No entanto, esta abordagem faz uso do algoritmo Fuzzy TOPSIS (CHU; LIN, 2003) para realizar a tomada de decisão sobre todos os servidores ativos no datacenter. Visto que a carga de trabalho para cada nó varia ao longo do tempo e pode atingir um ponto de saturação, o algoritmo verifica os servidores físicos para encontrar alguma sobrecarga. Uma vez que um hospedeiro

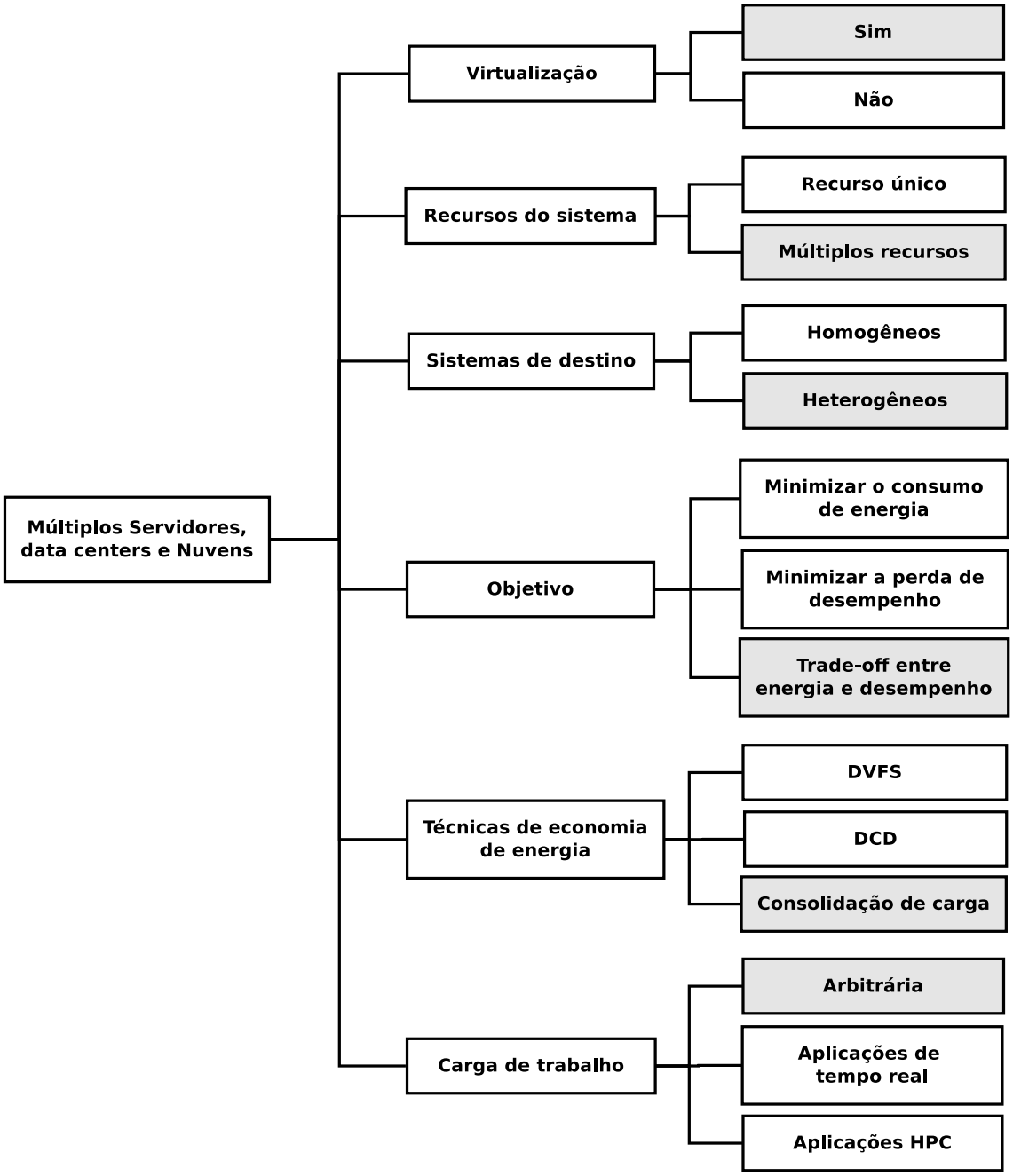


Figura 2.5: Taxonomia das características dos trabalhos correlatos. Adaptado de (BELOGLAZOV *et al.*, 2011)

sobrecarregado seja encontrado, o TOPSIS decide sobre o servidor de destino e inicia a operação automática de migração da VM para o hospedeiro menos sobrecarregado. Apesar de a abordagem proposta utilizar um poderoso algoritmo para tomadas de decisão, a complexidade do mesmo incrementa com o aumento do número de VMs, aumentando o tempo gasto com os cálculos, ou seja, para um número considerável de VMs, a solução pode se tornar inviável.

Os trabalhos anteriormente discutidos na Seção 2.1.1 visam economizar energia por meio de ajustes na frequência e/ou tensão de alimentação das CPUs. Estudos apontam que servidores ociosos apresentam um consumo de energia entre 49% e 78% em relação ao seu estado ativo (BERL *et al.*, 2010) (DAS *et al.*, 2010) (TALEBI; WAY, 2009). Esse fato justifica a estratégia de desligamento dos hospedeiros ociosos, inclusive adotada nesta dissertação, a fim de reduzir o consumo total de energia do sistema.

Assim como em nossa proposta, Liao, Hu e Jin (2010) desenvolveram um estudo que não se limita apenas a mitigar pontos de sobrecarga ou economizar energia. Os autores buscam o equilíbrio entre os estados de sobrecarga e ócio do ambiente computacional, economizando energia e provendo a QoS requisitada pelos usuários através uma nova estratégia, chamada Magnet, que usa a migração ao vivo de máquinas virtuais para transferência de carga entre servidores físicos com o objetivo de economizar energia em ambientes computacionais homogêneos. Assim, os servidores redundantes do sistema são desligados quando o mesmo está em estado de computação não intensiva, de forma a economizar energia, e a carga dos hospedeiros sobrecarregados é transferida para os hospedeiros livres quando o sistema está em estado de computação intensiva, visando obter um maior desempenho.Os autores levaram em consideração apenas a carga para eleger uma VM a participar do processo de migração. Em nosso trabalho, além de levar em consideração o volume de carga da VM, contamos com um política de seleção de VMs capaz de minimizar o impacto da migração no desempenho do sistema.

A fim de minimizar o consumo de energia elétrica, Beloglazov e Buyya (2010) propuseram uma política de gestão eficiente de recursos para ambientes de computação em nuvem, cujo objetivo consiste em consolidar VMs através da migração ao vivo de máquinas virtuais e desligar os hospedeiros ociosos para minimizar o consumo de energia, proporcionando a qualidade dos serviços exigida. A vantagem dessa solução reside na capacidade da proposta lidar com requisitos

rigorosos de QoS e recursos físicos e virtuais heterogêneos. Além disso, os algoritmos são implementados como heurísticas rápidas, pois eles não dependem de um tipo particular de carga de trabalho e não exige qualquer conhecimento sobre as aplicações em execução nas máquinas virtuais. A desvantagem dessa abordagem resume-se nas decisões de realocação não levarem em consideração múltiplos recursos de sistema, como interfaces de rede e discos, uma vez que tais recursos também contribuem para o consumo total de energia. Tais características diferem de nossa estratégia, onde as decisões de realocação são tomadas com base no volume de carga que capta o grau de sobrecarga/ócio ao longo de múltiplas dimensões, levando em consideração memória e processador.

A proposição de novas heurísticas para o problema de consolidação dinâmica de VMs considerando o *trade-off* entre desempenho e economia de energia é apresentada em (BELOGLAZOV; BUYYA, 2011). Ainda como contribuição, os autores conduziram uma análise competitiva de algoritmos para os problemas de migração de uma única VM e consolidação dinâmica de máquinas virtuais. Na identificação de sobrecargas no sistema, os autores levam em consideração apenas o percentual de CPU. Entretanto, em um ambiente de computação em nuvem real, um servidor pode estar com sobrecarga em termos de memória ou rede e ter uma baixa utilização de CPU. Assim, ele não será classificado como sobrecarregado pelo algoritmo, mas a sobrecarga de memória poderá implicar em uma quebra de SLA. Portanto, em nossa proposta consideramos também o percentual de memória utilizada para rotular um servidor físico como sobrecarregado.

Tabela 2.1: Comparação dos Trabalhos Correlatos

Trabalho	**Sistemas de destino**	**Recursos do sistema**	**Objetivo**	**Técnica para economia de energia**
[Verma et al. 2008]	Heterogêneos	CPU	Minimizar o consumo de energia, minimizar a perda de desempenho	DVFS, consolidação de VMs
[Petrucci et al. 2010]	Heterogêneos	CPU	Minimizar o consumo de energia, minimizar a perda de desempenho	DVFS
[Wood et al. 2009]	Homogêneos	CPU, Memória e Rede	Minimizar a perda de desempenho	-
[Tarigui et al 2010]	Heterogêneos	CPU, Memória e Rede	Minimizar a perda de desempenho	-
[Liao, Hu e Jin 2010]	Homogêneos	CPU	Minimizar o consumo de energia, minimizar a perda de desempenho	Consolidação de VMs
[Beloglazov e Buyya 2010]	Heterogêneos	CPU	Minimizar o consumo de energia, minimizar a perda de desempenho	Consolidação de VMs
[Beloglazov e Buyya 2011]	Heterogêneos	CPU	Minimizar o consumo de energia, minimizar a perda de desempenho	Consolidação de VMs

2.2 Abordagens de Migração de VMs

Esta Seção apresenta as principais abordagens de migração de máquinas virtuais, destacando seus aspectos positivos e negativos. Para tanto, utilizamos métricas

que auxiliaram na comparação dos trabalhos, entre elas: (1) *downtime*, refere-se ao tempo durante o qual a execução da VM em migração é interrompida. No mínimo, esse intervalo de tempo inclui a transferência do estado do processador e dispositivos de rede e disco. O *downtime* é uma métrica fundamental para avaliação de desempenho da migração, visto que a inatividade da VM impacta diretamente nos serviços por ela disponibilizados, podendo comprometer a transparência de migração em nível de usuário; (2) tempo total de migração refere-se ao tempo gasto na realização de todas as etapas necessárias ao processo de migração. Essa métrica é de suma importância porque afeta a liberação de recursos em ambos os hospedeiros participantes. A memória ocupada no hospedeiro de origem pela VM fica alocada à migração e só será liberada após a conclusão do processo.

2.2.1 *Non-live Migration (stop-and-copy)*

As abordagens de migração de VMs dividem-se em: non-live migration ou *pure stop-and-copy* e *live migration*. A *non-live migration* (SCHMIDT, 2000; OSMAN *et al.*, 2002) é caracterizada pela suspensão da VM no hospedeiro de origem e, em seguida, suas páginas de memória são copiadas para o hospedeiro de destino, sendo, por fim, executada no mesmo. Essa abordagem possui vantagens em termos de simplicidade e reduz o tempo total de migração em relação à abordagem *live*, uma vez que é necessária apenas uma iteração para a cópia da imagem da VM, sem a necessidade de iterações subsequentes para a transferência das páginas posteriormente atualizadas. Em contrapartida, o downtime sofrido pelas aplicações que executam na VM é maior nesse tipo de abordagem visto que a VM é suspensa durante a transferência de todas as páginas que compõem a memória.

2.2.2 *Live Migration* baseada na Pré-Cópia da Memória

O primeiro trabalho a desenvolver uma estratégia *live migration*, utilizando tecnologia de virtualização Xen, consistiu em fazer uso de um algoritmo de pré-cópia para realizar a migração (CLARK *et al.*, 2005). A etapa de pré-cópia consiste na reconstrução da imagem da memória da VM no hospedeiro de destino antes de sua execução no mesmo. Esta etapa se dá em várias iterações. Na primeira, todas as páginas são transferidas da origem para o destino. Nas iterações subsequentes são copiadas apenas as páginas que foram modificadas durante a fase de transferência anterior. Após a etapa de pré-cópia é dado início a fase *stop-and-copy*, onde a execução da VM no hospedeiro de origem é suspensa e o tráfego de rede juntamente

com o estado da CPU e páginas de memória inconsistentes remanescentes são redirecionadas para o hospedeiro de destino. Por fim, a VM pode ser finalmente executada no hospedeiro de destino.

A vantagem da *live migration* reside no fato da VM e, portanto, todas as aplicações nela hospedadas, continuarem a executar enquanto a imagem da memória é transferida, proporcionando, assim, um menor *downtime* dos serviços. No entanto, existem *workloads* em que as páginas de memória serão atualizadas com muita frequência. Logo, haverá uma grande sobrecarga causada pela transferência das páginas de memória que serão posteriormente modificadas. Além disso, os autores pressupõem que essas páginas atualizadas são transferidas muito mais rápido que a taxa de atualização, caso contrário, todo o trabalho será ineficaz e deverá ser interrompido. Essa limitação faz com que o algoritmo seja aplicado somente em LANs de alta velocidade.

2.2.3 *Live Migration* baseada na Pré-Cópia de *Trace/Checkpoint*

Com o intuito de reduzir o *overhead* gerado pela transferência das páginas de memória durante o processo de pré-cópia, (LIU *et al.*, 2009) propuseram uma nova abordagem que utiliza tecnologias de *trace/replay* (XU *et al.*, 2007) e *checkpointing/recovery* (CULLY *et al.*, 2008) para recuperar o estado da VM no hospedeiro de destino de forma eficiente e, assim, minimizar o *downtime* das aplicações que nela executam e o tráfego da rede. Diferente da etapa de pré-cópia descrita anteriormente, essa abordagem conta com um arquivo de *checkpoint*, que contém informações (conteúdo dos registradores da CPU e o estado dos dispositivos) a respeito da VM no hospedeiro de origem em um determinado instante. Na primeira iteração, o arquivo *checkpoint* é copiado do hospedeiro de origem para o hospedeiro de destino, enquanto a VM na origem está continuamente executando e eventos de sistema são gravados no arquivo de *log*. Iterações subsequentes copiam o arquivo de *log* gerado durante a transferência do ciclo anterior. Os *logs* orientam o sistema de destino sobre a forma como ele deve re-executar as ações a partir do *checkpoint* para alcançar um estado sincronizado com a VM no hospedeiro de origem.

Uma vez que o arquivo de *log* é bem menor que as páginas de memórias atualizadas na iteração anterior, a quantidade de dados transferidos durante a etapa de pré-cópia é reduzida significativamente. Além disso, reduz o *downtime* através da combinação da fase de transferência iterativa do *log* com a fase *stop-and-copy*,

uma vez que, após várias rodadas de iteração, o último arquivo de log transferido na fase de *stop-and-copy* é reduzido a um tamanho insignificante para que o tempo de *downtime* possa ser imperceptível. Todavia, a ferramenta utilizada para recuperação do estado da VM no hospedeiro de destino, chamada ReVirt (DUNLAP *et al.*, 2002), executa as instruções necessárias para tal, desde que o processador na origem e destino sejam iguais, ou seja, a solução está limitada a cenários onde o *hardware* é homogêneo.

2.2.4 *Live Migration* baseada na Pós-Cópia

Em contraste às abordagens de pré-cópia, Hines e Gopalan (2009) propuseram a estratégia *live migration* baseada na pós-cópia com o propósito de reduzir o tempo de realocação das VMs durante o processo de migração. Na abordagem de pós-cópia, o conteúdo dos registradores da CPU e o estado dos dispositivos são capturados no hospedeiro de origem e transferidos para o hospedeiro de destino. No destino, a VM é executada somente com base nessas informações e as páginas de memória são transferidas após a execução. A chave para a estratégia de migração baseada na pós-cópia é um mecanismo de transferência de memória sob demanda, que aprisiona o primeiro acesso a uma página de memória no destino e copia o seu conteúdo a partir do hospedeiro de origem.

Apesar da pós-cópia reduzir o tempo de realocação da VM, uma vez que é necessário apenas transferir o conteúdo dos registrados de CPU e o estado dos dispositivos para sua execução no destino, é provocada uma degradação no desempenho das aplicações que executam na VM, visto que ela é interrompida toda vez que uma página de memória não está disponível e somente é reinicializada quando a página de memória é recuperada a partir do host de origem. No sentido de atenuar o *downtime* sofrido pela VM depois de sua realocação no hospedeiro de destino, os autores implementaram a técnica de pré-paginação para trabalhar juntamente com mecanismo de paginação sob demanda. Esta técnica é utilizada para esconder a latência das falhas de página ocorridas quando a VM acessa uma página de memória que ainda não está disponível. Para tanto, ela tenta antecipar a ocorrência das falhas modificando a sequência de páginas requisitadas na origem através da rede para melhor refletir o padrão de acesso à memória da VM. Essa abordagem funciona usando os endereços de falha como dicas para estimar a localização espacial do padrão de acesso às páginas de memória da VM. O componente de pré-paginação muda então a sequência das páginas requisitadas de

tal forma que o conteúdo das páginas capturadas tem a maior probabilidade de ser acessado pela VM em um futuro próximo. Recentemente, a combinação de paginação sob demanda com a pré-paginação também foi utilizada em (HIROFUCHI *et al.*, 2010) para compor a estratégia de migração ao vivo de máquinas virtuais baseada na abordagem pós-cópia visando um processo de migração mais rápido e, ao mesmo tempo, reduzir o *downtime* experimentado pela VM.

2.2.5 *Non-Live Migration versus Live Migration*

Em (MAGALHÃES; SOARES; GOMES, 2011), apresentamos uma avaliação do impacto das abordagens de migração disponibilizadas, por padrão, pelo hipervisor Xen: stop-and-copy e pré-cópia de memória. Essas duas abordagens foram analisadas em um ambiente de experimentação real, com 4 servidores físicos heterogêneos, a partir de 4 métricas de performance: tempo total de migração, downtime, tempo de resposta e vazão de demanda. Os resultados experimentais obtidos corroboram aqueles encontrados na literatura: (i) o downtime da stop-and-copy é pelo menos 5 vezes maior que o da pré-cópia, (ii) o tempo total de migração da pré-cópia foi superior ao da stop-and-copy para todas as cargas de trabalho, (iii) a pré-cópia provocou um incremento no tempo de resposta das aplicações, caracterizando quebra de SLA para benchmarks de aplicações web, enquanto a stop-and-copy causou indisponibilidade dos serviços.

Apesar do *downtime* apresentado pela pré-cópia ser bem inferior ao da *stop-and-copy*, seu valor é bem acima do esperado em milésimos de segundos, previamente relatados na literatura para uma série de cargas (CLARK *et al.*, 2005; OLIVEIRA; PETRUCCI; LOQUES, 2010). Entretanto, a combinação de cargas de trabalho utilizada foi testada em outros trabalhos (VOORSLUYS *et al.*, 2009) para avaliar o impacto da *live migration* no desempenho do sistema e o mesmo *downtime* e comportamento do tempo de resposta foi encontrado, reforçando os resultados alcançados nesse trabalho.

Outro resultado interessante a ser destacado consiste na pré-cópia ser vulnerável ao tipo de carga que executa no interior da VM. Uma mesma VM foi submetida à diferentes cargas de trabalho com suas respectivas características. Para as cargas orientadas à memória, a sobrecarga causada pela transferência de páginas de memória que são posteriormente modificadas é grande, implicando em um maior tempo total de migração. Diante disso, é arriscado definir o impacto da migração de

uma VM no desempenho do sistema a priori, quando o perfil da carga de trabalho utilizada não é bem definido. Na Tabela 2.2 sumarizamos as abordagens de migração de máquinas virtuais discutidas nesta seção.

Tabela 2.2: Abordagens de Migração de Máquinas Virtuais

Categorias	**Abordagens**		**Características**	**Referências**
Non-Live Migration	Stop-and-Copy		A VM é suspensa durante todo o processo de migração	[Schmidt, 2000] [Osman et al, 2002]
Live Migration	Pré-copia	Memória	A transferência das páginas de memória durante a fase de pré-cópia	[Clark et al, 2005]
		Trace	Transferência dos arquivos de checkpoint e trace de execução durante a fase de pré-cópia	[Liu et al, 2009]
	Pós-copia		A transferência da memória é realizada após a execução da VM no host de destino	[Hines, 2009] [Hirofuchi et al, 2010]

2.3 Caracterização do Problema

Na Seção 2.1, discutimos trabalhos recentes que propuseram diferentes soluções para gerenciamento de recursos a fim de economizar energia e/ou minimizar a perda de desempenho a nível de clusters, *data centers* e Nuvens. Com base nesses trabalhos (VERMA; AHUJA; NEOGI, 2008; PETRUCCI; LOQUES; MOSSÉ, 2010; BELOGLAZOV; BUYYA, 2010; BELOGLAZOV; BUYYA, 2011), caracterizamos o problema de alocação de VMs em servidores físicos, destacando suas principais variáveis e um modelo de potência utilizado em nossa estratégia de alocação dinâmica de VMs.

A alocação de VMs em servidores físicos pode ser vista como um problema de empacotamento binário com caixas de diferentes tamanhos, onde os servidores representam as caixas e as VMs representam os objetos a serem empacotados. O

objetivo da nossa solução para este problema é reduzir o consumo de energia total do sistema. Neste caso, uma vez que a energia consumida por um hospedeiro é representada pelo custo da caixa, a solução visa empacotar os objetos de modo a minimizar o custo total das caixas utilizadas.

Apesar do consumo de energia de um hospedeiro ser determinado principalmente pelo processador, memória, armazenamento em disco e interfaces de rede, trabalhos recentes têm mostrado que a CPU representa uma grande parte desse todo (BELOGLAZOV; BUYYA, 2010; BELOGLAZOV; BUYYA, 2011), de modo que a energia consumida por um hospedeiro pode ser descrita por uma relação linear (LIAO; HU; JIN, 2010; BELOGLAZOV; ABAWAJY; BUYYA, 2011):

$$P(u) = k \cdot Pmax + (1 - k) \cdot Pmax \cdot u \tag{2.1}$$

na qual $Pmax_i$ é a potência máxima consumida pelo hospedeiro ocupado i, u consiste no percentual de utilização de CPU desse hospedeiro, e k representa a fração de potência consumida pelo servidor no estado ocioso.

Assim, temos um conjunto N de diferentes hospedeiros, onde cada hospedeiro $i \in N$ possui uma capacidade máxima W_i e um custo energético $P(u_i)$. O problema envolve o empacotamento de um conjunto M de máquinas virtuais em um conjunto mínimo de hospedeiros com o menor custo, onde cada máquina vitual $j \in M$ possui uma demanda W_j.

Considerando que o ambiente de computação virtualizada consiste de N hospedeiros, o custo energético deste ambiente consiste na soma da energia consumida por todos os hospedeiros ativos. Portanto, a função objetivo do problema consiste em minimizar o consumo de energia global do sistema:

$$P_{total}(u) = \sum_{i=1}^{N} k_i \cdot Pmax_i + (1 - k_i) \cdot Pmax_i \cdot u_i. \tag{2.2}$$

As restrições para o problema são apresentadas a seguir:

$$\sum_{j \in M} W_j \leq W_i \quad \forall i \in N \tag{2.3}$$

$$\sum_{j \in M} X_{ij} = 1 \quad \forall i \in N \tag{2.4}$$

$$X_{ij} \in \{0,1\} \quad u_i, k_i \in [0,1] \quad \forall i \in N, \forall j \in M, \tag{2.5}$$

onde (3) evita uma possível solução em que a demanda de todas as j máquinas virtuais pertencentes a M executando em um hospedeiro i excedam a capacidade do mesmo, (4) garante que uma maquina virtual j não seja atribuída a mais de um hospedeiro $i \in N$ a ultima restrição (5)define a variável de decisão X_{ij} como inteira e binária e limita as variáveis u_i e k_i ao intervalo fechado entre 0 e 1.

2.4 Conclusão

Tendo em vista que ambientes computacionais modernos, tais como *data centers* e Nuvens, consomem grandes quantidades de energia promovendo, assim, problemas de ordem econômica e ambiental, a eficiência energética despontou como um dos resquisitos mais importantes para tais ambientes. Neste capítulo, classificamos, discutimos e categorizamos, com auxílio da taxonomia proposta em (BELOGLAZOV *et al.*, 2011), diferentes soluções para alcançar eficiência energética em sistemas computacionais virtualizados.

O gerenciamento eficiente de recursos pode reduzir significamente o consumo de energia do sistema, enquanto mantem os requisitos de desempenho. Um dos avanços que tem facilitado um maior desenvolvimento na área é a implementação de DVFS por fornecedores de *hardware*. Estas tecnologias têm permitido o controle de *software* sobre o consumo de potência da CPU. A gestão de energia consiste no monitoramento da CPU e no ajuste contínuo da voltagem de alimentação/frequência de *clock* para atender aos requisitos de desempenho atual. Outro avanço significativo para o gerenciamento de recursos são as tecnologias de virtualização, que permitem a consolidação da VMs em um número reduzido de servidores físicos e subsequente desligamento dos servidores ociosos.

As abordagens *non-live migration* e *live migration* permitem a consolidação dinâmica de VMs de acordo com requisitos de desempenho atual do sistema. No entanto, a migração de VMs implica em degradação no desempenho do sistema através, uma vez que para realizá-la, é necessário parar a execução da VM, provocando sua indisponibilidade e, por conseguinte, das aplicações que nela executam. Além disso, há um consumo extra de ciclos de CPU e de largura de banda no processo de transferência da VM de um hospedeiro para outro, impactando nos recursos disponibilizados para os serviços providos pela mesma.

Assim, uma melhor compreensão do impacto da migração no desempenho do sistema é fundamental para concepção de estratégias aptas a utilizar os recursos de maneira eficiente sem comprometer a capacidade de resposta e disponibilidade dos serviços ofertados. Nesse contexto, discutimos abordagens de migração de *live* e *stop-and-copy*, destacando inclusive, por meio de experimentos realizados em nosso *testbed*, as vantagens e desvantagens de cada uma delas.

Discutimos trabalhos recentes que propuseram soluções onde lidam com o gerenciamento de recursos a fim de economizar energia e/ou minimizar a perda de desempenho a nível de clusters, *data centers* e Nuvens. Com base nesses trabalhos, caracterizamos o problema de alocação de VMs em servidores físicos, destacando suas principais variáveis e um modelo de potência útil para determinar o consumo de energia de um servidor. Uma melhor compreensão do problema orientou a concepção da estratégia proposta, apresentada no Capítulo 3.

Capítulo 3

Proposta: Estratégia de Alocação Dinâmica de VMs

Uma das principais causas da ineficiência energética em *data centers* é a baixa utilização média dos recursos (BELOGLAZOV *et al.*, 2011). Em vista disso, propomos nesta dissertação uma estratégia de alocação de VMs para que tais ambientes possam lidar com a natureza dinâmica da carga a fim de economizar energia sem promover sobrecargas. Dessa forma, quando a demanda por recursos é reduzida, as VMs são consolidadas em um número mínimo de servidores físicos, enquanto os recursos excedentes são desligados, reduzindo o consumo de energia total no sistema. Em contrapartida, quando a demanda por recursos excede a capacidade de provisionamento do sistema, as VMs dos hospedeiros sobrecarregados são distribuídas a fim de evitar que essas sobrecargas comprometam as expectativas de QoS dos usuários, promovendo até mesmo a indisponibilidade dos serviços ofertados. Caso os hospedeiros ativos não-sobrecarregados possuam recursos insuficientes para hospedar as VMs, novos servidores físicos podem ser ligados para dissipar as sobrecargas.

Este capítulo está organizado da seguinte maneira: a Seção 3.1 apresenta uma visão geral da proposta, com seus componentes principais e respectivas funcionalidades. A Seção 3.2 descreve as heurísticas de seleção e ordenação de hospedeiros origem/destino e das políticas de seleção das VMs a serem migradas. Na sequência, algoritmos de monitoramento e mitigação estão descritos nas Seções 3.3 e 3.4, respectivamente.

3.1 Descrição Geral da Proposta

A alocação de VMs, base da nossa proposta, pode ser dividida em duas fases: a primeira consiste na admissão de novos pedidos para provisionamento e alocação das VMs nos servidores físicos, enquanto a segunda fase consiste na otimização da alocação atual (BELOGLAZOV *et al.*, 2011). A primeira parte está diretamente relacionada ao algoritmo de escalonamento (orquestração das VMs). Nossa abordagem atua na segunda fase, por consequência, pressupomos que o algoritmo de escalonamento foi executado no sistema e existe uma alocação prévia de VMs nos servidores físicos.

Uma vez que a carga do sistema é dinâmica, a alocação das VMs é um problema de otimização perene. Nesta dissertação, utilizamos a abordagem *live migration* para mover dinamicamente as VMs através de hospedeiros distintos a fim de melhorar o mapeamento dos recursos. Entretanto, essa migração impacta negativamente na eficiência energética do sistema, visto que o processo migratório consome recursos como, por exemplo, CPU e interface de rede, implicando no aumento do consumo de energia que não pode ser desprezado (*vide* Seção 2.2). Portanto, neste trabalho, consideramos o impacto do processo migratório, tanto quando definimos uma política de alocação de VMs para reduzir o número de migrações como também quando definimos uma política de seleção de VMs que minimiza o tempo total de migração.

Nossa estratégia de alocação dos recursos implementa dois algoritmos principais: o de monitoramento e o de mitigação. A Figura 3.1 apresenta uma visão geral desses algoritmos e como se relacionam. O algoritmo de monitoramento é responsável por detectar pontos de sobrecarga/ócio do sistema e, consequentemente, determinar a necessidade de migração das máquinas virtuais. Para tanto, este algoritmo captura, em intervalos periódicos, dados a respeito da utilização dos recursos para cada servidor físico. Na sequência, ele classifica cada servidor de acordo com as políticas de limiar único e duplo, discutidas na Seção 3.2. Por fim, ele busca pela ocorrência de algum servidor físico cuja classificação diverge de normal, definindo a necessidade de ajuste no mapeamento de recursos virtuais em físicos. Caso nenhum servidor seja encontrado, o algoritmo volta a capturar os dados, respeitando o intervalo de monitoramento. Caso contrário, o algoritmo de mitigação é invocado. Assim, a tomada de decisão do algoritmo de monitoramento é realizada com base na observação da utilização dos recursos de cada servidor físico em um intervalo de

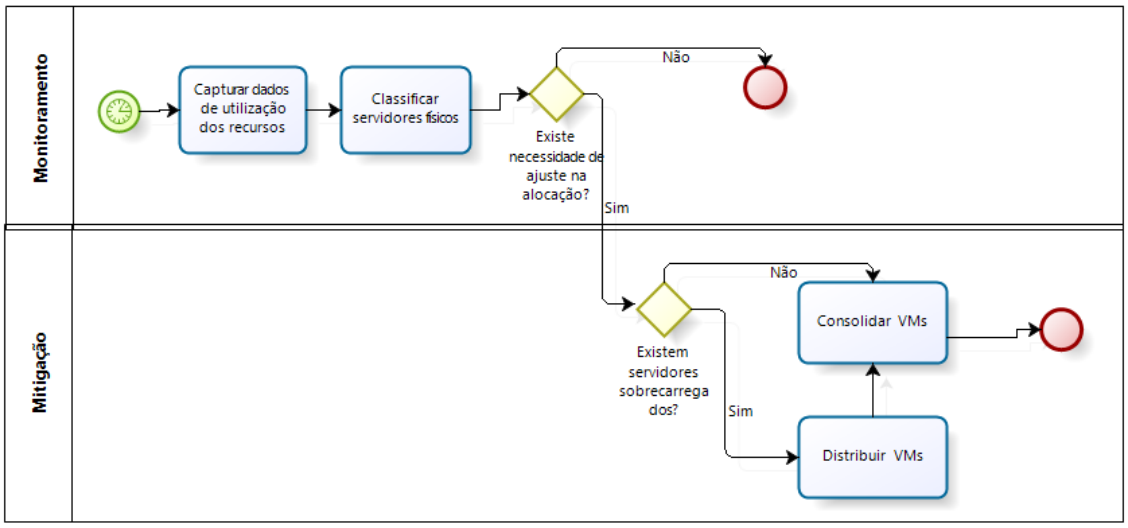


Figura 3.1: Visão geral da estratégia proposta para alocação dinâmica de VMs

medição.

O algoritmo de mitigação baseia-se em políticas de seleção de VMs e heurísticas de alocação de recursos para determinar quais VMs devem ser migradas e para onde migrá-las, respectivamente. Uma vez que o algoritmo de monitoramento detecte algum ponto de sobrecarga e/ou ócio no sistema, ele invoca o algoritmo de mitigação. Ainda na Figura 3.1, observamos que o algoritmo de mitigação verifica a existência de algum servidor físico sobrecarregado. Caso a assertiva seja verdadeira, o algoritmo dá início à etapa de distribuição das VMs visando dissipar a sobrecarga. Caso contrário, não existem servidores sobrecarregados, e o algoritmo de mitigação dá início à etapa da consolidação das VMs a fim de economizar energia. Salienta-se que existe uma prioridade do algoritmo de distribuição em relação a consolidação. Isso ocorre porque não vale a pena consolidar a carga enquanto existe uma sobrecarga que pode comprometer o serviço disponibilizado, frustrando as expectativas de QoS do usuário e incorrendo em custos ao provedor devido à quebra de SLA.

Vale destacar (*vide* Figura 3.1) que mesmo após dissipar a(s) sobrecarga(s), o algoritmo de distribuição de VMs dá inicio à etapa de consolidação com o intuito de economizar energia, caso algum servidor possa ser desligado. Os algoritmos de distribuição e consolidação das VMs serão detalhados nas Subseções 3.4.1 e 3.4.2 respectivamente.

3.2 Políticas e Heurísticas Utilizadas

O problema de alocação das VMs em servidores físicos é NP-difícil (BELOGLAZOV; ABAWAJY; BUYYA, 2011). Uma vez que ambientes em Nuvem necessitam de uma infraestrutura computacional dinâmica em larga escala, capaz de lidar com a natureza elástica da demanda por recursos, optamos por heurísticas, as quais são capazes de encontrar uma solução em um curto período de tempo, ainda que esta não seja a solução ótima, mas ainda com um grau de aproximação satisfatório (DASGUPTA; PAPADIMITRIOU; VAZIRANI, 2006).

Existem várias heurísticas que podem ser utilizadas para solucionar o problema de alocação das VMs , tais como: FF (*First-Fit*), BF (*Best-Fit*) e WF (*Worst-Fit*). Neste trabalho, utilizamos a BF e WF e propomos a BFM (*Best-Fit Modified*) e o WFM (*Worst-Fit Modified*). Na BF, cada VM do hospedeiro de origem é capturada e alocada no hospedeiro de destino que tem a menor capacidade disponível, mas que consegue, ainda, armazená-la sem gerar uma sobrecarga para o mesmo. Em contraste, na WF, cada VM do hospedeiro de origem é capturada e alocada no hospedeiro de destino que tem a maior capacidade disponível. Na BFM, cada VM do hospedeiro de origem é capturada e alocada no hospedeiro que apresenta o maior capacidade ocupada com menor custo energético. Na WFM, cada VM do hospedeiro de origem é capturada e alocada no hospedeiro que possuir maior capacidade disponível com menor custo energético. O custo energético de cada servidor físico é apresentado na Tabela 4.1. Portanto, inserimos nas heurísticas clássicas WF e BF a capacidade de também considerar o custo energético de um hospedeiro na tomada de decisão.

Uma vez determinados os hospedeiros de origem e destino para o processo migratório, as VMs serão dispostas dentro do hospedeiro de origem de acordo com duas políticas de seleção das VMs a serem migradas: MVC (Máximo Volume de Carga)(*Maximum Volume Load*) (WOOD *et al.*, 2009) e MTM (Menor Tempo de Migração)(*Minimum Migration Time*) (BELOGLAZOV; BUYYA, 2011). A política MVC dispõe as VMs candidatas conforme o volume de carga (WOOD *et al.*, 2009):

$$Vol = \frac{1}{1 - CPU} \times \frac{1}{1 - MEM}, \tag{3.1}$$

em que CPU e MEM correspondem ao percentual de utilização do processador

e memória, respectivamente. Assim, temos o conjunto M_i de VMs hospedadas no servidor i. O objetivo dessa política reside em selecionar a VM_j de maior volume de carga para ser migrada e, com isso, potencialmente minimizar o número de migrações. Desse modo, maior será a chance do hospedeiro deixar de ser sobrecarregado em uma única migração, reduzindo o impacto das migrações no desempenho do sistema. Portanto, a seguinte condição deve ser atendida:

$$Vol(j) \geq Vol(a), \quad j \in M_i, \forall a \in M_i, \tag{3.2}$$

onde $Vol(j)$ é o volume de carga da VM_j e $Vol(a)$ é o volume de carga da VM_a.

A política MTM dispõe as VMs canditadas à migração conforme o tamanho da imagem de memória e seu objetivo reside em encontrar a VM_j que requer a menor quantidade de memória a ser migrada tal que o tempo total de migração seja minimizado. Quanto maior o tempo de migração também maior o tempo em que os recursos de memória e processamento estarão sendo consumidos por esse processo e, consequentemente, ocorrerá um aumento na quantidade de energia consumida. Vale ressaltar que o tempo total de migração é definido pela razão (quociente) entre o tamanho da imagem de memória a ser transferida e a largura de banda de rede disponível para o host i. Uma vez que a largura de banda é a mesma para todas as VMs, consideramos apenas o tamanho da imagem de memória na seleção das VMs. Assim, as condições apresentadas na Equação 3.3 devem ser satisfeitas.

$$RAM(j) \leq RAM(a), \quad j \in M_i, \forall a \in M_i, \tag{3.3}$$

onde $RAM(j)$ é a quantidade de memória requerida pela VM_j. De forma análoga, $RAM(a)$ é a quantidade de memória requerida pela VM_a.

3.3 Monitoramento

O algoritmo de monitoramento busca pela ocorrência de hospedeiros ociosos/subutilizados e/ou sobrecarregados. Neste trabalho, nossa estratégia utiliza políticas para determinar a necessidade de ajuste no mapeamento de recursos baseada em limiares fixos: (i) a primeira delas possui um limiar único, através do qual a ideia principal consiste em estabelecer um limite de utilização superior para os recursos dos hospedeiros. Assim, as VMs são realocadas em um hospedeiro,

desde que o seu limiar não seja violado. O objetivo é preservar os recursos livres visando prevenir a violação de SLA devido à consolidação agressiva ou ao aumento na demanda por recursos; (ii) a segunda política estabelece limiares superior e inferior para utilização dos recursos dos hospedeiros. Seu objetivo é reduzir o número de migrações e, consequentemente, seu impacto no desempenho do sistema. Portanto, um hospedeiro é considerado sobrecarregado quando seus recursos (processador e memória) excedem o limiar superior e persistem nesse estado durante um intervalo de tempo. De forma semelhante, um hospedeiro é dito ocioso ou subutilizado quando seus recursos não atingem o limiar inferior e persistem nesse estado durante um intervalo de tempo pré-determinado. Esse intervalo de tempo depende do número de amostras que são consideradas na classificação dos servidores. Durante o intervalo de monitoramento, são coletadas, a cada 5 segundos, amostras a respeito da utilização dos recursos para cada servidor físico. Desse modo, se utilizarmos apenas a última amostra para classificar os servidores, estamos considerando que eles persistiram 5 segundos no mesmo estado.

Uma visão geral do *loop* de controle da estratégia é mostrada no Algoritmo 3.1, composto por 3 passos. O primeiro passo consiste em iniciar a coleta de informações a respeito da utilização dos recursos (memória e CPU) (linhas 2-4 do Algoritmo 3.1). Assim, o algoritmo de monitoramento captura, em intervalos periódicos, essas informações para cada hospedeiro ativo no sistema, dando início ao *loop* principal (linhas 5-16 do Algoritmo 3.1). O segundo passo consiste na classificação dos hospedeiros conforme a política de limiar previamente selecionada (linha 7 do Algoritmo 3.1). No limiar único, os hospedeiros podem assumir dois estados: sobrecarregado e normal. Já no limiar duplo, os hospedeiros podem assumir três estados: ocioso, normal e sobrecarregado.

Após a classificação dos hospedeiros, o algoritmo de monitoramento é capaz de identificar aqueles sobrecarregados e/ou ociosos. Uma vez identificada a necessidade ou possibilidade de ajuste no mapeamento de máquinas virtuais em servidores físicos, o algoritmo de monitoramento invoca o algoritmo de mitigação (*vide* Subseção 3.4) conforme a política de limiar selecionada (linhas 9-16 do Algoritmo 3.1) Assim, o terceiro e último passo do *loop* principal é realizado. No limiar único, o algoritmo de mitigação sempre é invocado, visto que essa política busca sempre consolidar a carga desde que não exista nenhum hospedeiro sobrecarregado. Em contrapartida, no limiar duplo, o algoritmo de mitigação é executado somente se existir pelo menos

um hospedeiro classificado como sobrecarregado e/ou ocioso. Assim, caso a assertiva não seja verdadeira, o algoritmo de mitigação não é executado e o *loop* do Algoritmo 3.1 volta a capturar as informações de utilização dos recursos, respeitando o intervalo de monitoramento.

Algoritmo 3.1 Loop Principal

Input: listaHosts, heuristicaLimiar, heuristicaAlocacao, heuristicaSelecaoVM, intervaloMonitoramento

```
1: // 1. Monitorar servidores físicos
2: para todo host ∈ listaHosts fazer
3:    iniciaColetaDados(host)
4: fim para
5: laço
6:    // 2. Classificar servidores físicos
7:    hostsClassificados ← classificaHost(heuristicaLimiar, limiarSuperior,
      limiarInferior, intervaloMonitoramento, listaLogRecursosUtilizadosHosts)
8:    // 3. Invocar algoritmo de mitigação
9:    se heuristicaLimiar =′ limiarUnico′ então
10:      executaMitigacao(hostsClassificados, heuristicaLimiar, heuristicaAlocacao,
         heuristicaSelecaoVM)
11:   senão se heuristicaLimiar =′ limiarDuplo′ então
12:      se possuiHostSobrecarregado(hostsClassificados) or
         possuiHostOcioso(hostsClassificados) então
13:         executaMitigacao(hostsClassificados, heuristicaLimiar, heuristicaAlocacao,
            heuristicaSelecaoVM)
14:      fim se
15:   fim se
16: fim laço
```

A abordagem de monitoramento utilizada neste trabalho é conhecida como black-box (WOOD *et al.*, 2009), e captura o percentual de uso de CPU e memória de cada hospedeiro e suas respectivas VMs em um intervalo de monitoramento e gera *logs* para tomada de decisões. A vantagem desse tipo de abordagem reside no fato da decisão ser baseada na aferição da utilização dos recursos apenas a partir de observações externas e independente do sistema operacional residente dentro de cada VM. Portanto, pode ser aplicada tanto em Nuvens privadas quanto em Nuvens públicas, onde os provedores necessitam de abordagens de monitoramento menos intrusivas.

3.4 Mitigação

O algoritmo de mitigação é responsável por dissipar os pontos de ócio e/ou sobrecarga do sistema através da migração de VMs entre hospedeiros distintos conforme variação da demanda por recursos. Esse algoritmo é constituído por dois procedimentos: o de consolidção das VMs e o de distribuição das VMs. Primeiramente, o algoritmo de mitigação verifica a lista de hospedeiros classificados (pelo algoritmo de monitoramento) em busca da existência de algum sobrecarregado. Se existir pelo menos um hospedeiro sobrecarregado, o algoritmo de mitigação invoca o algoritmo de distribuição das VMs. Caso contrário, significa que os hospedeiros estão ociosos/subutilizados e/ou normais, e o algoritmo de mitigação chama o algoritmo de consolidação das VMs, conforme ilustrado no Algoritmo 3.2.

Nesta etapa da estratégia, as heurísticas de alocação BF, BFM, WF e WFM são utilizadas para definir o par de servidores origem/destino envolvidos no processo migratório. Os procedimentos de consolidação e distribuição das VMs estão descritos nas Subseções 3.4.1 e 3.4.2, respectivamente.

Algoritmo 3.2 Mitigaçao

Input: hostsClassificados, heuristicaLimiar, heuristicaAlocaçao, heuristicaSelecaoVM

1: **se** $possuiHostSobrecarregado(hostsClassificados)$ **então**
2: $\quad distribuiVMs(hostsClassificados, heuristicaAlocacao, heuristicaSelecaoVM)$
3: **senão**
4: $\quad consolidaVMs(hostsClassificados, heuristicaAlocacao, heuristicaLimiar)$
5: **fim se**

3.4.1 Consolidação das VMs

O Algoritmo 3.3 ilustra o primeiro passo da consolidação de VMs, o qual consiste em buscar por hospedeiros ociosos que não estejam executando nenhuma VM a fim de desligá-los para economizar energia (linha 1). Umas vez que esses servidores físicos forem desligados, o processo de consolidação dispõe o restante dos hospedeiros classificados conforme a heurística de alocação definida (BF, BFM, WF e WFM) visando determinar a origem e o destino. No entanto, o algoritmo observa a política de limiar utilizada. No caso do limiar duplo, a consolidação verifica a existência de hospedeiros classificados como normais (pelo algoritmo de monitoramento), pois ele tenta primeiramente migrar as VMs dos hospedeiros ociosos para os hospedeiros

normais. Na linha 6, o operador "seta para a direita"indica que os hospedeiros normais são inseridos na lista de hospederos de destino. O mesmo significado é extrapolado para as linhas 8, 9, 11 e 12. Caso não haja hospedeiros cujo estado é “normal” ou eles não possuam recursos suficientes para hospedar as VMs, a consolidação ocorre entre os próprios hospedeiros ociosos. No caso de limiar único, uma vez que todos os hospedeiros são classificados como ”normais”, a consolidação ocorre entre eles (linhas 10-13 do Algoritmo 3.3). Após ordenar a lista de hospedeiros de origem e de destino, respeitando a heurística de alocação pré-estabelecida, o algoritmo de consolidação das VMs percorre a lista de hospedeiros de origem buscando alocar todas as suas VMs em execução nos hospedeiros presentes na lista de destino, gerando um novo mapeamento (linhas 14-29 do Algoritmo 3.3).

Uma vez que, em média, um hospedeiro inativo consome aproximadamente 70 % da energia que é consumida no caso de uso pleno do processador (BELOGLAZOV; ABAWAJY; BUYYA, 2011), as migrações só começam a acontecer se pelo menos um hospedeiro puder ser desligado. Desse modo, nossa estratégia de economia de energia visa reduzir o consumo total de energia do sistema e evita migrações desnecessárias. O processo é repetido até que toda a lista de hospedeiros de origem tenha sido percorrida. O funcionamento da consolidação independe das políticas de seleção das VMs (MTM e MVC) visto que todas as VMs de um hospedeiro de origem precisam ser migradas para que o mesmo possa ser desligado, independente da ordem em que elas estejam dispostas.

3.4.2 Distribuição de VMs

A ideia básica do algoritmo de distribuição das VMs consiste em deslocá-las dos hospedeiros mais sobrecarregados para os menos sobrecarregados de modo a distribuir a carga do sistema e, assim, dissipar as sobrecargas. Para determinar quais VMs devem ser migradas, primeiro, o algoritmo conta com as heurísticas de alocação BF, BFM, WF e WFM a fim de determinar os hospedeiros de origem e destino. Em seguida, dentro do hospedeiro de origem (mais sobrecarregado), as VMs a serem migradas são dispostas de acordo com as duas políticas de seleção das VMs: MVC e MTM.

O Algoritmo 3.4 apresenta a distribuição das VMs. O primeiro passo deste algoritmo consiste em dispor os hospedeiros sobrecarregados em ordem decrescente de seus volumes de carga (linha 2 do Algoritmo 3.4). O objetivo dessa ordenação é

Algoritmo 3.3 consolidaVMs - Consolidação de VMs (Limiar único e duplo)

Input: hostsClassificados, heuristicaAlocacao, heuristicaLimiar
Output: novoMapeamento

```
1: desligaHostsSemVM(hostClassificados)
2: // Dispõe os hosts conforme heurística de alocação selecionada
3: ordenaHostsConsolidacaoHeuristica(hostsClassificados, heuristicaAlocacao)
4: se heuristicaLimiar =' limiarDuplo' então
5:    se possuiHostsNormais(hostsClassificados) então
6:       hostsNormais → listaHostsDestino
7:    fim se
8:    hostsOciosos → listaHostsOrigem
9:    hostsOciosos → listaHostsDestino
10: senão se heuristicaLimiar =' limiarUnico' então
11:    hostsNormais → listaHostsDestino
12:    hostsNormais → listaHostsOrigem
13: fim se
14: para todo host ∈ listaHostsOrigem fazer
15:    capturaVMsHost(host)
16:    para todo VM ∈ listaVMsHost fazer
17:       buscaNovoMapeamentoConsolidacao(host, VM, listaHostDestino, limiarSuperior)
18:    fim para
19:    se TodasasVMsdohostforamrealocadas então
20:       para todo linha ∈ novoMapeamento fazer
21:          migracaoVM(host, VM, hostDestino)
22:       fim para
23:       verificaMaster(host)
24:       se host =' Master' então
25:          designaNovoMaster(host, listaHostDestino)
26:       fim se
27:       desligaHost(host)
28:    fim se
29: fim para
```

tentar primeiro dissipar a sobrecarga do hospedeiro mais sobrecarregado e, portanto, mais predisposto à quebra de SLAs. Em seguida, o algoritmo verifica se todos os hospedeiros classificados estão sobrecarregados. Caso a assertiva seja verdadeira, o algoritmo busca na lista de servidores desligados um hospedeiro que possua recursos suficientes para hospedar a(s) VM(s) do hospedeiro sobrecarregado (linhas 22-29 do Algoritmo 3.4). Caso existam hospedeiros normais, estes são inseridos na lista de hospedeiros de destino que é ordenada de acordo com a heurística de alocação definida. Então, a lista de hospedeiros sobrecarregados é percorrida. Para cada hospedeiro sobrecarregado, as VMs são capturadas e dispostas de acordo com a

abordagem de seleção de VMs, em seguida, o algoritmo busca um novo mapeamento para os recursos de modo a dissipar a sobrecarga (linhas 3-21 do Algoritmo 3.4).

Algoritmo 3.4 distribuiCarga - Distribuição de VMs

Input: hostsClassificados, heuristicaAlocacao, heuristicaSelecaoVM
Output: novoMapeamento

```
1: // Dispõe os hosts em ordem decrescente de seu volume de carga
2: ordenaHosts(hostsClassificados)
3: se existeHostsNaoSobrecarregado(hostsClassificados) então
4:   para todo host ∈ hostsNaoSobrecarregados(hostsClassificados) fazer
5:     insereHostListaDestino(host)
6:   fim para
7:   // Dispõe os candidatos a hospedar as VMs conforme heurística de alocação
     selecionada
8:   ordenaHostsDistribuicaoHeuristica(listaHostsDestino, heuristicaAlocacao)
9:   para todo host ∈ hostsSobrecarregados fazer
10:    capturaVMsHostSobrecarregado(host)
11:    ordenaVMsHeuristica(listaVMsHost, heuristicaSelecaoVM)
12:    buscaNovoMapeamentoDistribuicao(host, listaVMsHost, listaHostDestino)
13:    se novoMapeamento ≠ ∅ então
14:      para todo linha ∈ novoMapeamento fazer
15:        migracaoVM(host, vm, hostDestino)
16:      fim para
17:    senão
18:      // Uma vez que não foi possível dissipar a sobrecarga, o algoritmo
         consulta a lista de hosts desligados a fim encontrar um host capaz de
         hospedar a(s) VM(s) do host sobrecarregado
19:      buscaNovoMapeamentoDistribuicao(host, listaVMsHost, listaHostDesligados)
20:    fim se
21:  fim para
22: senão
23:  para todo host ∈ hostsSobrecarregados fazer
24:    capturaVMsHostSobrecarregado(host)
25:    ordenaVMsHeuristica(listaVMsHost, heuristicaSelecaoVM)
26:    // Uma vez que todos os hosts ligados estão sobrecarregados, o algoritmo
       consulta a lista de hosts desligados a fim encontrar um host capaz de
       hospedar VM(s) visando dissipar as sobrecargas
27:    buscaNovoMapeamentoDistribuicao(host, listaVMsHost, listaHostDesligados)
28:  fim para
29: fim se
```

3.5 Resumo

Neste capítulo foi apresentada a estratégia de alocação dinâmica de VMs proposta. Esta estratégia tem como objetivo permitir que ambientes de *cluster* e *data centers* possam lidar com a natureza dinâmica da carga a fim de economizar energia e dissipar sobrecargas que podem comprometer o desempenho dos serviços ofertados.

A solução proposta compreende dois algoritmos principais: o algoritmo de monitoramento e o algoritmo de mitigação. O algoritmo de monitoramento conta com duas políticas de limiar (único e duplo) para identificar a necessidade de ajuste no mapeamento de recursos virtuais em físicos. O algoritmo de mitigação é composto por dois procedimentos: um de consolidação e outro de distribuição das VMs. Tanto o algoritmo de consolidação como o de distribuição utilizam as heurísticas de alocação (BF, BFM, WF e WFM) para definir os hospedeiros de origem e destino envolvidos no processo de migração. O algoritmo de distribuição conta, ainda, com as políticas de seleção (MTM e MVC) para determinar as VMs mais aptas a participar do mesmo processo.

Capítulo 4

Resultados

Neste capítulo descrevemos experimentos realizados com medições reais (Seção 4.2) para avaliar o desempenho da proposta. A Seção 4.1 apresenta em detalhes o ambiente de experimentação.

São avaliadas as heurísticas e políticas que compõem nossa proposta: políticas de limiar único e duplo (LU e LD), heurísticas de alocação (BF, BFM, WF e WFM) e políticas de seleção de VMs (MVC e MTM). O desempenho é quantificado a partir de cinco métricas: (1) utilização de CPU, refere-se ao percentual de CPU utilizado por cada hospedeiro durante a execução da estratégia proposta; (2) utilização de memória, refere-se ao percentual de memória utilizado por cada hospedeiro durante a execução da estratégia proposta; (3) número de migrações, refere-se à quantidade de máquinas virtuais que são migradas durante a execução da estratégia proposta; (4) percentual de utilização dos recursos abaixo do limiar superior, métrica que está relacionada à quantidade total de amostras que apresentam a utilização de seus recursos (CPU e memória) abaixo do limiar superior, ou seja, que não estão sobrecarregadas; (5) energia consumida, refere-se à potência consumida pela estratégia, de acordo com as políticas e heurísticas previamente definidas, ao longo do tempo.

4.1 Ambiente Experimental

O ambiente utilizado nas experimentações consiste em cinco servidores com especificações de *hardware* e *software* heterogêneas (*vide* Tabela 4.1), definidas de acordo com o papel desempenhado por cada máquina no sistema.

A arquitetura do ambiente é ilustrada na figura 4.1, na qual podemos destacar três funções distintas: (1) o *Frontend* é responsável por gerenciar o conjunto de recursos formados pelo XCP-storage, XCP-resource01, XCP-resource02 e XCP-resource03, através da ferramenta *Open Xen Manager* (OPEN..., 2012); (2) XCP-resource01, XCP-resource02 e XCP-resource03 representam os servidores onde serão instanciadas as VMs e suas respectivas aplicações. Para tanto, é utilizado o XCP Server 1.1 que faz parte da infraestrutura de virtualização para Nuvens do XCP. Essa infraestrutura é baseada no hipervisor Xen e foi escolhida por permitir agregar recursos físicos heterogêneos em um mesmo conjunto de recursos. Todos os hospedeiros executam VMs heterogêneas no que diz respeito ao sistema operacional (Ubuntu/Debian) e arquitetura (32/64 bits). Todas as VMs instanciadas contam com a pilha LAMP (Linux, Apache, MySQL e PHP) para a execução das aplicações e, por fim, (3) o XCP-storage representa o servidor NFS. Os cinco nós compartilham esse dispositivo de armazenamento, onde residem as imagens das VMs e os discos virtuais. Todas as máquinas estão conectadas através de uma rede LAN *Fast Ethernet*.

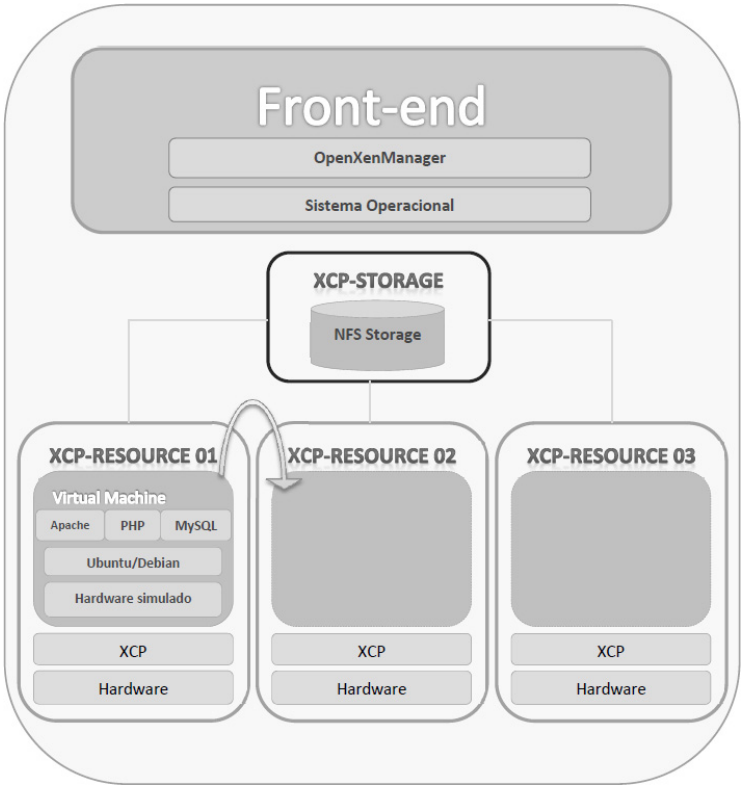


Figura 4.1: Arquitetura do Ambiente de Experimentação

Tabela 4.1: *Hardware* e *software* utilizados para compor o ambiente de experimentação

Hadware	**Software**	**Papel no ambiente**
CPU AMD Turion X2 Dual Core Memória DDR2 4GB Disco Rígido 300GB	Windows OpenXenManager	*Fontend* - Gerenciamento do conjunto de recursos
CPU Intel Core Duo Memória DDR2 2GB Disco Rígido 80GB	Ubuntu Server 10.04	Servidor de armazenamento NFS
CPU Intel Core 2 Duo Memória DDR2 1.5GB Consumo de energia (ocupado): 51.88W Consumo de energia (ocioso): 24.9W	XCP 1.1	XCP-resource01
CPU Intel Pentium Memória 2GB Disco Rígido 500GB Consumo de energia (ocupado): 56.6W Consumo de energia (ocioso): 29.4W	XCP 1.1	XCP-resource02
CPU Intel Core 2 Duo Memória DDR2 2GB Disco Rígido 120GB Consumo de energia (ocupado): 53.3W Consumo de energia (ocioso): 24.4W	XCP 1.1	XCP-resource03

4.2 Resultados Experimentais

Esta seção traz os testes de validação e experimentações realizadas no *testbed* descrito na Seção 4.1. Em todos os experimentos, os nós do ambiente e as conexões de rede são dedicados. Os valores das curvas dos gráficos correspondem a uma média

de 5 (cinco) repetições, com intervalos de confiança de 95%.

4.2.1 Validação da Proposta

Antes de dar início à avaliação e comparação de desempenho das políticas de limiar único e duplo (LU e LD), heurísticas de alocação (BF, BFM, WF e WFM) e políticas de seleção de VMs (MVC e MTM), é imprescindível avaliar se a estratégia proposta comporta-se conforme esperado. Em outras palavras, verificar se a mesma é capaz de identificar a necessidade de ajuste no mapeamento de recursos virtuais em físicos (detectar pontos de sobrecarga/ócio no sistema) e, ainda, realizar esse ajuste através da consolidação e/ou distribuição das VMs. Para tanto, criamos os cenários de testes descritos a seguir.

4.2.1.1 Teste de Validação #1

O primeiro teste de validação consistiu em executar a estratégia proposta no cenário ilustrado na Figura 4.2. O objetivo é validar o comportamento da abordagem de consolidação (algoritmo de mitigação) e das políticas de limiar único e duplo utilizadas na classificação dos servidores (algoritmo de monitoramento). A heurística de alocação aplicada foi a WFM e a política de limiar único foi utilizada, nessa primeira etapa, na classificação dos servidores. Portanto, apenas o limiar superior (fixado em 80%) é levado em consideração. Na figura 4.2, notamos que tanto os percentuais de utilização de CPU quanto os de memória estão abaixo do limiar superior para todos os servidores. Por consequência, os hospedeiros devem ser classificados como não-sobrecarregados, podendo ocorrer a consolidação das VMs entre os mesmos.

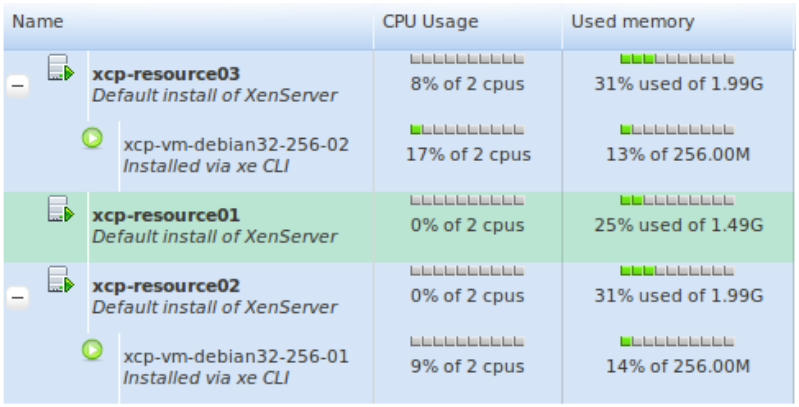


Figura 4.2: Cenário de Validação #1

Antes de observar o comportamento da consolidação, é importante configurar os parâmetros pertinentes à estratégia a fim de obter um melhor desempenho. Em nossa solução, ajustamos os parâmetros a seguir: (i) intervalo de monitoramento, consiste em um intervalo periódico previamente estabelecido, a partir do qual a estratégia coleta informações referentes à utilização dos recursos do ambiente computacional; (ii) janela de monitoramento, consiste no número de amostras dos dados coletados que são utilizadas na classificação dos servidores físicos; (iii) limiar inferior, consiste de um limite inferior para o percentual de utilização dos recursos (CPU e memória) do hospedeiro físico e; (iv) limiar superior, consiste de um limite superior para o percentual de utilização dos recursos do hospedeiro físico.

A figura 4.3 apresenta o consumo de energia do ambiente computacional durante a execução da estratégia proposta no cenário ilustrado na figura 4.2. O intervalo de monitoramento é configurado para 60 segundos e a janela de monitoramento estabelecida em 1 amostra. Conforme o limiar superior previamente determinado, o algoritmo de monitoramento associado a política de limiar único classificou corretamente os servidores xcp-resource01, xcp-resource02 e xcp-resource03 como não sobrecarregados. Considerando apenas a última amostra dos dados coletados obtivemos uma taxa de acerto de 100% na etapa de classificação. Então, repetimos o experimento cinco vezes e a mesma taxa de acerto foi alcançada, evitando a necessidade de avaliar um número maior de amostras para obter um resultado satisfatório na classificação. No que diz respeito aos limiares, nessa primeira etapa a política de limiar único foi utilizada. Portanto, não existe limiar inferior para ser ajustado. Quanto ao limiar superior, o cenário é de consolidação e todos os hospedeiros possuem percentual de utilização dos recursos muito abaixo do limiar superior. Desse modo, o ajuste do limiar superior não se faz necessário.

No gráfico ilustrado na figura 4.3, notamos que após o primeiro intervalo de monitoramento (instante t=60s), ocorre um incremento na potência utilizada. Esse fato decorre da execução do algoritmo de consolidação, que despende processamento com o desligamento de servidores físicos e migração de máquinas virtuais. Em seguida, no instante t=105s, percebemos que o consumo de potência é reduzido e se estabiliza, caracterizando o fim do processo de consolidação. Visto que o algoritmo de consolidação gasta 45 s para convergir, ajustamos o intervalo de monitoramento para 50 s.

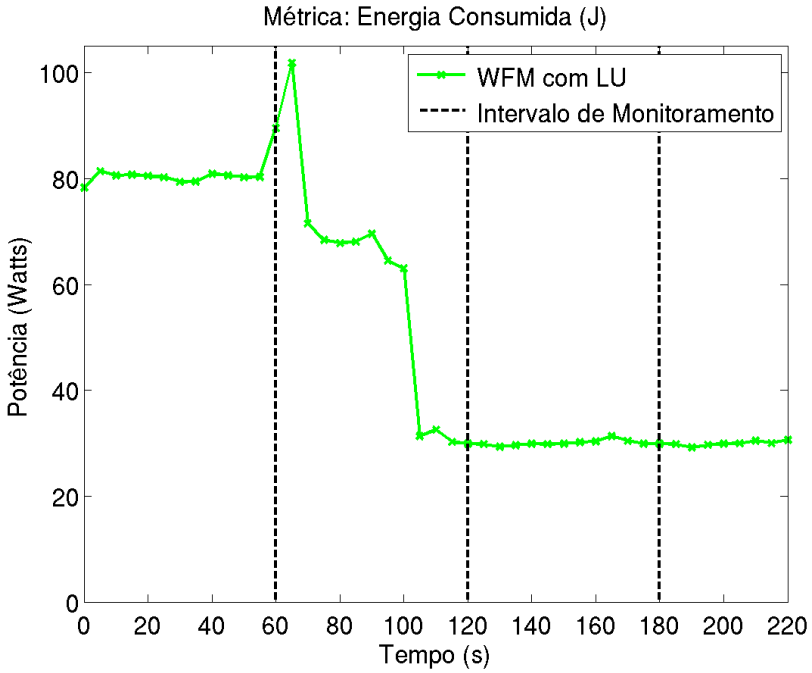


Figura 4.3: Teste de validação #1 - Impacto da abordagem de consolidação associada a política de limiar único no consumo de energia do ambiente

Uma vez que os parâmetros foram determinados, repetimos o teste com os novos valores a fim de avaliar o comportamento do algoritmo de consolidação. A Figura 4.4 mostra o impacto desse teste em termos do consumo de memória do ambiente de experimentação e, através dela, podemos observar o comportamento da consolidação associada à política de limiar único. Ao final do primeiro intervalo de monitoramento (t=50s), os dados a respeito da utilização de CPU e memória são coletados e os servidores físicos classificados. Como nenhum dos servidores está sobrecarregados, o algoritmo de mitigação invoca a consolidação da carga que, por sua vez, inicia o processo de desligamento do xcp-resource01, visto que não possui VMs em execução. Em paralelo, é iniciada a migração da VM xcp-vm-debian32-256-01 do hospedeiro xcp-resource02 para o hospedeiro xcp-resource03 a fim de consolidar a carga. O desligamento do servidor xcp-resource01 pode ser observado (*vide* figura 4.4), através da redução do percentual de utilização de memória até 0, a partir do instante t=60s. A migração apresenta-se através da redução do percentual de utilização de memória no xcp-resource02 e incremento de tal percentual no xcp-resource03.

Dado o término do processo de migração, o xcp-resource02 é finalmente desligado. Ao final do segundo intervalo de monitoramento (t = 100 s), observamos que apenas o servidor xcp-resource03 permanece ligado, executando as VMs

xcp-vm-debian32-256-01 e xcp-vm-debian32-256-02. Portanto, concluímos que o servidores foram corretamente classificados, acarretando na ação do algoritmo de consolidação que, por sua vez, hospedou as VMs em um único hospedeiro sem promover sobrecarga. Assim, foi possível o desligamento de dois servidores, promovendo uma redução de mais de 50% (*vide* figura 4.3) no consumo de energia do ambiente de experimentação.

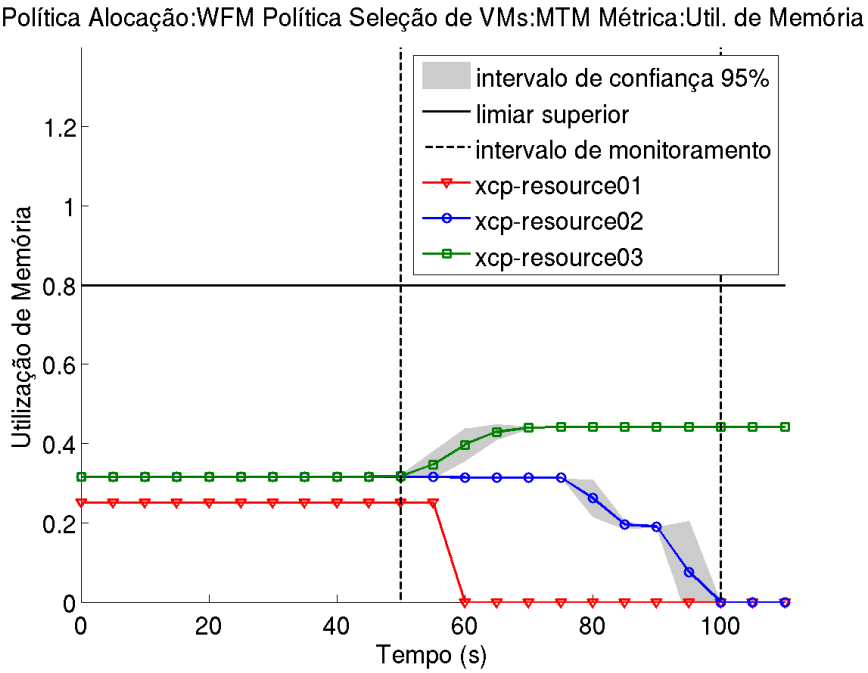


Figura 4.4: Teste de validação #1 - Impacto da abordagem de consolidação associada a política de limiar único no consumo de memória do ambiente

Na segunda etapa do teste de validação, a heurística de alocação WFM é novamente aplicada juntamente com a política de limiar duplo. Antes de observarmos o comportamento da consolidação, configuramos alguns parâmetros pertinentes a estratégia. O intervalo de monitoramento é inicialmente estabelecido em 50 segundos, o limiar superior fixado em 80% e a janela de monitoramento definida em 1 (uma) amostra. Considerando que o cenário foi idealizado para testar a consolidação e que a política de limiar duplo é utilizada, iniciamos a configuração dos parâmetros variando o limiar inferior.

A figura 4.5 apresenta o consumo de energia do ambiente computacional durante a execução da abordagem de consolidação associada às políticas de limiar único e duplo, variando o limiar inferior para o último. Podemos observar que

o comportamento da política de limiar duplo com limiar inferior igual a 25% apresenta-se praticamente constante durante todo o experimento. Ao examinarmos a figura 4.2, notamos que nenhum dos servidores físicos possui os percentuais de utilização de CPU e memória abaixo de 25%. Portanto, nenhum servidor é rotulado como ocioso e, consequentemente, o algoritmo de consolidação não atua no ambiente.

Ainda na figura 4.5, o comportamento da política de limiar duplo com limiar inferior igual a 30% apresenta, a partir do instante t=50s, um incremento no consumo de potência seguido de seu decremento até se estabilizar no instante t=60s. Isso ocorre devido ao servidor xcp-resource01 possuir ambos os percentuais de utilização de recursos abaixo de 30% (*vide* figura 4.2). Desse modo, ele é classificado como ocioso e, então, é invocado o algoritmo de consolidação. O comportamento da política de limiar duplo com limiar inferior igual a 35% apresenta, a partir do instante t=50s, um incremento no consumo de potência seguido de seu decremento até se estabilizar no instante t=105s. Esse comportamento é semelhante ao que ocorre na etapa de consolidação apresentada pela política de limiar único. Isso é justificado pois todos os hospedeiros presentes na figura 4.2 apresentam seus percentuais de utilização de CPU e memória abaixo de 35%. Logo, todos são classificados como ociosos e, então, o algoritmo atua realizando a consolidação entre eles. Assim, de forma análoga ao que ocorre na consolidação com limiar único, as VMs são alocadas no servidor xcp-resource03, enquanto os demais servidores são desligados permitindo uma economia de energia mais significativa.

Com esse experimento, notamos que a variação nos valores atribuídos ao limiar inferior pode impactar diretamente no comportamento da estratégia e, consequentemente, no consumo de energia. Dessa forma, podemos escolher valores maiores para obter uma abordagem de consolidação mais agressiva ou valores menores para uma abordagem mais passiva. Os requisitos do ambiente geralmente norteiam essa escolha.

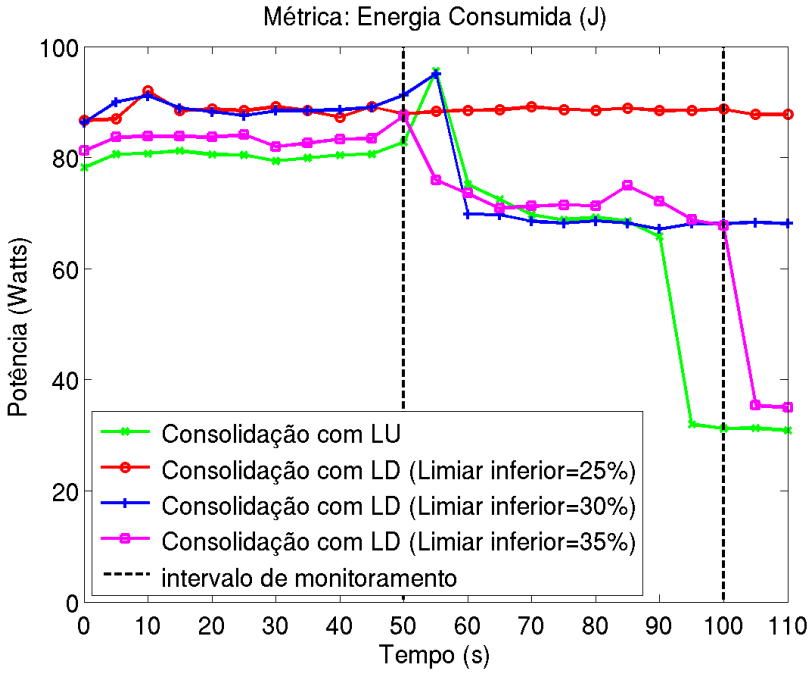


Figura 4.5: Teste de validação #1 - Impacto da abordagem de consolidação associada as políticas de limiar único e duplo no consumo de memória do ambiente de experimentação

Visto que um comportamento semelhante ao da política de limiar duplo com limiar inferior igual a 35% foi apresentado pela política de limiar único e discutido na primeira fase do teste de validação, optamos por analisar o comportamento da consolidação associada à política de limiar duplo com limiar inferior igual a 30%. Desse modo, observamos na figura 4.5 que essa combinação promove uma redução no consumo de potência que se estabiliza antes do intervalo de monitoramento acabar. Visto que o algoritmo de consolidação gasta 10 segundos para convergir, ajustamos o intervalo de monitoramento para 20s. Além disso, considerando apenas a última amostra dos dados coletados obtivemos uma taxa de acerto de 100% na etapa de classificação. Portanto, este parâmetro manteve-se inalterado.

Uma vez que os parâmetros foram devidamente definidos, repetimos o teste com os novos valores. A Figura 4.6 apresenta o impacto desse teste em termos do consumo de memória do ambiente de experimentação e, através dela, podemos observar o comportamento da consolidação associada à política de limiar duplo com limiar inferior igual a 30% . Ao final do primeiro intervalo de monitoramento (t=20s), os dados a respeito da utilização de CPU e memória são coletados e os servidores físicos classificados. Como o servidor xcp-resource01 está ocioso, o algoritmo de mitigação

invoca a consolidação da carga que, por sua vez, inicia o processo de desligamento desse hospedeiro a fim de economizar energia. Visto que o xcp-resource01 não possui VMs em execução, o algoritmo não efetua nenhuma migração. O desligamento pode ser observado, através da redução do percentual de utilização de memória até 0 (zero), a partir do instante t=30s.

Ao final do segundo intervalo de monitoramento (t = 40 s), observamos que apenas os servidores xcp-resource02 e xcp-resource03 permanecem ligados, executando suas respectivas VMs. Isso é esperado, pois conforme a política de limiar duplo, eles são classificados como normais. Logo, não existe a necessidade de invocar o algoritmo de mitigação. A curva do xcp-resource02 está oculta na figura 4.6 porque as curvas dos servidores xcp-resource02 e xcp-resource03 estão sobrepostas.

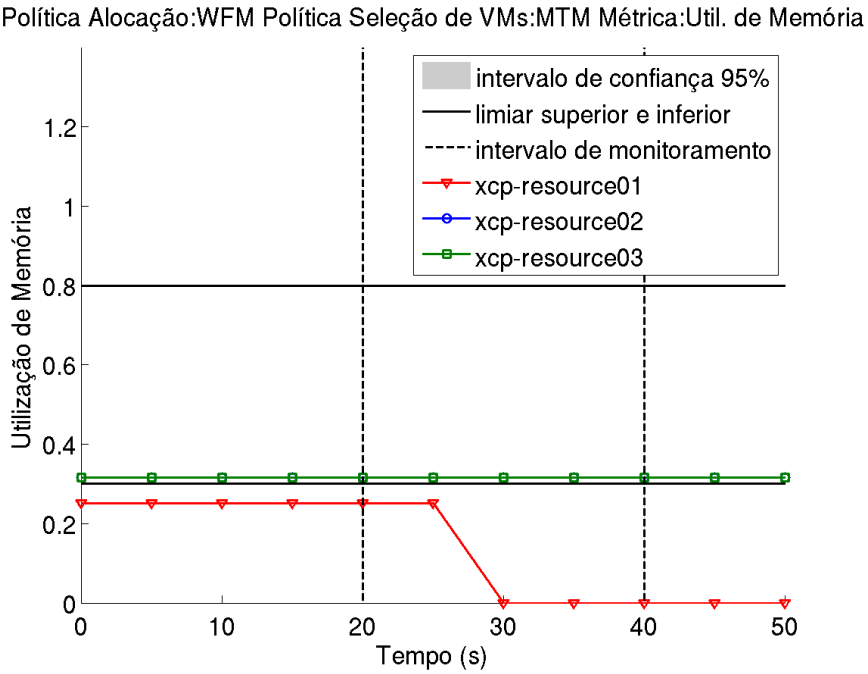


Figura 4.6: Teste de validação #1 - Impacto da abordagem de consolidação associada a política de limiar duplo com limiar inferior igual a 30% no consumo de memória do ambiente

4.2.1.2 Teste de Validação #2

O segundo teste de validação consistiu em executar a estratégia proposta no cenário ilustrado na Figura 4.7. O objetivo deste segundo teste é validar o comportamento da abordagem de distribuição (algoritmo de mitigação) e

das políticas de limiar utilizadas na classificação dos servidores (algoritmo de monitoramento). Como ambas as políticas de limiar possuem um limiar superior, o algoritmo de distribuição comporta-se de modo idêntico para as duas. Portanto, os resultados alcançados por uma política podem ser extrapolados para a outra. Neste teste, aplicamos a heurística de alocação WFM e a política de limiar único. Desse modo, apenas o limiar superior é levado em consideração.

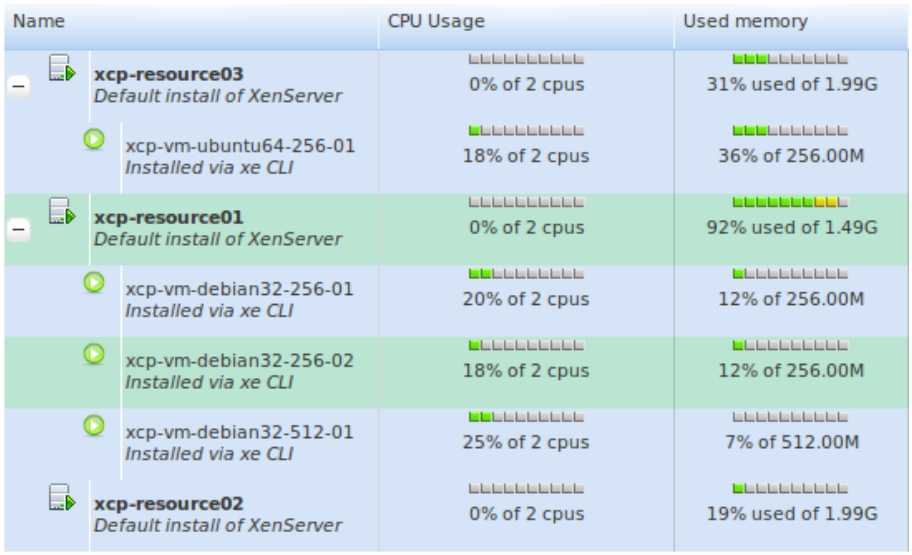


Figura 4.7: Cenário de Validação #2

Antes de observar o comportamento do algoritmo de distribuição, demos início à etapa de ajustes dos parâmetros relevantes para a estratégia. O intervalo de monitoramento foi configurado para 60 segundos e a janela de monitoramento estabelecida em 1 (uma) amostra. O limiar superior foi fixado em 70%.

A figura 4.8 apresenta o consumo de energia do ambiente computacional durante a execução da estratégia proposta no cenário descrito na figura 4.7, com os parâmetros anteriormente definidas. Durante a realização do teste, o algoritmo de monitoramento associado à política de limiar único classificou o servidor xcp-resource01 como sobrecarregado e os demais como não sobrecarregados, corroborando os valores encontrados na figura 4.7, na qual apenas o servidor xcp-resource01 apresenta um percentual de utilização de memória acima de 70%. Considerando apenas a última amostra dos dados coletados obtivemos uma taxa de acerto de 100% na etapa de classificação. Então, realizamos cinco repetições do experimento e a taxa de 100% de acerto foi alcançada em todas elas. Logo, este parâmetro manteve-se inalterado.

Conforme ilustrado na figura 4.8, após o primeiro intervalo de monitoramento (instante t = 60 s), ocorre um incremento na potência utilizada. Esse fato decorre da execução dos algoritmos de distribuição e consolidação, que despendem processamento com o desligamento e migração de máquinas virtuais. Somente no instante t = 160 s, percebemos que o consumo de potência é reduzido e se estabiliza, caracterizando o fim do processo de ajuste dos recursos. Visto que a estratégia gasta mais de 60 segundos para convergir, ajustamos o intervalo de monitoramento para 120 s.

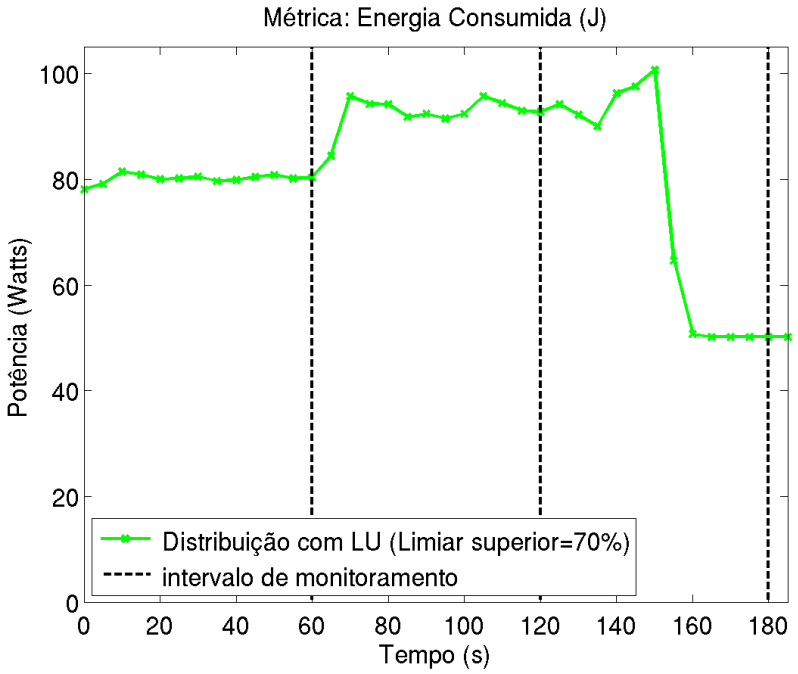


Figura 4.8: Teste de validação #2 - Impacto da abordagem de distribuição associada a política de limiar único, com limiar superior igual a 70%, no consumo de energia do ambiente de experimentação

Considerando que o cenário foi idealizado para testar o algoritmo de distribuição, demos continuidade à etapa de configuração dos parâmetros variando o limiar superior. A figura 4.9 apresenta o consumo de energia do ambiente computacional durante a execução da abordagem de distribuição associada à política de limiar único, com diferentes valores de limiar superior. Podemos observar que o comportamento da política de limiar único com limiar superior igual a 70% apresenta, a partir do instante t = 120 s, um incremento no consumo de potência que irá decair e se estabilizar apenas no instante t = 220 s. Por outro lado, a política de limiar único com limiares 80% e 90% apresentam, a partir do instante t = 120s,

um incremento no consumo de potência seguido de seu decremento até se estabilizar no instante t = 175s. Os pormenores dessa divergência de comportamento serão discutidos ainda nesta seção.

Com esse experimento, notamos que a variação nos valores atribuídos ao limiar superior impactam no consumo de energia. Assim, as combinações com limiar superior igual 80 e 90% apresentaram um desempenho melhor, em termos energéticos, que a abordagem com limiar superior igual a 70%. Visto que as 2 (duas) combinações consumiram menos energia para ajustar o mapeamento de recursos e, ainda, dissiparam a sobrecarga em um menor intervalo de tempo, alcançado uma economia de 37,5% no consumo de energia mais rapidamente. Dessa forma, podemos escolher valores maiores para obter uma abordagem de distribuição mais passiva ou valores menores para uma abordagem mais agressiva.

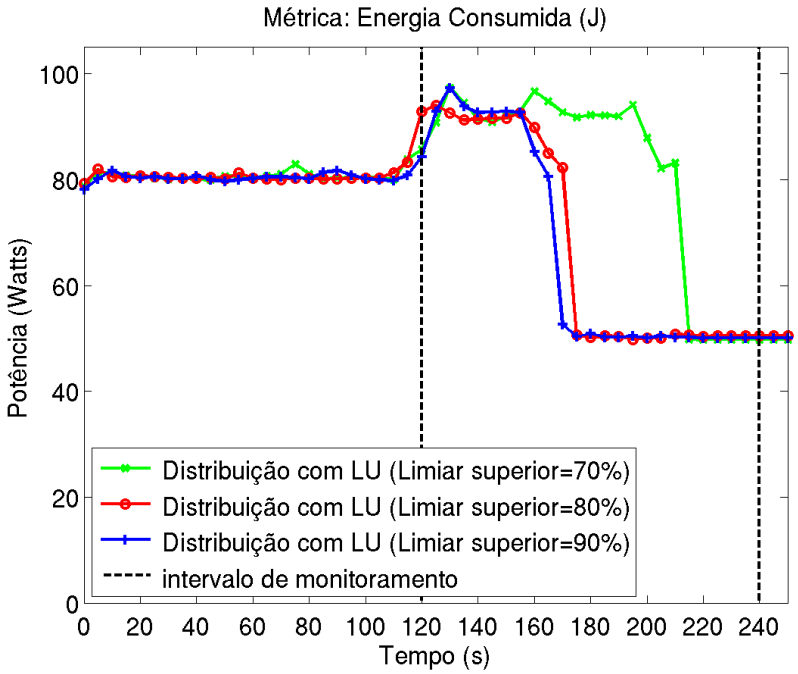


Figura 4.9: Teste de validação #2 - Impacto da abordagem de distribuição associada a políticas de limiar único, com diferentes limiares, no consumo de energia do ambiente de experimentação

Uma vez que as parâmetros foram devidamente ajustados, repetimos o teste com os novos valores a fim de avaliar o comportamento do algoritmo de distribuição. A figura 4.10 apresenta o impacto desse teste em termos do consumo de memória do ambiente de experimentação e, através dela, podemos observar o comportamento

da consolidação associada a política de limiar único com limiar superior igual a 70%. Nessa figura, ao final do primeiro intervalo de monitoramento (t = 120 s), os dados a respeito da utilização de CPU e memória são coletados e os servidores físicos classificados. Conhecido que o hospedeiro xcp-resource01 está sobrecarregado (*vide* Figura 4.7), o algoritmo de mitigação invoca a distribuição de carga que, por sua vez, desloca a VM xcp-vm-debian32-256-01 do hospedeiro xcp-resource01 para o hospedeiro xcp-resource03 a fim de dissipar a sobrecarga. No entanto, mesmo com essa migração, o percentual de utilização de memória inferior a 70% não foi alcançado. Como consequência, a distribuição desloca a VM xcp-vm-debian32-256-02 do hospedeiro xcp-resource01 para o hospedeiro xcp-resource03 e, finalmente, a sobrecarga é dissipada. Logo que não existe mais servidores sobrecarregados, o algoritmo de consolidação é invocado com o intuito de economizar energia. Este, por sua vez, inicia o processo de desligamento do xcp-resource02, visto que não possui VMs em execução.

Ao final do segundo intervalo de monitoramento (t = 240 s), observamos que o servidor xcp-resource02 foi desligado e os servidores xcp-resource01 e xcp-resource03 permanecem ligados, executando suas respectivas VMs, com o percentual de utilização de memória abaixo do limiar superior previamente estabelecido. Portanto, concluímos que o servidores foram corretamente classificados, acarretando na ação do algoritmo de distribuição que, por sua vez, migrou VMs do servidor sobrecarregado para um único hospedeiro sem promover sobrecarga. Como consequência, ainda foi possível o desligamento de um servidor, promovendo uma redução de 37,5% (*vide* figura 4.8) no consumo de energia do ambiente de experimentação.

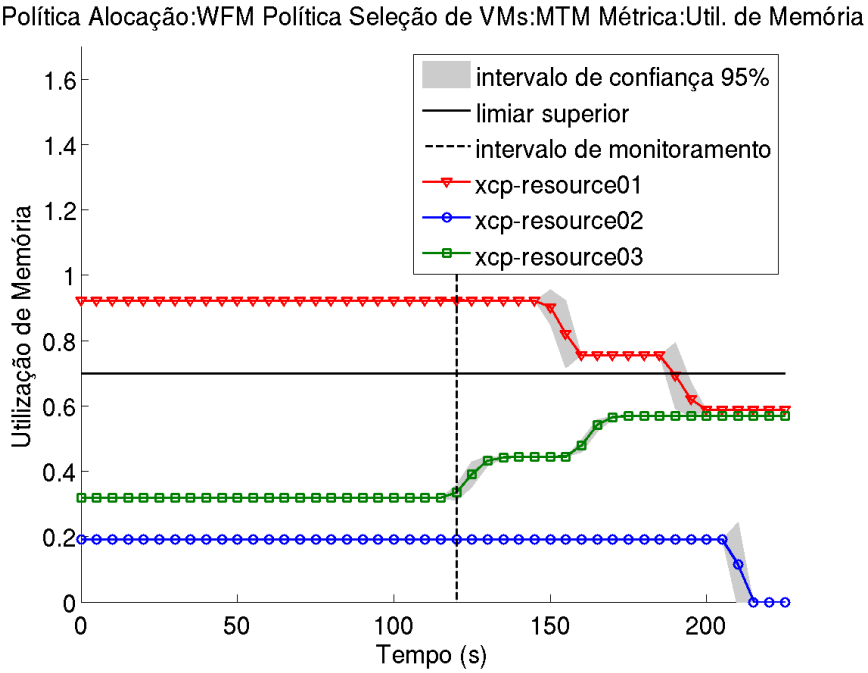


Figura 4.10: Teste de validação #2 - Impacto da abordagem de distribuição associada a política de limiar único (limiar superior = 70%) no consumo de memória do ambiente de experimentação

A figura 4.11 apresenta o impacto desse teste em termos do consumo de memória do ambiente de experimentação e, através dela, podemos observar o comportamento da consolidação associada à política de limiar único com limiar superior igual a 80%. Nessa figura, ao final do primeiro intervalo de monitoramento (t=90s), os dados a respeito da utilização de CPU e memória são coletados e os servidores físicos classificados. Conhecido que o hospedeiro xcp-resource01 está sobrecarregado (*vide* figura 4.7), o algoritmo de mitigação invoca a distribuição de carga que, por sua vez, desloca a VM xcp-vm-debian32-256-01 do hospedeiro xcp-resource01 para o hospedeiro xcp-resource03 e dissipa a sobrecarga. Logo que não existem mais servidores sobrecarregados, o algoritmo de consolidação é invocado com o intuito de economizar energia. Este, por sua vez, inicia o processo de desligamento do xcp-resource02, visto que não possui VMs em execução. A migração apresenta-se através da redução do percentual de utilização de memória no xcp-resource01 e incremento de tal percentual no xcp-resource03. O desligamento pode ser observado na figura através da redução do percentual de utilização de memória até 0 (zero), a partir do instante t = 150 s. Portanto, notamos que quando o limiar superior assume o valor maior que 80%, uma única migração é necessária para dissipar a

sobrecarga.

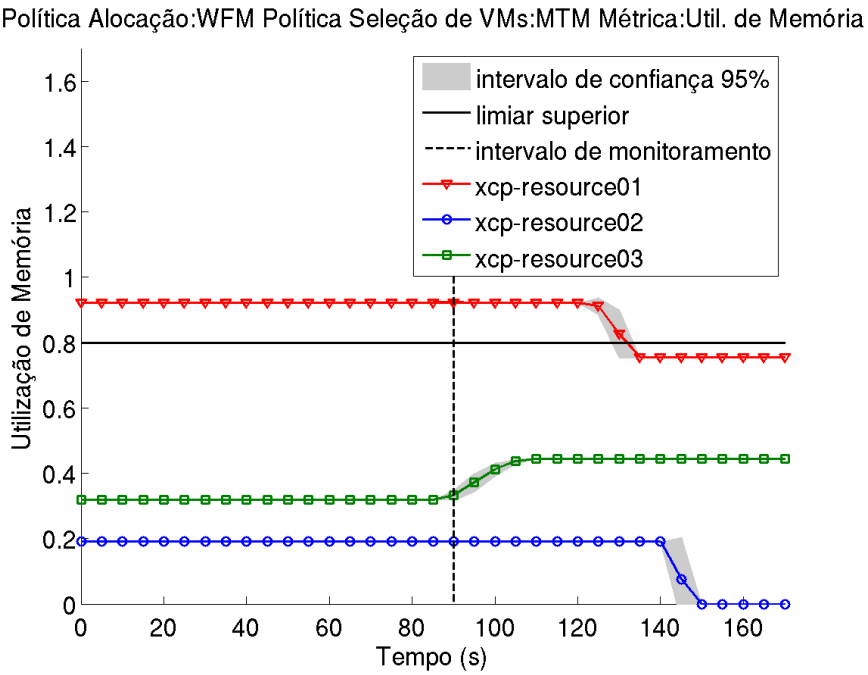


Figura 4.11: Teste de validação #2 - Impacto da abordagem de distribuição associada a política de limiar único (limiar superior = 80%) no consumo de memória do ambiente de experimentação

4.2.2 Experimento #1

Depois de validar o comportamento da estratégia proposta, realizamos um experimento cujo objetivo consiste em submeter os hospedeiros a variações na carga a fim de avaliar o comportamento das abordagens de distribuição e consolidação das VMs atuando em conjunto. Para tanto, nós desenvolvemos uma carga de trabalho orientada a CPU capaz de variar seu comportamento de acordo com um intervalo de valores previamente estabelecido. Este experimento visa, ainda, comparar as heurísticas de alocação BF, BFM, WF e WFM e as políticas de limiar único e duplo. Os resultados correspondem a uma média de 5 (cinco) repetições, cada uma delas com duração de 12 minutos, com intervalo de confiança de 95%. O mapeamento inicial das VMs em hospedeiros é ilustrado na figura 4.12. O intervalo de monitoramento foi configurado para três minutos e a janela de monitoramento foi estabelecida em 3 amostras, i.e., as três últimas amostras de utilização dos recursos são levadas em consideração na classificação dos servidores físicos.

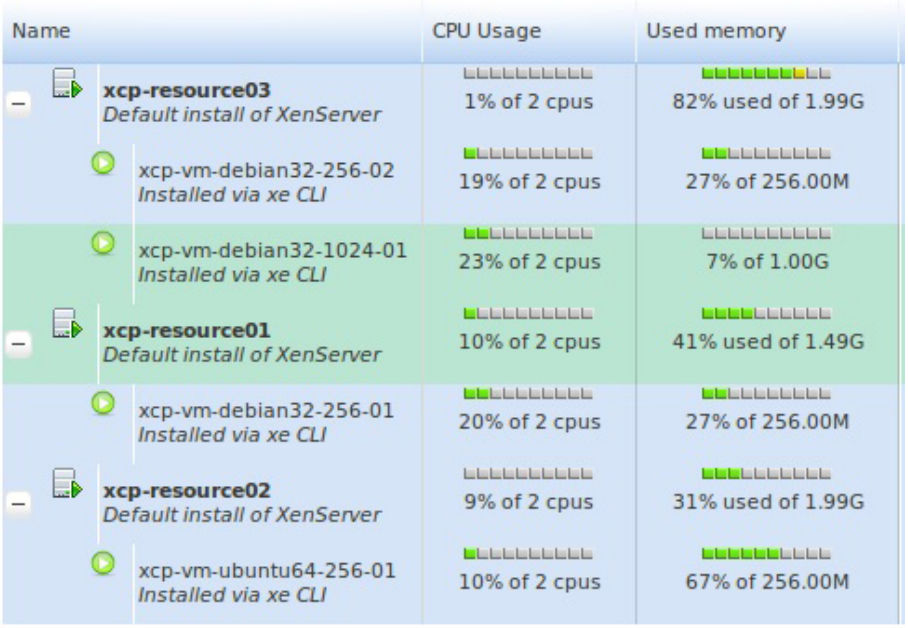


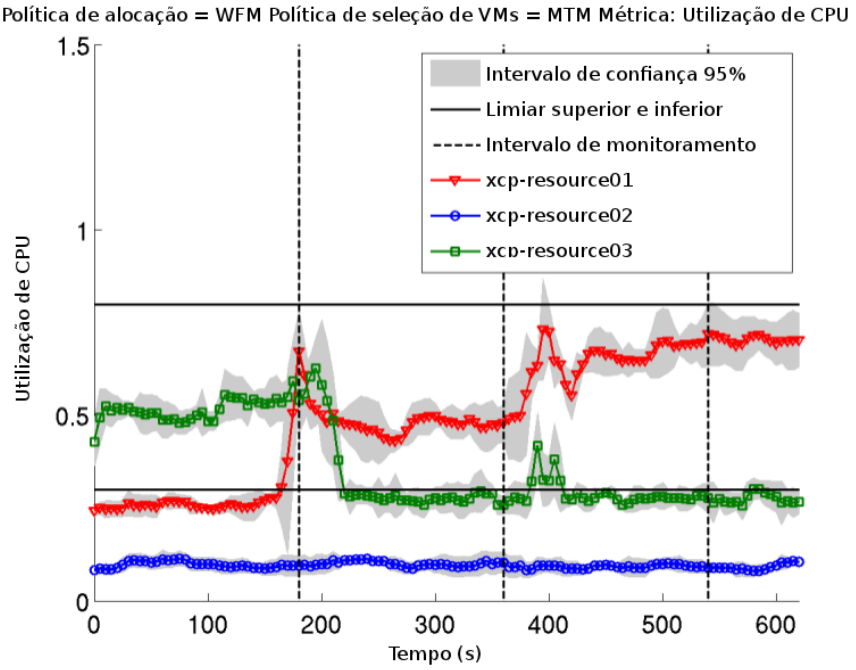
Figura 4.12: Cenário de Experimentação #1 (medições reais)

Na primeira etapa do experimento, a heurística de alocação das VMs WFM foi combinada à política de limiar duplo para avaliação. A figura 4.13 apresenta o impacto dessa combinação em termos do consumo de CPU e memória do ambiente de experimentação. Ao final do primeiro intervalo de monitoramento (t = 180 s), os dados a respeito da utilização de CPU e memória são coletados e os servidores físicos classificados. Uma vez que o hospedeiro xcp-resource03 está sobrecarregado, o algoritmo de mitigação invoca o balanceamento de carga que, por sua vez, desloca a VM xcp-vm-debian32-256-02 do hospedeiro xcp-resource03 para o hospedeiro xcp-resource01 a fim de dissipar a sobrecarga. Isso é justificado, pois o xcp-resource01, apesar de apresentar uma capacidade disponível inferior ao xcp-resource02, possui um menor consumo de energia, tornando o quociente capacidade disponível sobre consumo de energia maior para o xcp-resource01 em relação ao xcp-resource02. Esta migração pode ser observada nas figuras 4.13(a) e 4.13(b), através da redução do percentual de utilização de CPU e memória no xcp-resource03 e incremento de tais percentuais no xcp-resource01. Após a migração, todos os hospedeiros apresentaram seus percentuais de utilização de CPU e memória dentro dos limiares inferior e superior. Portanto, para este caso não existe a necessidade de invocar o algoritmo de mitigação de acordo com a política de limiar duplo.

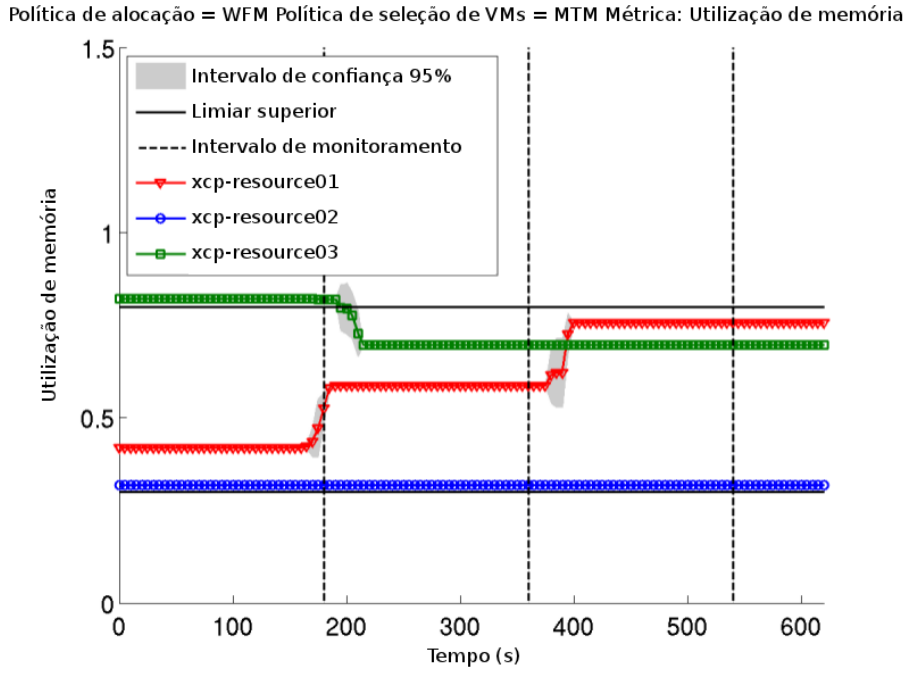
Ao final do segundo intervalo de monitoramento (t = 360 s), os dados são coletados e os hospedeiros xcp-resource01, xcp-resource02 e xcp-resource03 são classificados como normais. No instante t = 400 s, a carga do xcp-resource01 cresce mas não provoca sobrecarga. Ao final do terceiro intervalo de monitoramento (t = 540 s), os dados são coletados e os hospedeiros são classificados como normais. Visto que nenhum ponto de sobrecarga e/ou ócio foi detectado, o algoritmo de mitigação não é invocado e, consequentemente, nenhuma migração é realizada.

O mesmo procedimento foi repetido para as heurísticas de alocação BF, BFM e WF, i.e., elas foram combinadas à política de limiar duplo e seus impactos avaliados. Um comportamento idêntico ao WFM foi alcançado por essas heurísticas, exceto a WF, que ao final do primeiro intervalo de monitoramento, desloca a VM xcp-vm-debian32-256-02 do hospedeiro xcp-resource03 para o hospedeiro xcp-resource02 a fim de dissipar a sobrecarga. Isso é justificado pois o WF ignora o consumo de energia apresentado pelo hospedeiro, considerando apenas sua capacidade disponível nas tomadas de decisão. Uma vez que o hospedeiro xcp-resource02 apresenta uma capacidade disponível maior que o xcp-resource01, o primeiro é selecionado como hospedeiro de destino. Apesar desta diferença, os resultados apresentados pela heurística WF em termos de número de migrações, percentual de utilização de recursos abaixo do limiar superior e consumo de energia são semelhantes aos resultados alcançados pelas demais heurísticas, quando combinadas à política de limiar duplo. Isto pode ser observado nas figuras 4.17(a), 4.17(b) e 4.18, respectivamente.

Na segunda etapa do experimento, a heurística WFM foi combinada à política de limiar único para avaliação. A figura 4.14 apresenta o impacto dessa combinação em termos do consumo de CPU e memória do ambiente de experimentação. Da mesma forma como procedemos na 1a etapa do experimento, em t = 180s, i.e. ao final do primeiro intervalo de monitoramento, os dados a respeito da utilização dos recursos são coletados e os servidores classificados. Uma vez que o hospedeiro xcp-resource03 está sobrecarregado, visto que seu percentual de utilização de memória está acima do limiar superior (vide 4.14(b)), o algoritmo de mitigação invoca o balanceamento de carga o qual, por sua vez, desloca a VM xcp-vm-debian32-256-02 do xcp-resource03 para o xcp-resource01 a fim de dissipar a sobrecarga. Esta migração pode ser observada nas figuras 4.14(a) e 4.14(b), através da redução do percentual de utilização de CPU e memória no xcp-resource03 e incremento de tais percentuais



(a)



(b)

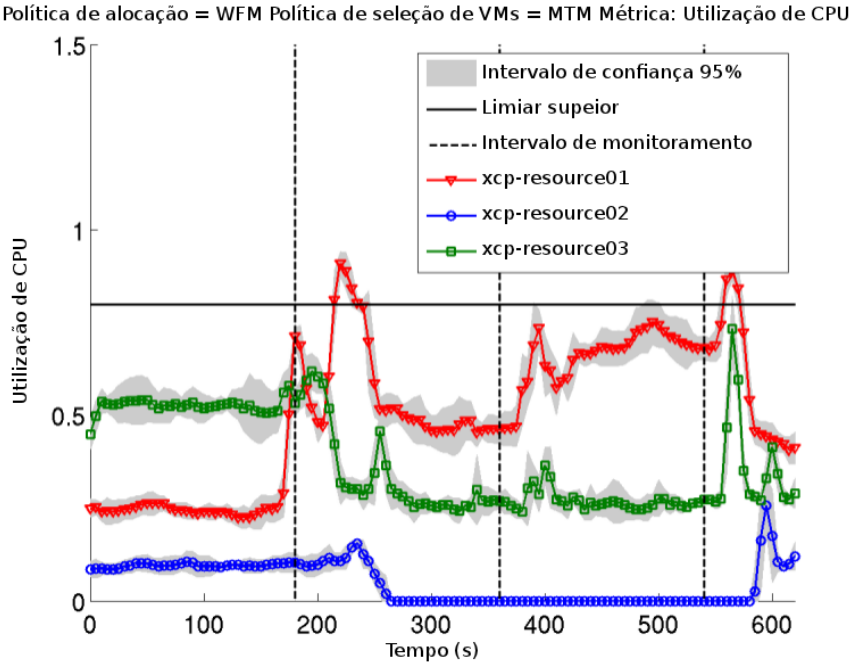
Figura 4.13: Experimento #1 - Impacto da heurística WFM com limiar duplo no consumo de processamento e memória do ambiente (a) CPU; (b) Memória

no xcp-resource01. Visto que a sobrecarga foi eliminada e a política de limiar é único, o algoritmo de balaceamento chama o algoritmo de consolidação que migra a VM xcp-vm-ubuntu64-256-01 do hospedeiro xcp-resource02 para o xcp-resource01. Em seguida, o xcp-resource02 é desligado e a consolidação finalizada, a fim de economizar energia. Essa etapa de consolidação é apresentada na figura 4.14(a), através de um incremento no percentual de utilização de CPU do xcp-resource02, seguido de sua redução até 0%. A consolidação também pode ser observada na figura 4.14(b) conforme a redução do percentual de utilização de memória do hospedeiro xcp-resource02 até alcançar 0%. O incremento do percentual de utilização de CPU do xcp-resource02 antes de seu desligamento é decorrente do consumo de CPU provocado pelos processos de migração da VM xcp-vm-ubuntu64-256-01 e desligamento do próprio servidor.

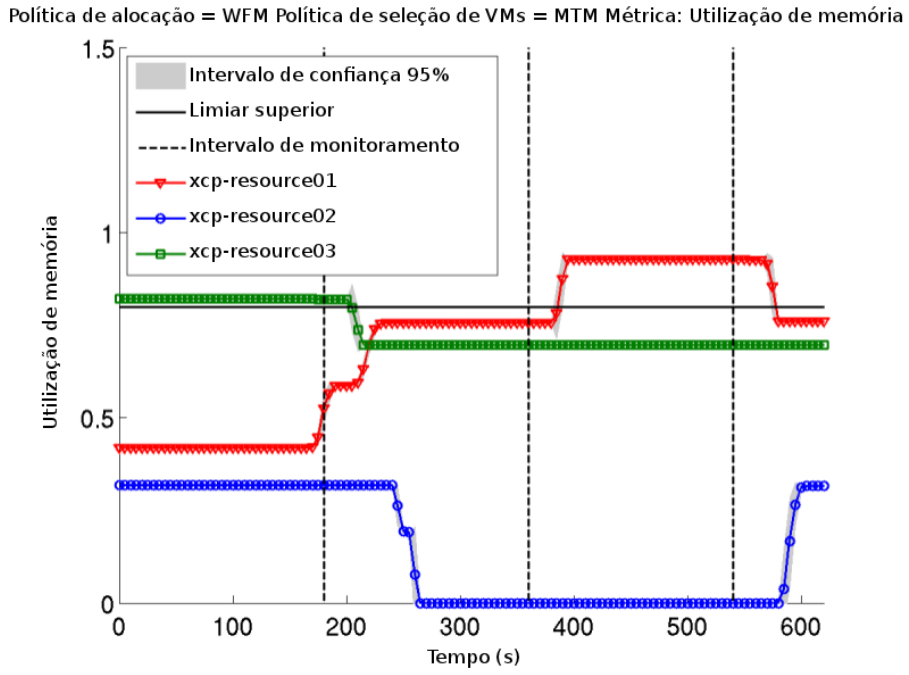
Ao final do segundo intervalo de monitoramento (t= 360 s), os dados são coletados e os hospedeiros xcp-resource01 e xcp-resource03 são classificados como normais. Uma vez que a política de limiar único foi selecionada, o algoritmo de consolidação é invocado. No entanto, nenhuma migração é realizada visto que o xcp-resource01 não possui recursos suficientes para hospedar a VM presente no xcp-resource03 e este, por sua vez, não pode hospedar as VMs do xcp-resource01 sem ocasionar uma sobrecarga. No instante 400 s, a carga do xcp-resource01 cresce e excede o limiar superior, caracterizando uma sobrecarga.

Ao final do terceiro intervalo de monitoramento (t= 540 s), os dados são coletados e os hospedeiros xcp-resource01 e xcp-resource03 são classificados como sobrecarregado e normal, respectivamente. Visto que o xcp-resource01 está sobrecarregado, o algoritmo de mitigação invoca o balanceamento de carga que, por sua vez, busca um hospedeiro de destino entre os servidores ligados. Como o xcp-resource03 não possui recursos suficientes para hospedar qualquer VM do xcp-resource01 sem promover uma nova sobrecarga, o algoritmo de balanceamento busca o hospedeiro de destino na lista de hospedeiros desligados e, então, liga o hospedeiro xcp-resource02. Em seguida, desloca a VM xcp-vm-ubuntu64-256-02 do xcp-resource01 para o xcp-resource02 a fim de dissipar a sobrecarga. Esta migração pode ser observada nas figuras 4.14(a) e 4.14(b), através da redução do percentual de utilização de CPU e memória no xcp-resource01 e incremento de tais percentuais no xcp-resource02.

O mesmo procedimento foi repetido para as heurísticas de alocação BF, BFM



(a)



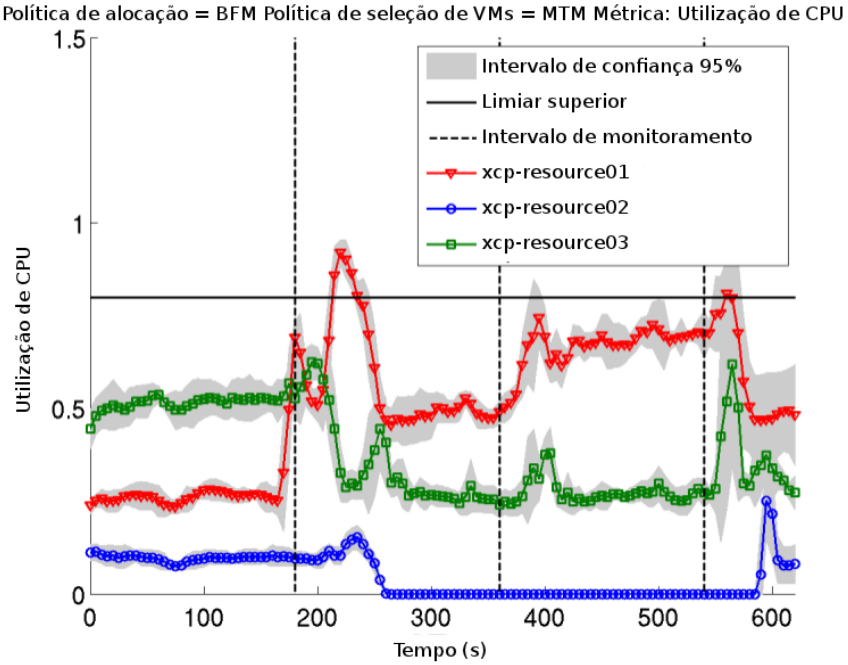
(b)

Figura 4.14: Experimento #1 - Impacto da heurística WFM com limiar único no consumo de (a) CPU; (b) Memória

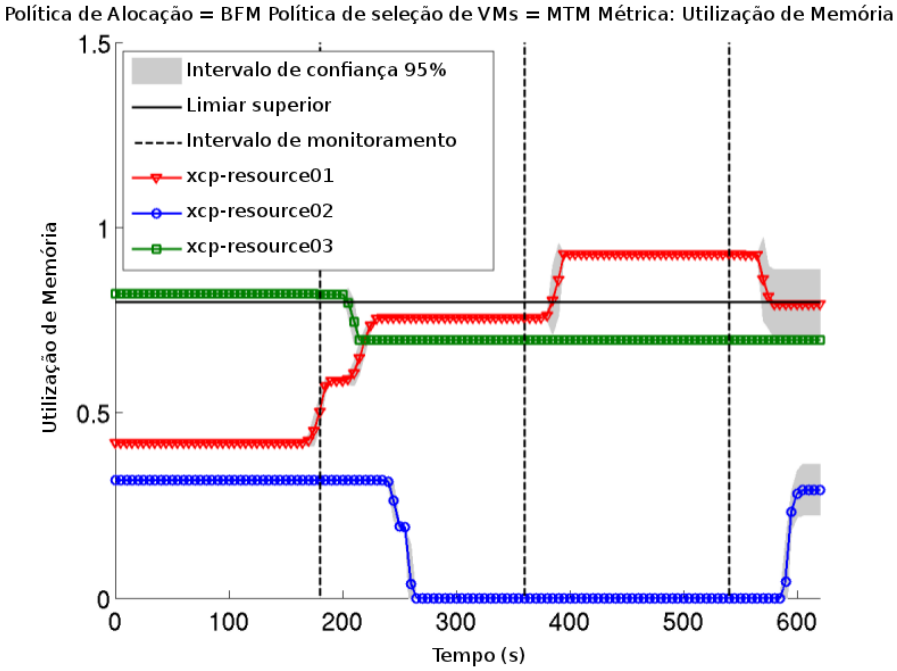
e WF, ou seja, elas também foram combinadas à política de limiar único e seus impactos avaliados. Um comportamento idêntico ao WFM foi alcançado pelas heurísticas BF e BFM. Essa semelhança é ilustrada na figura 4.15, que apresenta o impacto da heurística BFM associada a política de limiar único no consumo de processamento e memória do ambiente.

Em contrapartida, o comportamento apresentado pela WF diverge das demais heurísticas, pois no instante t = 180 s desloca a VM xcp-vm-debian32-256-02 do hospedeiro xcp-resource03 para o xcp-resource02 a fim de dissipar a sobrecarga, como pode ser observado na figura 4.16. Observamos na etapa de consolidação seguinte que a VM presente no xcp-resource01 é migrada para o xcp-resource02 e o primeiro é desligado a fim de economizar energia. Percebemos, em vista disso, que o hospedeiro desligado na fase de consolidação é o xcp-resource-01 ao invés do xcp-resource02. Apesar desta diferença no comportamento da WF não impactar no número de migrações em relação as demais heurísticas (vide figura 4.17(a)), ela impacta nas métricas percentual de utilização de recursos abaixo do limiar superior e consumo de energia, conforme pode ser observado nas figuras 4.17(b) e 4.18, respectivamente.

Visto que o processo migratório e as sobrecargas provocam uma degradação considerável no desempenho do sistema, podendo promover até quebras de SLAs, monitoramos a quantidade de migrações e o percentual de utilização dos recursos abaixo do limiar superior apresentados pelas heurísticas BF, BFM, WF e WFM associadas às políticas de limiar único e duplo, conforme apresentado na figura 4.17. Note na figura 4.17(a) que o número de migrações apresentados pelas heurísticas de alocação com limiar duplo é menor que o número apresentado pelas mesmas heurísticas com limiar único. Isso ocorre devido à política de limiar único ser caracterizada por uma consolidação mais agressiva, ou seja, existe um maior número de casos em que é possível realizar consolidação e, consequentemente, ocorre um maior número de migrações. A vantagem de uma política de consolidação mais agressiva consiste em desligar um maior número de servidores e, assim, economizar mais energia (vide figura 4.18). No entanto, se a carga de trabalho nesse sistema começa a crescer, haverá um número menor de hospedeiros para comportá-la e, portanto, uma maior probabilidade de ocorrer sobrecargas. Isso é corroborado pelos resultados apresentados na figura 4.17(b), onde o percentual de recursos abaixo do limiar superior é maior para as heurísticas associadas ao limiar duplo em relação às

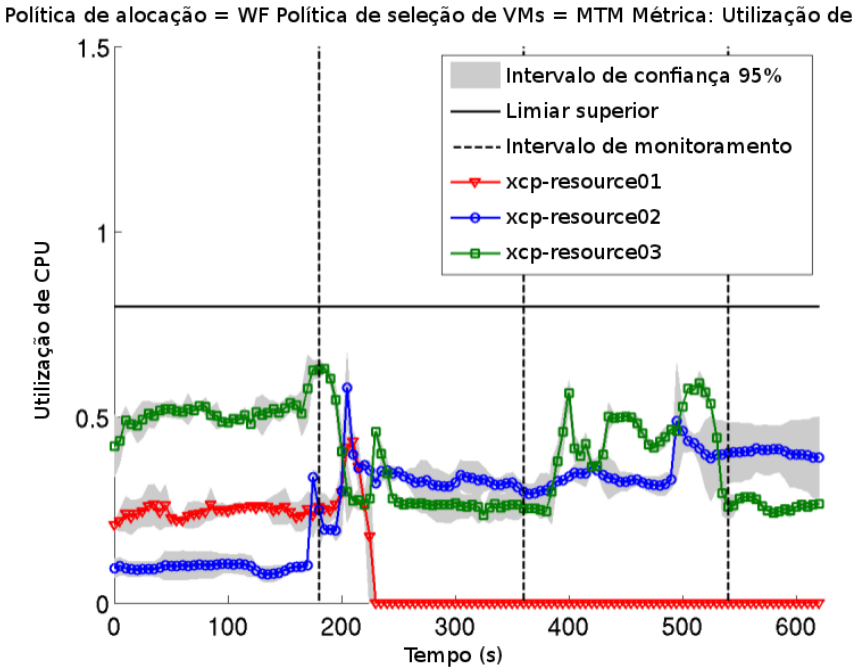


(a)

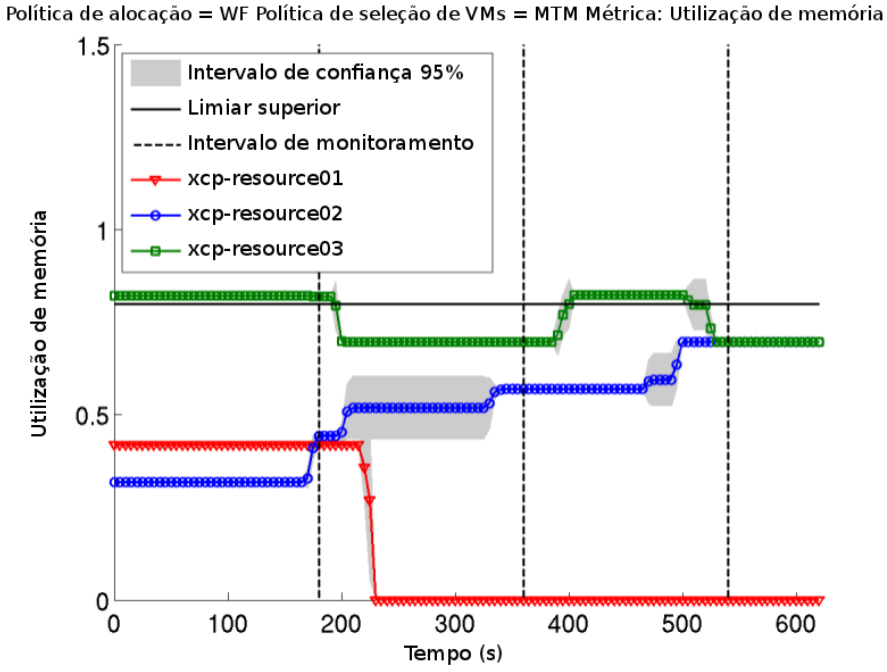


(b)

Figura 4.15: Experimento #1 - Impacto da heurística BFM com limiar único no consumo de (a) CPU; (b) Memória



(a)



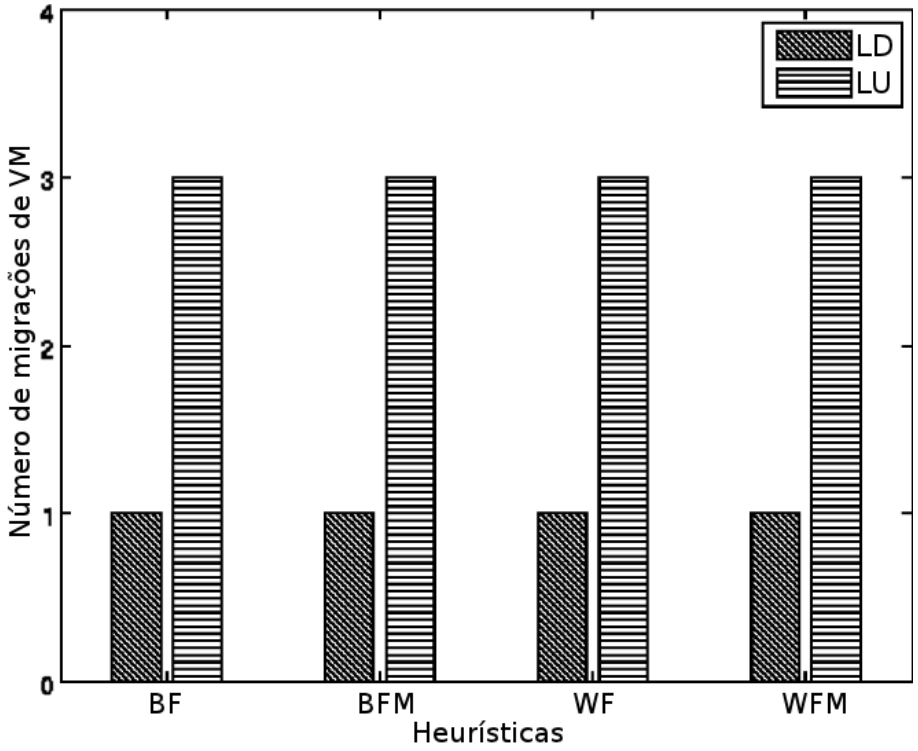
(b)

Figura 4.16: Experimento #1 - Impacto da heurística WF com limiar único no consumo de (a) CPU; (b) Memória

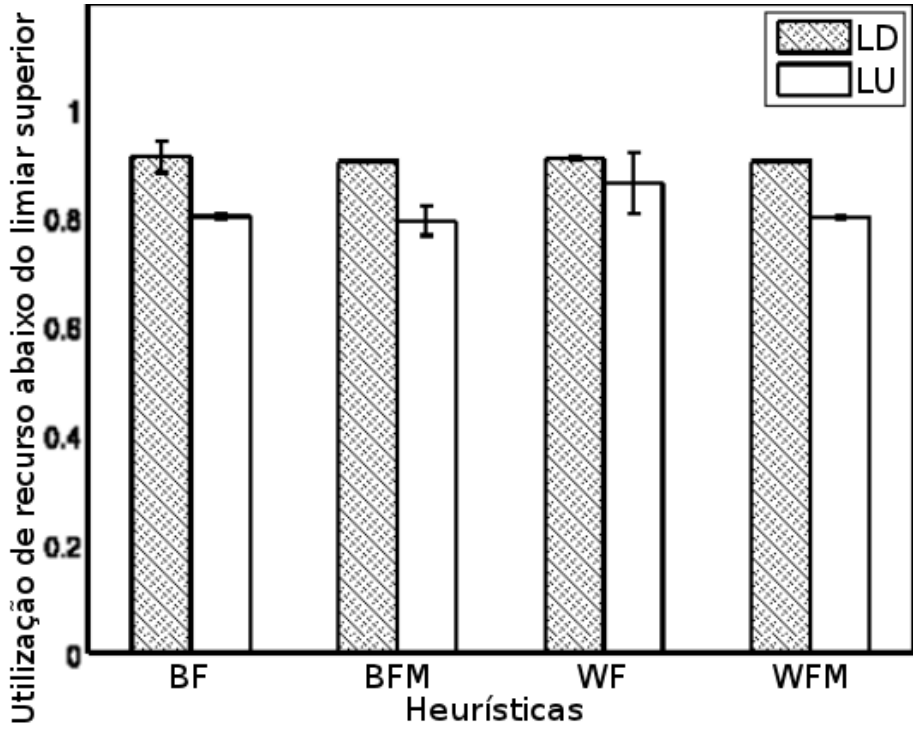
mesmas heurísticas associadas ao limiar único.

Através do modelo de potência (*vide* Eq. 2.1), apresentamos na figura 4.18 a comparação, em termos energéticos, das heurísticas de alocação das VMs BF, BFM, WF e WFM combinadas com políticas de limiar único (4.18(a)) e duplo (4.18(b)). No que se refere às políticas de limiar, percebemos uma clara diferença entre único e duplo, sendo a maior delas apresentada pela heurística WFM: uma redução de 15.5% no consumo total de energia do limiar único em relação ao duplo. E a menor diferença apresentada pela heurística WF: uma redução de 14.2% no consumo total de energia do limiar único em relação ao duplo. Essa diferença é justificada por uma etapa de consolidação presente nas heurísticas com limiar único que permitiu a estratégia economizar energia sem comprometer o balanceamento da carga. Portanto, a política limiar único propicia um maior economia de energia que a de limiar duplo.

De acordo com as etapas do experimento apresentadas nesta seção, as heurísticas BF, BFM e WFM comportaram-se de forma similar, o que se refletiu diretamente no consumo de energia apresentado por cada abordagem (*vide* figuras 4.18(a) e 4.18(b)). Todavia, a heurística WF apresenta um comportamento diferenciado, o qual não impacta em seu consumo de energia (*vide* figura 4.18(b)), visto que no limiar duplo não ocorre consolidação para nenhuma das heurísticas envolvidas. Em contrapartida, conforme ilustrado na figura 4.18(a), há uma etapa de consolidação e percebemos que a heurística WF apresenta um desempenho inferior às demais heurísticas, apresentando um consumo de energia 6% maior que a WFM na etapa de consolidação. Isso ocorre porque a WF considera apenas a capacidade disponível de um hospedeiro no momento da consolidação e distribuição das VMs. Portanto, o WF desligou o hospedeiro de menor capacidade disponível naquele momento (xcp-resource01), ao passo que algoritmos propostos, WFM e BFM desligaram o xcp-resource02, o qual possuía não só uma maior capacidade disponível mas também o maior consumo de energia, fazendo uma escolha mais eficiente em termos energéticos. Assim como a WF, a heurística BF leva em consideração apenas a capacidade disponível do servidor. No entanto, ela seleciona aquele com a maior capacidade disponível para ser desligado, nesse caso, o xcp-resource02, por isso obteve resultados semelhantes às heurísticas modificadas. Esse resultado reforça a nossa proposta de modificação dos algoritmos WF e BF clássicos, os quais passaram a considerar o *trade-off* entre capacidade disponível e custo energético e, com isso,



(a)



(b)

Figura 4.17: Experimento #1 - Impacto das políticas de limiar único e duplo nas heurísticas de alocação de VMs em termos de (a) Número de Migrações; (b) Percentual de Utilização Abaixo do Limiar Superior

obtiveram um resultado superior ao WF.

4.2.3 Experimento #2

O cenário de experimentação #2 é ilustrado na figura 4.19. O objetivo deste cenário de testes é avaliar o impacto das políticas de seleção das VMs MVC e MTM no comportamento da estratégia e compará-las entre si. Para tanto, além de fazer uso da carga de trabalho orientada a CPU, desenvolvemos uma carga de trabalho orientada a memória capaz de variar seu comportamento de acordo com um volume de memória a ser consumido previamente estabelecido. A heurística de alocação WFM combinada ao limiar único foi fixada nessa fase. Os resultados correspondem a uma média de 5 (cinco) repetições, cada uma delas com duração de 12 minutos, com intervalo de confiança de 95%.

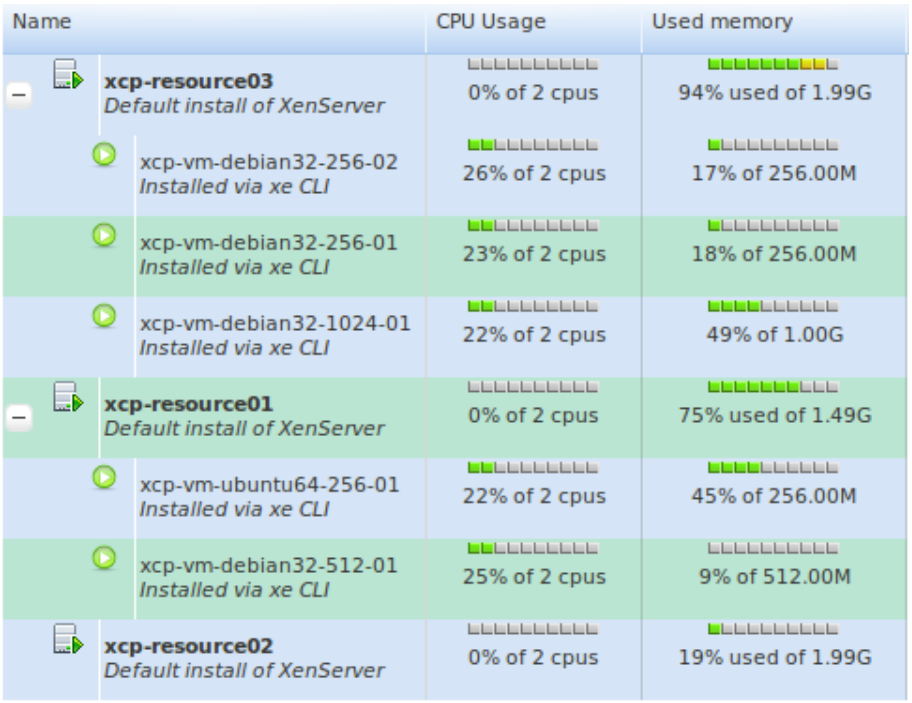
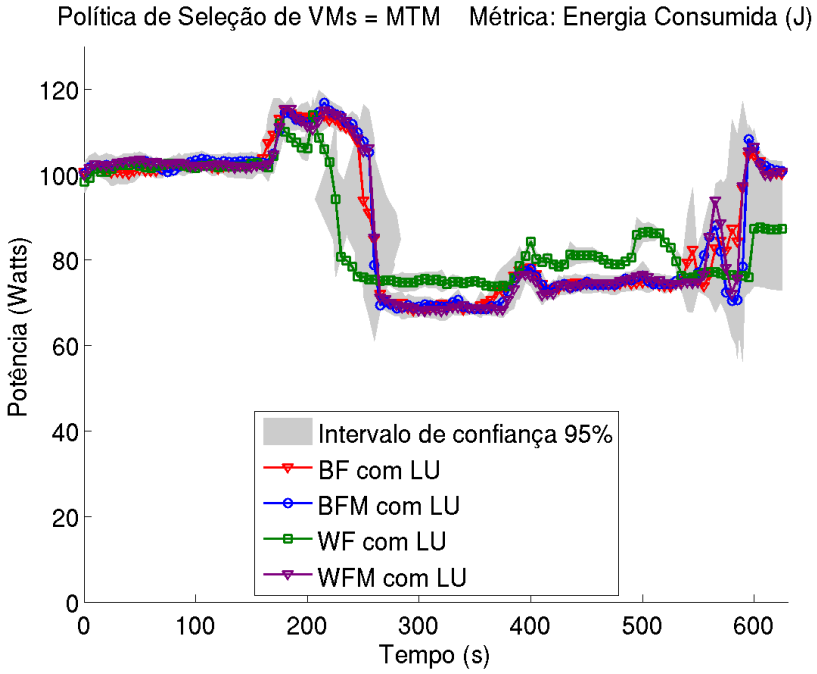
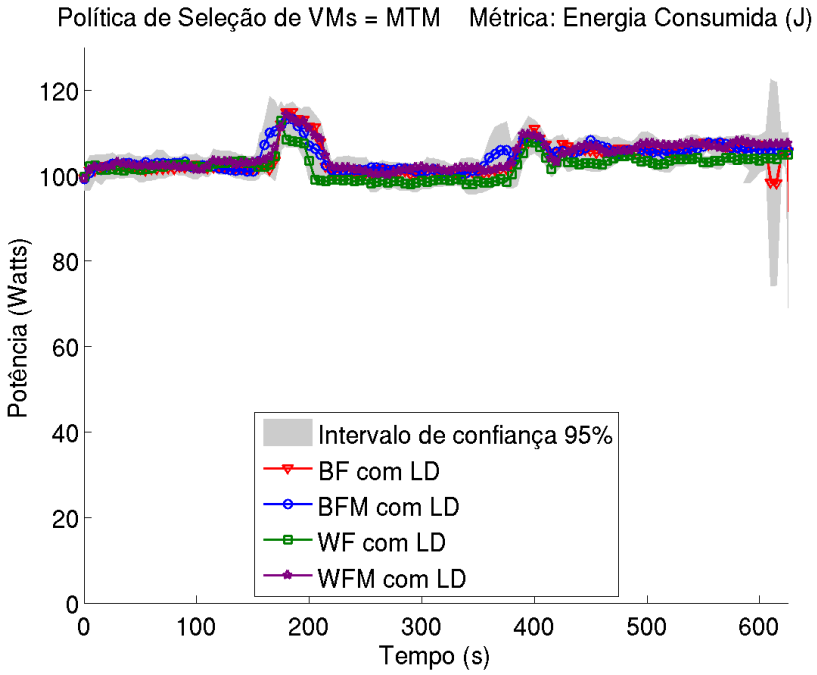


Figura 4.19: Cenário de Experimentação #2 (medições reais)

Na primeira etapa do experimento, a política de seleção das VMs MVC foi utilizada para avaliação. A figura 4.20 apresenta o impacto dessa combinação em termos do consumo de CPU e memória do ambiente de experimentação. Ao final do primeiro intervalo de monitoramento (t =180 s), os dados a respeito da utilização de CPU e memória são coletados e os hospedeiros classificados. Uma vez que o hospedeiro xcp-resource03 está sobrecarregado, o algoritmo de mitigação invoca o



(a)



(b)

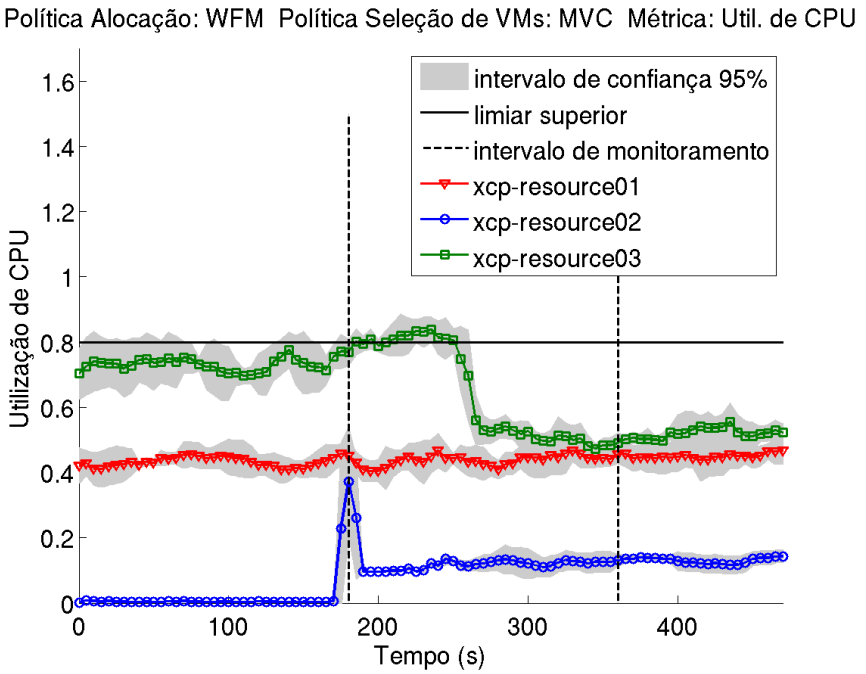
Figura 4.18: Experimento #1 - Impacto das heurísticas de alocação de VMs em termos de consumo de energia (a) Limiar único; (b) Limiar Duplo

balanceamento de carga que, por sua vez, seleciona uma ou mais VMs a serem migradas a fim de dissipar a sobrecarga. Portanto, a VM xcp-vm-debian32-1024-01 é deslocada do xcp-resource03 para o xcp-resource02. Isso é justificado uma vez que a política MVC seleciona a VM com maior volume de carga (vide Fórmula 3.1) para participar do processo migratório. Esta migração pode ser observada nas figuras 4.20(a) e 4.20(b), através da redução do percentual de utilização de CPU e memória no hospedeiro xcp-resource03 e incremento de tais percentuais no xcp-resource02. Após essa migração, todos os servidores apresentaram seus percentuais de utilização de CPU e memória abaixo do limiar superior.

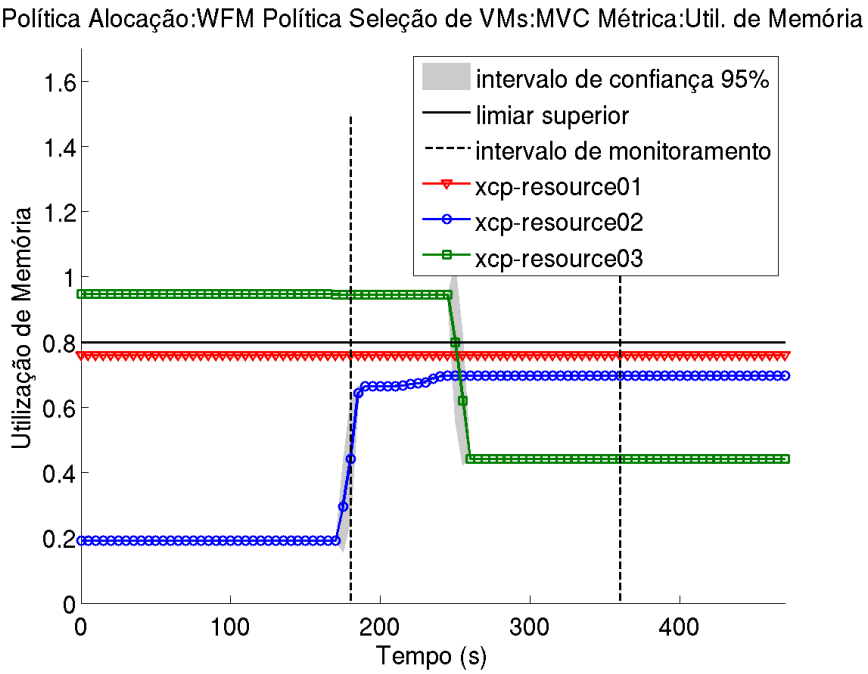
Ao final do segundo intervalo de monitoramento (t = 360 s), os dados são coletados e os hospedeiros xcp-resource01, xcp-resource02 e xcp-resource03 são classificados como normais. Visto que nenhum ponto de sobrecarga foi detectado e não é possível realizar consolidação, nenhuma outra migração é realizada.

Na segunda etapa do experimento, a política de seleção das VMs MTM foi utilizada para avaliação. A figura 4.21 apresenta o impacto dessa combinação em termos do consumo de CPU e memória do ambiente de experimentação. Ao final do primeiro intervalo de monitoramento (t =180 s), os dados a respeito da utilização de CPU e memória são coletados e os servidores classificados. Uma vez que o hospedeiro xcp-resource03 está sobrecarregado, o algoritmo de mitigação invoca o balanceamento de carga que, por sua vez, seleciona uma ou mais VMs a serem migradas a fim de dissipar a sobrecarga. Portanto, a VM xcp-vm-debian32-256-01 é deslocada do xcp-resource03 para o xcp-resource02. Isso é justificado uma vez que a política MTM seleciona a VM com menor tamanho de memória para participar do processo migratório. No entanto, mesmo com essa migração, a sobrecarga não foi dissipada. Logo, a política de seleção MTM é acionada novamente e seleciona a VM xcp-vm-debian32-256-02, que possui o menor tamanho de memória, visando a migração para o hospedeiro xcp-resource02. Estas migrações podem ser observadas nas figuras 4.21(a) e 4.21(b), através da redução do percentual de utilização de CPU e memória no xcp-resource03 e incremento de tais percentuais no xcp-resource02. Após a segunda migração, todos os servidores apresentaram seus percentuais de utilização de CPU e memória abaixo do limiar superior e, portanto, a sobrecarga é dissipada.

Ao final do segundo intervalo de monitoramento (t = 360 s), os dados são coletados e os hospedeiros xcp-resource01, xcp-resource02 e xcp-resource03 são

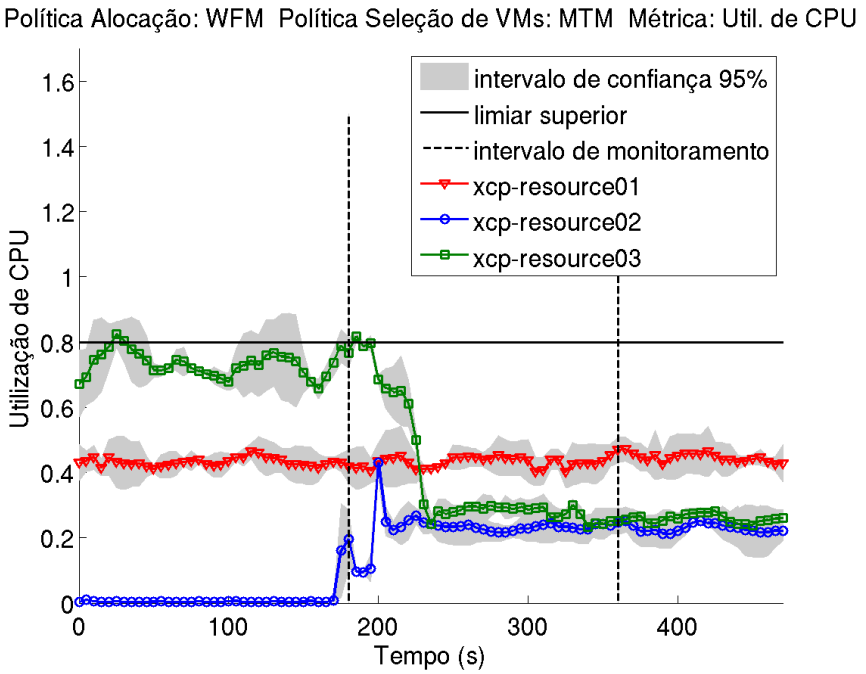


(a)

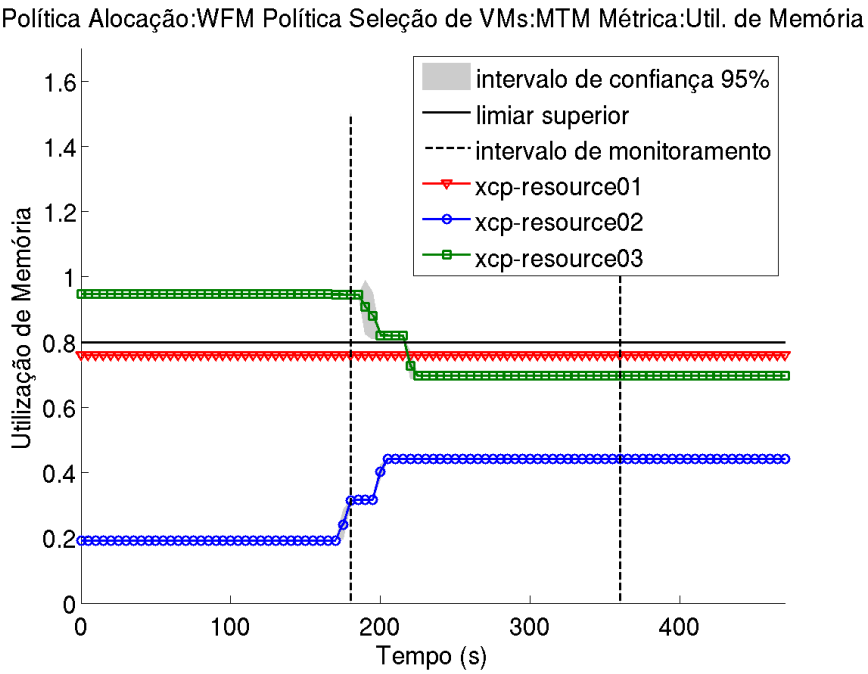


(b)

Figura 4.20: Experimento #2 - Impacto da política MVC associada à heurística WFM em termos de consumo de processamento e memória do ambiente (a) CPU; (b) Memória



(a)



(b)

Figura 4.21: Experimento #2 - Impacto da política MTM associada à heurística WFM em termos de consumo de processamento e memória do ambiente (a) CPU; (b) Memória

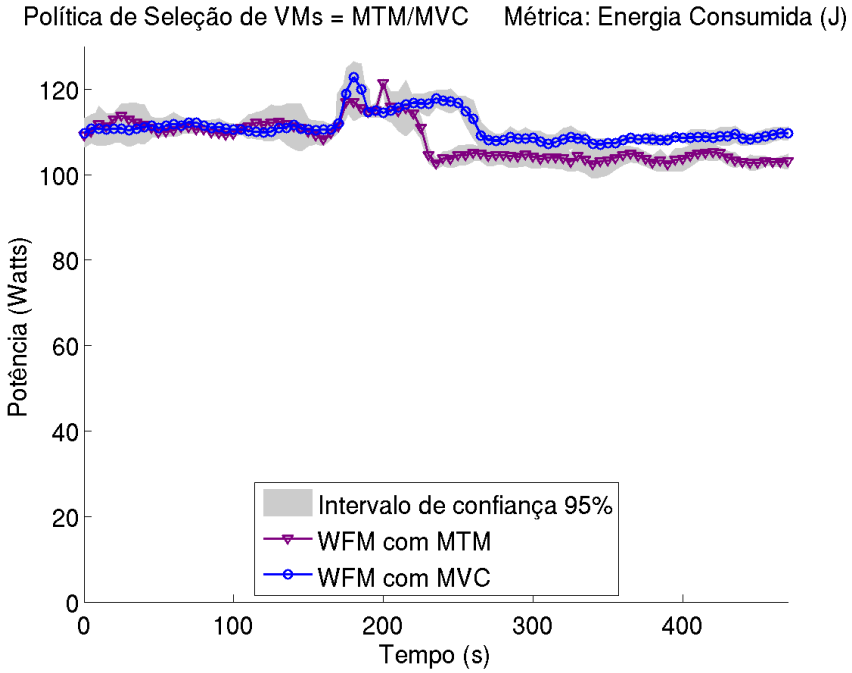


Figura 4.22: Impacto das políticas MVC e MTM na energia consumida pelo ambiente de experimentação

classificados como normais. Visto que nenhum ponto de sobrecarga foi detectado e não é possível realizar consolidação, nenhuma outra migração é realizada. Observamos que a política MVC dissipa a sobrecarga em uma única migração, enquanto a política MTM necessita de duas migrações. Portanto, a política de seleção de VMs MVC apresenta um desempenho melhor que a MTM em termos de número de migrações. No entanto, obsevamos na figura 4.21, que a política MTM gasta menos tempo para dissipar a sobrecarga que a MVC. Isso ocorre porque as duas VMs migradas somavam 512 MB de memória RAM a ser transferido através da rede, enquanto a VM selecionada pela política MVC possuia 1024 MB de RAM a ser tranferido pela rede durante o processo migratório. Isso reflete diretamente no percentual de utilização de recursos abaixo do limiar superior. A política MTM apresenta 86.33% da utilização dos recursos abaixo do limiar superior, enquanto a política MVC apresenta 83.33% . Portanto, a política MTM apresentou um melhor desempenho em relação a MVC no que diz respeito ao percentual de utilização dos recursos abaixo do limiar superior. Apesar da divergência no comportamento das políticas MVC e MTM, elas impactaram de forma semelhante no consumo de energia apresentado pela estratégia, conforme ilustrado na figura 4.22.

4.3 Conclusão

A proposta foi detalhadamente verificada e validada por medições em um ambiente real composto por servidores virtualizados heterogêneos. O desempenho da proposta é avaliado em 4 (quatro) cenários distintos a partir das métricas utilização de CPU, utilização de memória, número de migrações e consumo de energia.

Os testes de validação (Seção 4.2.1) mostraram de forma geral que o algoritmo de monitoramento se comportou conforme esperado, classificando corretamente os servidores físicos. O teste de validação #1 (Seção 4.2.1.1) mostrou que o algoritmo de consolidação associado à política de LU foi capaz de reduzir, em mais de 50%, o consumo de energia do cenário em que foi validado. O teste de validação realizado com o algoritmo de consolidação associado à política de LD mostrou que a variação nos valores atribuídos ao limiar inferior impactam diretamente no comportamento da estratégia e, consequentemente, no consumo de energia. Dessa forma, podemos atribuir valores maiores para obter uma abordagem de consolidação mais agressiva ou valores menores para uma abordagem mais passiva. O teste de validação #2 (Seção 4.2.1.2) mostrou que algoritmo de distribuição das VMs se mostrou apto a dissipar as sobrecargas. Além disso, notamos que a variação nos valores atribuídos ao limiar superior impactam no consumo de energia. Assim, as combinações com limiar superior igual 80 e 90% apresentaram um desempenho melhor, em termos energéticos, que a abordagem com limiar superior igual a 70% pois alcançaram mais rapidamente uma economia de 37,5% no consumo de energia.

Os resultados de medições (seções 4.2.2 e 4.2.3) mostraram que: (1) a política de limiar único apresentou um desempenho melhor que a de limiar duplo em termos energéticos, promovendo uma redução de até 15.5% no consumo total de energia em relação ao limiar duplo; (2) a modificação proposta na heurística WF apresentou uma redução de 6% no consumo de energia durante a etapa de consolidação em relação à mesma heurística sem alterações. As demais heurísticas (BF e BFM) associadas as políticas de limiar único e duplo apresentaram resultados semelhantes aos alcançados pela WFM;

Capítulo 5

Considerações Finais

Nesta dissertação, propomos e desenvolvemos uma estratégia para alocação dinâmica de máquinas virtuais capaz de realizar uso efetivo dos recursos computacionais a fim de reduzir o consumo de energia sem comprometer os requisitos de desempenho do sistema. Na Seção 5.1, são apresentadas conclusões obtidas a partir dos resultados do capítulo anterior e a Seção 5.2 lista as perspectivas de trabalhos futuros que podem ser realizados a partir desta dissertação.

5.1 Conclusões

O principal objetivo com o desenvolvimento da estratégia de alocação dinâmica de VMs consiste em permitir que os recursos de ambientes computacionais virtualizados sejam gerenciados de modo a economizar energia sem desprezar as sobrecargas que comprometem o desempenho dos serviços ofertados, as quais podem provocar violações de SLA.

Para que esse objetivo fosse alcançado, concebemos e desenvolvemos dois algoritmos principais: algoritmo de monitoramento e algoritmo de mitigação. O algoritmo de monitoramento conta com as políticas de limiar único e duplo para identificar a necessidade de ajuste nos recursos. Uma vez que essa necessidade é identificada, o algoritmo de mitigação é responsável por realizar o ajuste no mapeamento de recursos, através de dois outros algoritmos: um de consolidação e outro de distribuição das VMs. A consolidação das VMs é realizada a fim de economizar energia, desde que uma nova sobrecarga no sistema não seja criada e uma sobrecarga já existente seja dissipada. Portanto, esses dois algoritmos atuam

em conjunto levando em consideração o *trade-off* entre economia de energia e desempenho do ambiente. Assim, o primeiro objetivo específico desta dissertação é alcançado.

Tanto o algoritmo de consolidação como o da distribuição fazem uso das heurísticas de alocação (BF, BFM, WF, and WFM) para definir o hospedeiro de origem e destino e das políticas de seleção das VMs (MVC e MTM) para selecionar as VMs mais aptas a participar do processo migratório de modo a minimizar o consumo de energia ou distribuir a carga do sistema. Por consequência, o segundo objetivo específico foi atingido.

Além disso, a solução busca alternativas para reduzir o impacto negativo promovido pelas migrações de máquinas virtuais no desempenho do sistema, quando define uma política de alocação de VMs para reduzir o número de migrações e quando define uma política de seleção de VMs que minimiza o tempo total de migração. Portanto, os objetivos específicos previstos nesta dissertação foram totalmente atingidos.

A proposta foi extensivamente validada e avaliada através de medições em um ambiente real composto por servidores virtualizados heterogêneos. Os experimentos mostram que os algoritmos responsáveis pela consolidação e distribuição das máquinas virtuais entre servidores físicos são capazes de reduzir o consumo de energia e dissipar os pontos de ócio e sobrecarga do sistema.

5.2 Limitações e Trabalhos Futuros

No decorrer desta dissertação, foram identificadas algumas questões que ainda necessitam ser aprofundadas e solucionadas. A seguir são listadas algumas limitações do trabalho atual e trabalhos futuros decorrentes desta pesquisa:

- Nossa estratégia não considera a utilização dos recursos de rede no processo de classificação dos servidores físicos;
- Este trabalho não trata a perda de desempenho das VMs ocasionada pelas migrações entre servidores físicos com características de *hardware* distintas. Potanto, em perspectiva nossa estratégia pode atuar em conjunto com a solução proposta em (REGO *et al.*, 2011) a fim de garantir que a capacidade de processamento da VM seja mantida independente da máquina física em que foi alocada.

Outras sugestões em perspectiva:

- Implementação de uma política de limiar dinâmico, capaz de ajustar-se conforme as mudanças na carga de trabalho em execução no ambiente de experimentação;
- Dependabilidade. Distribuir a estratégia entre os nós do ambiente de experimentação, de modo a evitar um ponto único de falha no sistema;
- Utilizar aplicações com diferentes características (orientadas a CPU, rede, memória, I/O de disco) para avaliar o impacto da estratégia proposta no desempenho das VMs, considerando novas métricas como, por exemplo, tempo de resposta e vazão;
- Uma vez que a solução proposta foi implementada e testada em um ambiente computacional com poucos servidores físicos, desenvolvê-la em simulação visando realizar validação cruzada. Em seguida, extrapolar os cenários de simulação, variando o número de hospedeiros e VMs, com o intuito de testar a escalabilidade da proposta. Desse modo, podemos comparar o desempenho das heurísticas e políticas utilizadas e definir o escopo de utilização da estratégia. Para tanto, esperamos contar com o CloudReports (SÁ; SOARES; GOMES, 2011), uma ferramenta gráfica para simulação de ambientes computacionais em Nuvem baseada em extensões do arcabouço CloudSim (CALHEIROS *et al.*, 2011), que recentemente disponibilizou uma versão com novas características, tais como: modelagem aprimorada de eficiência energética em Computação em Nuvem e um novo pacote que permite a modelagem da topologia interna do *data center* e aplicações de transmissão de mensagens (RELEASE..., 2012).

Apêndice A

Pseudocódigos

Como parte da metodologia descrita no Capítulo 1, após definirmos a arquitetura da solução, concebemos os pseudocódigos dos algoritmos que formam a estratégia proposta nesta dissertação. Estes pseudocódigos serviram de insumo para o desenvolvimento da mesma no ambiente real e são apresentados a seguir:

Algoritmo A.1 Loop Principal

Input: listaHosts, heuristicaLimiar, heuristicaAlocacao, heuristicaSelecaoVM, intervaloMonitoramento

```
1:  // 1. Monitora servidores físicos
2:  para todo host ∈ listaHosts fazer
3:     iniciaColetaDados(host)
4:  fim para
5:  laço
6:     // 2. Classificaçao de servidores físicos
7:     hostsClassificados ← classificaHost(heuristicaLimiar, limiarSuperior,
       limiarInferior, intervaloMonitoramento, listaLogRecursosUtilizadosHosts)
8:     se heuristicaLimiar =′ limiarUnico′ então
9:        executaMitigacao(hostsClassificados, heuristicaLimiar, heuristicaAlocacao,
          heuristicaSelecaoVM)
10:    senão se heuristicaLimiar =′ limiarDuplo′ então
11:       se possuiHostSobrecarregado(hostsClassificados) or
          possuiHostOcioso(hostsClassificados) então
12:          executaMitigacao(hostsClassificados, heuristicaLimiar, heuristicaAlocacao,
             heuristicaSelecaoVM)
13:       fim se
14:    fim se
15: fim laço
```

Algoritmo A.2 classificaHost - Classificação dos Hosts

Input: heuristicaLimiar, limiarSuperior,
limiarInferior // Caso heuristicaLimiar seja limiarUnico,
intervaloMonitoramento, listaLogRecursosUtilizadosHosts
Output: hostsClassificados // Lista de hosts com a respectiva classificação

```
1: para todo logRecursosUtilizadosHosts ∈ listaLogRecursosUtilizadosHosts fazer
2:   // 1. Captura amostras do log
3:   listaAmostra ← capturaAmostras(intervaloMonitoramento)
4:   // 2. Classifica amostras
5:   para todo amostra ∈ listaAmostras fazer
6:     se heuristicaLimiar =' limiarUnico' então
7:       se percentualMemUtilizada ≥ limiarSuperior or
         percentualCPUUtilizada ≥ limiarSuperior então
8:         contadorSobrecarregado ← contadorSobrecarregado + 1
9:       senão
10:        contadorNormal ← contadorNormal + 1
11:      fim se
12:    senão se heuristicaLimiar =' limiarDuplo' então
13:      se percentualMemUtilizada ≥ limiarSuperior or
         percentualCPUUtilizada ≥ limiarSuperior então
14:        contadorSobrecarregado ← contadorSobrecarregado + 1
15:      senão se percentualMemUtilizada ≤ limiarSuperior and
         percentualCPUUtilizada ≤ limiarSuperior então
16:        contadorOcioso ← contadorOcioso + 1
17:      senão
18:        contadorNormal ← contadorNormal + 1
19:      fim se
20:    fim se
21:  fim para
22:  // 3. Classificação hosts
23:  limiar ← intervaloMonitoramento ÷ 2
24:  se contadorSobrecarregado ≥ limiar então
25:    classificacao ←' Sobrecarregado'
26:  senão se contadorOcioso ≥ limiar então
27:    classificacao ←' Ocioso'
28:  senão se contadorNormal ≥ limiar então
29:    classificacao ←' Normal'
30:  senão
31:    classificacao ← classificacaoUltimaAmostra
32:  fim se
33: fim para
```

Algoritmo A.3 Mitigaçao

Input: hostsClassificados, heuristicaLimiar, heuristicaAlocaçao, heuristicaSelecaoVM

1: **se** $possuiHostSobrecarregado(hostsClassificados)$ **então**
2: $distribuiVMs(hostsClassificados, heuristicaAlocacao, heuristicaSelecaoVM)$
3: **senão**
4: $consolidaVMs(hostsClassificados, heuristicaAlocacao, heuristicaLimiar)$
5: **fim se**

Algoritmo A.4 orndenaHostsConsolidacaoHeuristica - Ordena os hosts segundo a heurística desejada

Input: hostsClassificados, heuristicaAlocacao

1: **se** $heuristicaAlocacao =' BFM'$ **então**
2: $ordenaHostsConsolidacaoBFM(hostsClassificados)$
3: **senão se** $heuristicaAlocacao =' WFM'$ **então**
4: $ordenaHostsConsolidacaoWFM(hostsClassificados)$
5: **fim se**

Algoritmo A.5 orndenaHostsConsolidacaoBF

Input: hostsClassificados
Output: hostsClassificadosOrdenados

1: $i \leftarrow 1$
2: **para** $i \rightarrow tamanho(hostsClassificados)$ **fazer**
3: **para** $j = i + 1 \rightarrow tamanho(hostsClassificados)$ **fazer**
4: $vol_i \leftarrow \frac{1}{1-percentualCPUUtilizada(i)} \times \frac{1}{1-percentualMemUtilizada(i)}$
5: $vol_j \leftarrow \frac{1}{1-percentualCPUUtilizada(j)} \times \frac{1}{1-percentualMemUtilizada(j)}$
6: **se** $vol_i > vol_j$ **então**
7: $t \leftarrow hostsClassificados(i)$
8: $hostsClassificados(i) \leftarrow hostsClassificados(j)$
9: $hostsClassificados(j) \leftarrow t$
10: **fim se**
11: **fim para**
12: $i \leftarrow i + 1$
13: **fim para**

Algoritmo A.6 orndenaHostsConsolidacaoBFM

Input: hostsClassificados
Output: hostsClassificadosOrdenados

1: $i \leftarrow 1$
2: **para** $i \rightarrow tamanho(hostsClassificados)$ **fazer**
3: **para** $j = i + 1 \rightarrow tamanho(hostsClassificados)$ **fazer**
4: $vol_i \leftarrow \frac{1}{1-percentualCPUUtilizada(i)} \times \frac{1}{1-percentualMemUtilizada(i)}$
5: $razao_i \leftarrow \frac{vol_i}{potenciaHostFull(i)}$
6: $vol_j \leftarrow \frac{1}{1-percentualCPUUtilizada(j)} \times \frac{1}{1-percentualMemUtilizada(j)}$
7: $razao_j \leftarrow \frac{vol_j}{potenciaHostFull(j)}$
8: **se** $razao_i > razao_j$ **então**
9: $t \leftarrow hostsClassificados(i)$
10: $hostsClassificados(i) \leftarrow hostsClassificados(j)$
11: $hostsClassificados(j) \leftarrow t$
12: **fim se**
13: **fim para**
14: $i \leftarrow i + 1$
15: **fim para**

Algoritmo A.7 orndenaHostsConsolidacaoWFM

Input: hostsClassificados
Output: hostsClassificadosOrdenados

1: $i \leftarrow 1$
2: **para** $i \rightarrow tamanho(hostsClassificados)$ **fazer**
3: **para** $j \leftarrow i + 1 \rightarrow tamanho(hostsClassificados)$ **fazer**
4: $vol_i \leftarrow \frac{1}{1-percentualCPUUtilizada(i)} \times \frac{1}{1-percentualMemUtilizada(i)}$
5: $volFree_i \leftarrow volTotal(i) - vol_i$
6: $razao_i \leftarrow \frac{volFree_i}{potenciaHostFull(i)}$
7: $vol_j \leftarrow \frac{1}{1-percentualCPUUtilizada(j)} \times \frac{1}{1-percentualMemUtilizada(j)}$
8: $volFree_j \leftarrow volTotal(j) - vol_j$
9: $razao_j \leftarrow \frac{volFree_j}{potenciaHostFull(j)}$
10: **se** $razao_i > razao_j$ **então**
11: $t \leftarrow hostsClassificados(i)$
12: $hostsClassificados(i) \leftarrow hostsClassificados(j)$
13: $hostsClassificados(j) \leftarrow t$
14: **fim se**
15: **fim para**
16: $i \leftarrow i + 1$
17: **fim para**

Algoritmo A.8 orndenaHostsConsolidacaoWF

Input: hostsClassificados
Output: hostsClassificadosOrdenados

1: $i \leftarrow 1$
2: **para** $i \rightarrow tamanho(hostsClassificados)$ **fazer**
3: **para** $j \leftarrow i+1 \rightarrow tamanho(hostsClassificados)$ **fazer**
4: $vol_i \leftarrow \frac{1}{1-percentualCPUUtilizada(i)} \times \frac{1}{1-percentualMemUtilizada(i)}$
5: $volFree_i \leftarrow volTotal(i) - vol_i$
6: $vol_j \leftarrow \frac{1}{1-percentualCPUUtilizada(j)} \times \frac{1}{1-percentualMemUtilizada(j)}$
7: $volFree_j \leftarrow volTotal(j) - vol_j$
8: **se** $volFree_i > volFree_j$ **então**
9: $t \leftarrow hostsClassificados(i)$
10: $hostsClassificados(i) \leftarrow hostsClassificados(j)$
11: $hostsClassificados(j) \leftarrow t$
12: **fim se**
13: **fim para**
14: $i \leftarrow i+1$
15: **fim para**

Algoritmo A.9 buscaNovoMapeamentoConsolidacao

Input: hostOrigem,listaVMs,listaHostDestino,limiarSuperior
Output: novoMapeamento

1: $percMemFreeHostOrigem \leftarrow 0$
2: $percCPUFreeHostOrigem \leftarrow 0$
3: $numVMs \leftarrow tamanho(listaVMs)$
4: **para todo** $VM \in listaVMs$ **fazer**
5: $memRequeridaVM \leftarrow capturaMemRequerida(VM)$
6: $percCPUUtilizadaVM \leftarrow capturaPercCPUUtilizada(VM)$
7: **para todo** $hostDestino \in listaHostDestino$ **fazer**
8: $percCPURequeridaVM \leftarrow \frac{percCPUUtilizadaVM \times capturaCPUFreq(hostOrigem)}{capturaCPUFreq(hostDestino)}$
9: // Verifica se o servidor de destino pode hospedar a VM sem gerar uma sobrecarga
10: **se** $capturaMemFreeHost(hostDestino) > memRequeridaVM$ **and** $capturaPercCPUFreeHost(hostDestino) > percCPURequeridaVM$ **então**
11: $armazenaMapeamento(hostOrigemID, VMID, hostDestinoID)$
12: $numVMs \leftarrow numVMs - 1$
13: // Atualiza dados de utilização dos recursos do hospedeiro de origem
14: $memFreeHostOrigem \leftarrow capturaMemFreeHost(hostOrigem) + memRequeridaVM$
15: $percMemFreeHostOrigem \leftarrow \frac{memFreeHostOrigem}{capturaMemTotalHost(hostOrigem)}$
16: $percCPUFreeHostOrigem \leftarrow capturaPercCPUFreeHost(hostOrigem) + percCPUUtilizadaVM$
17: // Atualiza dados de utilização dos recursos do hospedeiro de destino
18: $memFreeHostDestino \leftarrow capturaMemFreeHost(hostDestino) - memRequeridaVM$
19: $percCPUFreeHostDestino \leftarrow capturaPercCPUFreeHost(hostDestino) - percCPURequeridaVM$
20: **se** $numVMs = 0$ **então**
21: **return** $novoMapeamento$
22: **fim se**
23: **goto** *linha* 3
24: **fim se**
25: **fim para**
26: **fim para**

Algoritmo A.10 consolidaVMs - Consolidação de VMs (Limiar único e duplo)

Input: hostsClassificados, heuristicaAlocacao, heuristicaLimiar
Output: novoMapeamento

```
1: desligaHostsSemVM(hostClassificados)
2: // Dispõe os hosts conforme heurística de alocação selecionada
3: ordenaHostsConsolidacaoHeuristica(hostsClassificados, heuristicaAlocacao)
4: se heuristicaLimiar =' limiarDuplo' então
5:    se possuiHostsNormais(hostsClassificados) então
6:       hostsNormais → listaHostsDestino
7:    fim se
8:    hostsOciosos → listaHostsOrigem
9:    hostsOciosos → listaHostsDestino
10: senão se heuristicaLimiar =' limiarUnico' então
11:    hostsNormais → listaHostsDestino
12:    hostsNormais → listaHostsOrigem
13: fim se
14: para todo host ∈ listaHostsOrigem fazer
15:    capturaVMsHost(host)
16:    para todo VM ∈ listaVMsHost fazer
17:       buscaNovoMapeamentoConsolidacao(host, VM, listaHostDestino, limiarSuperior)
18:    fim para
19:    se TodasasVMsdohostforamrealocadas então
20:       para todo linha ∈ novoMapeamento fazer
21:          migracaoVM(host, VM, hostDestino)
22:       fim para
23:       verificaMaster(host)
24:       se host =' Master' então
25:          designaNovoMaster(host, listaHostDestino)
26:       fim se
27:       desligaHost(host)
28:    fim se
29: fim para
```

Algoritmo A.11 orndenaHostsDistribuicaoHeuristica - Ordena os hosts segundo a heurística desejada

Input: listaHostsDestino, heuristicaAlocacao

```
1: se heuristicaAlocacao =' BFM' então
2:    ordenaHostsDistribuicaoBFM(listaHostsDestino)
3: senão se heuristicaAlocacao =' WFM' então
4:    ordenaHostsDistribuicaoWFM(listaHostsDestino)
5: fim se
```

Algoritmo A.12 orndenaHostsDistribuicaoBF

Input: hostsClassificados
Output: hostsClassificadosOrdenados

1: $i \leftarrow 1$
2: **para** $i \rightarrow tamanho(hostsClassificados)$ **fazer**
3: **para** $j = i + 1 \rightarrow tamanho(hostsClassificados)$ **fazer**
4: $vol_i \leftarrow \frac{1}{1-percentualCPUUtilizada(i)} \times \frac{1}{1-percentualMemUtilizada(i)}$
5: $vol_j \leftarrow \frac{1}{1-percentualCPUUtilizada(j)} \times \frac{1}{1-percentualMemUtilizada(j)}$
6: **se** $vol_i < vol_j$ **então**
7: $t \leftarrow hostsClassificados(i)$
8: $hostsClassificados(i) \leftarrow hostsClassificados(j)$
9: $hostsClassificados(j) \leftarrow t$
10: **fim se**
11: **fim para**
12: $i \leftarrow i + 1$
13: **fim para**

Algoritmo A.13 orndenaHostsDistribuicaoBFM

Input: hostsClassificados
Output: hostsClassificadosOrdenados

1: $i \leftarrow 1$
2: **para** $i \rightarrow tamanho(hostsClassificados)$ **fazer**
3: **para** $j = i + 1 \rightarrow tamanho(hostsClassificados)$ **fazer**
4: $vol_i \leftarrow \frac{1}{1-percentualCPUUtilizada(i)} \times \frac{1}{1-percentualMemUtilizada(i)}$
5: $razao_i \leftarrow \frac{vol_i}{potenciaHostFull(i)}$
6: $vol_j \leftarrow \frac{1}{1-percentualCPUUtilizada(j)} \times \frac{1}{1-percentualMemUtilizada(j)}$
7: $razao_j \leftarrow \frac{vol_j}{potenciaHostFull(j)}$
8: **se** $razao_i < razao_j$ **então**
9: $t \leftarrow hostsClassificados(i)$
10: $hostsClassificados(i) \leftarrow hostsClassificados(j)$
11: $hostsClassificados(j) \leftarrow t$
12: **fim se**
13: **fim para**
14: $i \leftarrow i + 1$
15: **fim para**

Algoritmo A.14 orndenaHostsDistribuicaoWFM

Input: hostsClassificados
Output: hostsClassificadosOrdenados

1: $i \leftarrow 1$
2: **para** $i \rightarrow tamanho(hostsClassificados)$ **fazer**
3: **para** $j \leftarrow i+1 \rightarrow tamanho(hostsClassificados)$ **fazer**
4: $vol_i \leftarrow \frac{1}{1-percentualCPUUtilizada(i)} \times \frac{1}{1-percentualMemUtilizada(i)}$
5: $volFree_i \leftarrow volTotal(i) - vol_i$
6: $razao_i \leftarrow \frac{volFree_i}{potenciaHostFull(i)}$
7: $vol_j \leftarrow \frac{1}{1-percentualCPUUtilizada(j)} \times \frac{1}{1-percentualMemUtilizada(j)}$
8: $volFree_j \leftarrow volTotal(j) - vol_j$
9: $razao_j \leftarrow \frac{volFree_j}{potenciaHostFull(j)}$
10: **se** $razao_i < razao_j$ **então**
11: $t \leftarrow hostsClassificados(i)$
12: $hostsClassificados(i) \leftarrow hostsClassificados(j)$
13: $hostsClassificados(j) \leftarrow t$
14: **fim se**
15: **fim para**
16: $i \leftarrow i+1$
17: **fim para**

Algoritmo A.15 orndenaHostsDistribuicaoWF

Input: hostsClassificados
Output: hostsClassificadosOrdenados

1: $i \leftarrow 1$
2: **para** $i \rightarrow tamanho(hostsClassificados)$ **fazer**
3: **para** $j \leftarrow i+1 \rightarrow tamanho(hostsClassificados)$ **fazer**
4: $vol_i \leftarrow \frac{1}{1-percentualCPUUtilizada(i)} \times \frac{1}{1-percentualMemUtilizada(i)}$
5: $volFree_i \leftarrow volTotal(i) - vol_i$
6: $vol_j \leftarrow \frac{1}{1-percentualCPUUtilizada(j)} \times \frac{1}{1-percentualMemUtilizada(j)}$
7: $volFree_j \leftarrow volTotal(j) - vol_j$
8: **se** $volFree_i < volFree_j$ **então**
9: $t \leftarrow hostsClassificados(i)$
10: $hostsClassificados(i) \leftarrow hostsClassificados(j)$
11: $hostsClassificados(j) \leftarrow t$
12: **fim se**
13: **fim para**
14: $i \leftarrow i+1$
15: **fim para**

Algoritmo A.16 ordenaVMsHeuristica - Ordena as VMs segundo a heurística desejada

Input: listaVMsHost, heuristicaSelecaoVM

1: **se** $heuristicaSelecaoVM =' MVC'$ **então**
2: $ordenaVMsMVC(listaVMsHost)$
3: **senão se** $heuristicaSelecaoVM =' MMR'$ **então**
4: $ordenaVMsMMR(listaVMsHost)$
5: **fim se**

Algoritmo A.17 distribuiCarga - Distribuição de VMs

Input: hostsClassificados, heuristicaAlocacao, heuristicaSelecaoVM
Output: novoMapeamento

```
1: // Dispõe os hosts em ordem decrescente de seu volume de carga
2: ordenaHosts(hostsClassificados)
3: se existeHostsNaoSobrecarregado(hostsClassificados) então
4:    para todo host ∈ hostsNaoSobrecarregados(hostsClassificados) fazer
5:       insereHostListaDestino(host)
6:    fim para
7:    // Dispõe os candidatos a hospedar as VMs conforme heurística de alocação
      selecionada
8:    ordenaHostsDistribuicaoHeuristica(listaHostsDestino, heuristicaAlocacao)
9:    para todo host ∈ hostsSobrecarregados fazer
10:      capturaVMsHostSobrecarregado(host)
11:      ordenaVMsHeuristica(listaVMsHost, heuristicaSelecaoVM)
12:      buscaNovoMapeamentoDistribuicao(host, listaVMsHost, listaHostDestino, limiarSuper
13:      se novoMapeamento ≠ ∅ então
14:         para todo linha ∈ novoMapeamento fazer
15:            migracaoVM(host, vm, hostDestino)
16:         fim para
17:      senão
18:         // Uma vez que não foi possível dissipar a sobrecarga, o algoritmo
            consulta a lista de hosts desligados a fim encontrar um host capaz de
            hospedar a(s) VM(s) do host sobrecarregado
19:         buscaNovoMapeamentoDistribuicao(host, listaVMsHost, listaHostDesligados, limiarS
20:      fim se
21:   fim para
22: senão
23:   para todo host ∈ hostsSobrecarregados fazer
24:      capturaVMsHostSobrecarregado(host)
25:      ordenaVMsHeuristica(listaVMsHost, heuristicaSelecaoVM)
26:      // Uma vez que todos os hosts ligados estão sobrecarregados, o algoritmo
         consulta a lista de hosts desligados a fim encontrar um host capaz de
         hospedar VM(s) visando dissipar as sobrecargas
27:      buscaNovoMapeamentoDistribuicao(host, listaVMsHost, listaHostDesligados, limiarSu
28:   fim para
29: fim se
```

Algoritmo A.18 buscaNovoMapeamentoDistribuicao

Input: hostOrigem,listaVMs,listaHostDestino,limiarSuperior
Output: novoMapeamento

1: $percMemFreeHostOrigem \leftarrow 0$
2: $percCPUFreeHostOrigem \leftarrow 0$
3: **para todo** $VM \in listaVMs$ **fazer**
4: $memRequeridaVM \leftarrow capturaMemRequerida(VM)$
5: $percCPUUtilizadaVM \leftarrow capturaPercCPUUtilizada(VM)$
6: **para todo** $hostDestino \in listaHostDestino$ **fazer**
7: $percCPURequeridaVM \leftarrow \frac{percCPUUtilizadaVM \times capturaCPUFreq(hostOrigem)}{capturaCPUFreq(hostDestino)}$
8: **se** $capturaMemFreeHost(hostDestino) > memRequeridaVM$ **and** $capturaPercCPUFreeHost(hostDestino) > percCPURequeridaVM$ **então**
9: $armazenaMapeamento(hostOrigemID, VMID, hostDestinoID)$
10: // Atualiza dados de utilização dos recursos do hospedeiro de origem
11: $memFreeHostOrigem \leftarrow capturaMemFreeHost(hostOrigem) + memRequeridaVM$
12: $percMemFreeHostOrigem \leftarrow \frac{memFreeHostOrigem}{capturaMemTotalHost(hostOrigem)}$
13: $percCPUFreeHostOrigem \leftarrow capturaPercCPUFreeHost(hostOrigem) + percCPUUtilizadaVM$
14: // Atualiza dados de utilização dos recursos do hospedeiro de destino
15: $memFreeHostDestino \leftarrow capturaMemFreeHost(hostDestino) - memRequeridaVM$
16: $percCPUFreeHostDestino \leftarrow capturaPercCPUFreeHost(hostDestino) - percCPURequeridaVM$
17: **se** $percMemFreeHostOrigem > (1 - limiarSuperior)$ **or** $percCPUFreeHostOrigem > (1 - limiarSuperior)$ **então**
18: **return** $novoMapeamento$
19: **fim se**
20: **fim se**
21: **break**
22: **fim para**
23: **fim para**

Referências Bibliográficas

ALENCAR, J. M. U. de; ANDRADE, R.; VIANA, W.; SCHULZE, B. P2pscheme: a p2p scheduling mechanism for workflows in grid computing. *Concurrency and Computation: Practice and Experience*, John Wiley Sons, Ltd, p. n/a–n/a, 2011.

BARHAM, P.; DRAGOVIC, B.; FRASER, K.; HAND, S.; HARRIS, T.; HO, A.; NEUGEBAUER, R.; PRATT, I.; WARFIELD, A. Xen and the art of virtualization. *ACM SIGOPS Operating Systems Review*, ACM, v. 37, n. 5, p. 164–177, 2003.

BELOGLAZOV, A.; ABAWAJY, J.; BUYYA, R. Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing. *Future Generation Computer Systems*, Elsevier, 2011.

BELOGLAZOV, A.; BUYYA, R. Energy efficient resource management in virtualized cloud data centers. In: IEEE COMPUTER SOCIETY. *Proceedings of the 2010 10th IEEE ACM International Conference on Cluster, Cloud and Grid Computing*. 2010. p. 826–831.

BELOGLAZOV, A.; BUYYA, R. Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in cloud data centers. *Concurrency and Computation: Practice and Experience*, Wiley Online Library, 2011.

BELOGLAZOV, A.; BUYYA, R.; LEE, Y.; ZOMAYA, A. A taxonomy and survey of energy-efficient data centers and cloud computing systems. *Advances in Computers*, Academic Press, v. 82, p. 47–111, 2011.

BERL, A.; RACE, N.; ISHMAEL, J.; MEER, H. de. Network virtualization in energy-efficient office environments. *Computer Networks*, Elsevier, p. 2856–2868, 2010.

BUYYA, R.; YEO, C.; VENUGOPAL, S.; BROBERG, J.; BRANDIC, I. Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation computer systems*, Elsevier, v. 25, n. 6, p. 599–616, 2009.

CALHEIROS, R.; RANJAN, R.; BELOGLAZOV, A.; ROSE, C. D.; BUYYA, R. Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Software: Practice and Experience*, Wiley Online Library, v. 41, n. 1, p. 23–50, 2011.

CHU, T.; LIN, Y. A fuzzy topsis method for robot selection. *The International Journal of Advanced Manufacturing Technology*, Springer, v. 21, n. 4, p. 284–290, 2003.

CLARK, C.; FRASER, K.; HAND, S.; HANSEN, J.; JUL, E.; LIMPACH, C.; PRATT, I.; WARFIELD, A. Live migration of virtual machines. In: USENIX ASSOCIATION. *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation*. 2005. v. 2, p. 273–286.

CULLY, B.; LEFEBVRE, G.; MEYER, D.; FEELEY, M.; HUTCHINSON, N.; WARFIELD, A. Remus: High availability via asynchronous virtual machine replication. In: USENIX ASSOCIATION. *Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation*. 2008. p. 161–174.

DAS, T.; PADALA, P.; PADMANABHAN, V.; RAMJEE, R.; SHIN, K. Litegreen: Saving energy in networked desktops using virtualization. In: USENIX ASSOCIATION. *Proceedings of the 2010 USENIX conference on USENIX annual technical conference*. 2010. p. 3–3.

DASGUPTA, S.; PAPADIMITRIOU, C.; VAZIRANI, U. *Algorithms*. : McGraw-Hill, 2006.

DUNLAP, G.; KING, S.; CINAR, S.; BASRAI, M.; CHEN, P. Revirt: Enabling intrusion analysis through virtual-machine logging and replay. *ACM SIGOPS Operating Systems Review*, ACM, v. 36, p. 211–224, 2002.

ELNOZAHY, E.; KISTLER, M.; RAJAMONY, R. Energy-efficient server clusters. *Power-Aware Computer Systems*, Springer, p. 179–197, 2003.

HINES, M.; GOPALAN, K. Post-copy based live virtual machine migration using adaptive pre-paging and dynamic self-ballooning. In: ACM. *Proceedings of the 2009 ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*. 2009. p. 51–60.

HIROFUCHI, T.; NAKADA, H.; ITOH, S.; SEKIGUCHI, S. Enabling instantaneous relocation of virtual machines with a lightweight vmm extension. In: IEEE. *Cluster, Cloud and Grid Computing (CCGrid), 2010 10th IEEE/ACM International Conference on*. 2010. p. 73–83.

KVM. 2011. Http://www.linux-kvm.org/.

LIAO, X.; HU, L.; JIN, H. Energy optimization schemes in cluster with virtual machines. *Cluster Computing*, Springer, v. 13, n. 2, p. 113–126, 2010.

LIU, H.; JIN, H.; LIAO, X.; HU, L.; YU, C. Live migration of virtual machine based on full system trace and replay. In: ACM. *Proceedings of the 18th ACM international Symposium on High Performance Distributed Computing*. 2009. p. 101–110.

MAGALHÃES, D.; SOARES, J.; GOMES, D. Análise do impacto de migração de máquinas virtuais em ambiente computacional virtualizado. In: FACOM-UFMS. *Proceedings of XXIX Brazilian Symposium on Computer Networks and Distributed Systems (SBRC 2011)*. 2011. p. 235–248.

MELL, P.; GRANCE, T. The nist definition of cloud computing. *NIST special publication*, v. 800, p. 145, 2011.

OLIVEIRA, C.; PETRUCCI, V.; LOQUES, O. Impact of server dynamic allocation on the response time for energy-efficient virtualized web clusters. In: *XXVIII Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos-12th Brazillian Workshop on Real-Time and Embedded Systems (WTR)*. 2010.

OPEN Xen Manager. 2012. Http://wiki.xensource.com/xenwiki/XCP_Projects.

OSMAN, S.; SUBHRAVETI, D.; SU, G.; NIEH, J. The design and implementation of zap: A system for migrating computing environments. *ACM SIGOPS Operating Systems Review*, ACM, v. 36, n. SI, p. 361–376, 2002.

PALLIPADI, V.; STARIKOVSKIY, A. The ondemand governor. In: *Proceedings of the Linux Symposium*. 2006. v. 2, p. 215–230.

PETRUCCI, V.; LOQUES, O.; MOSSÉ, D. A dynamic optimization model for power and performance management of virtualized clusters. In: ACM. *Proceedings of the 1st International Conference on Energy-Efficient Computing and Networking (e-Energy '10)*. 2010. p. 225–233.

REGO, P.; COUTINHO, E.; GOMES, D.; SOUZA, J. de. Faircpu: Architecture for allocation of virtual machines using processing features. In: IEEE. *Utility and Cloud Computing (UCC), 2011 Fourth IEEE International Conference on*. 2011. p. 371–376.

RELEASE CloudSim Toolkit 3.0. 2012. Http://code.google.com/p/cloudsim/wiki/ReleaseNotes.

SCHMIDT, B. *Supporting ubiquitous computing with stateless consoles and computation caches*. Tese (Doutorado) — Stanford University, 2000.

SEMERARO, G.; MAGKLIS, G.; BALASUBRAMONIAN, R.; ALBONESI, D.; DWARKADAS, S.; SCOTT, M. Energy-efficient processor design using multiple clock domains with dynamic voltage and frequency scaling. In: IEEE. *Proceedings of the 8th International Symposium on High-Performance Computer Architecture*. 2002. p. 29–41.

SPECTOR, S. New to xen guide. In: XEN.ORG COMMUNITY. 2010. p. 1–10.

SÁ, T.; SOARES, J. M.; GOMES, D. Cloudreports: uma ferramenta gráfica para a simulação de ambientes computacionais em nuvem baseada no framework cloudsim. *IX Workshop em Clouds, Grids e Aplicações (WCGA 2011)*, SBRC 2011, p. 103–116, 2011.

TALEBI, M.; WAY, T. Methods, metrics and motivation for a green computer science program. In: ACM. *ACM SIGCSE Bulletin*. 2009. v. 41, n. 1, p. 362–366.

TARIGHI, M.; MOTAMEDI, S.; SHARIFIAN, S. A new model for virtual machine migration in virtualized cluster server based on fuzzy decision making. *Journal of Telecommunications, vol. 1, issue 1*, p. 40–51, 2010.

VERMA, A.; AHUJA, P.; NEOGI, A. pmapper: power and migration cost aware application placement in virtualized systems. Springer, p. 243–264, 2008.

VMWARE Virtualization Software. 2011. Http://www.vmware.com/.

VOORSLUYS, W.; BROBERG, J.; VENUGOPAL, S.; BUYYA, R. Cost of virtual machine live migration in clouds: A performance evaluation. Springer, p. 254–265, 2009.

WOOD, T.; SHENOY, P.; VENKATARAMANI, A.; YOUSIF, M. Sandpiper: Black-box and gray-box resource management for virtual machines. *Computer Networks*, Elsevier, v. 53, n. 17, p. 2923–2938, 2009.

XCP. 2011. Http://xen.org/products/cloudxen.html.

XU, M.; MALYUGIN, V.; SHELDON, J.; VENKITACHALAM, G.; WEISSMAN, B. Retrace: Collecting execution trace with virtual machine deterministic replay. In: *Proceedings of the Third Annual Workshop on Modeling, Benchmarking and Simulation*. 2007.