MINISTÉRIO DA DEFESA
EXÉRCITO BRASILEIRO
DEPARTAMENTO DE CIÊNCIA E TECNOLOGIA
INSTITUTO MILITAR DE ENGENHARIA
CURSO DE GRADUAÇÃO EM ENGENHARIA DE COMPUTAÇÃO

PEDRO HENRIQUE SANTOS FERREIRA
YAN CARVALHO MARTINS

BALANCEAMENTO DE CARGA DE MÁQUINAS VIRTUAIS EM UM AMBIENTE DE CLOUD COMPUTING

Rio de Janeiro
2013

INSTITUTO MILITAR DE ENGENHARIA

PEDRO HENRIQUE SANTOS FERREIRA

YAN CARVALHO MARTINS

BALANCEAMENTO DE CARGA DE MÁQUINAS VIRTUAIS EM UM AMBIENTE DE CLOUD COMPUTING

Projeto Final de Curso apresentado ao Curso de Engenharia de Computação do Instituto Militar de Engenharia, como requisito parcial para a obtenção da graduação em Engenharia de Computação.

Orientadora: Raquel Coelho Gomes Pinto - D.Sc.

Rio de Janeiro

2013

INSTITUTO MILITAR DE ENGENHARIA

Praça General Tibúrcio, 80 – Praia Vermelha

Rio de Janeiro - RJ CEP: 22290-270

621.39 Ferreira, Pedro Henrique Santos.

F383b Balanceamento de Carga de Máquinas Virtuais em um Ambiente de Cloud Computing/ Pedro Henrique Santos Ferreira. Yan Carvalho Martins; orientado por Raquel Coelho Gomes Pinto. -Rio de Janeiro: Instituto Militar de Engenharia, 2013.

48: il.

Projeto de Final de Curso. - Instituto Militar de Engenharia. - Rio de Janeiro, 2013.

1. Engenharia de Computação. 2. Máquina Virtual. 3. Migração. 4. Balanceamento de Carga. I. Martins, Yan Carvalho II. Pinto, Raquel Coelho Gomes III. Título IV. Instituto Militar de Engenharia.

CDD 621.39

INSTITUTO MILITAR DE ENGENHARIA

PEDRO HENRIQUE SANTOS FERREIRA
YAN CARVALHO MARTINS

BALANCEAMENTO DE CARGA DE MÁQUINAS VIRTUAIS EM UM AMBIENTE DE CLOUD COMPUTING

Projeto Final de Curso apresentado ao Curso de Engenharia de Computação do Instituto Militar de Engenharia, como requisito parcial para a obtenção da graduação em Engenharia de Computação.

Orientadora: Raquel Coelho Gomes Pinto - D.Sc.

Aprovada em 11 de Junho de 2013 pela seguinte Banca Examinadora:

_____

Raquel Coelho Gomes Pinto - D.Sc., do IME

_____

Anderson Fernandes Pereira dos Santos – D.Sc., do IME

_____

Ricardo Choren Noya - D.Sc., do IME

Rio de Janeiro

2013

SUMÁRIO

LISTA DE ILUSTRAÇÕES

LISTA DE SIGLAS

API	*Application Programming Interface*
CPU	*Central Processing Unit*
E/S	Entrada e Saída
GFS2	*Global File System 2*
HLL	*High Level Language*
IP	*Internet Protocol*
iSCSI	*Internet Small Computer System Interface*
JVM	*Java Virtual Machine*
KVM	*Kernel-based Virtual Machine*
MF	Máquina Física
MMV	Monitor de Máquina Virtual
MV	Máquina Virtual
NFS	*Network File System*
SJF	*Shortest Job First*
SO	Sistema Operacional
UDP	*User Datagram Protocol*
VT	*Virtualization Technology*

RESUMO

Com o aumento considerável do uso de máquinas virtuais em servidores e a alta necessidade de proporcionar confiabilidade de serviços aos usuários, a técnica de balanceamento de carga tem se tornado cada vez mais importante para melhorar o desempenho das aplicações. Este trabalho tem por objetivo desenvolver e implementar um algoritmo de balanceamento de carga de máquinas virtuais. São abordados inicialmente os assuntos referentes à virtualização básica, conceitos de migração de máquinas virtuais e balanceamento de carga. Por fim, são apresentados os experimentos realizados de migração de máquinas virtuais e o algoritmo a ser implementado.

ABSTRACT

With the increase of the use of virtual machines in servers and data centers and the elevated need to provide reliability of services to users, the load balance technique has become ever more important to improve the performance of the applications. This work's main goal is to develop and implement a load balance algorithm of virtual machines. At first, we approach a basic study of virtualization, concepts of virtual machine migrations and load balance. At last, we present the virtual machine migration experiments and the algorithm that will be implemented.

1 INTRODUÇÃO

Os conceitos iniciais de virtualização surgiram há mais de 30 anos (POPEK e GOLDBERG, 1974) e seu principal objetivo foi permitir a execução de *softwares* antigos nos caros *mainframes* da década de 70 (TANENBAUM e STEEN, 2006). O *software* não inclui somente as diversas aplicações, mas também os sistemas operacionais para os quais elas foram desenvolvidas.

Recentemente, a virtualização voltou a ser um tema de grande interesse na área da computação. Assim como os *mainframes* nas décadas de 60 e 70, os *clusters* computacionais atuais são compartilhados por diversos usuários, sendo frequente a utilização da virtualização para suportar as diversas atividades dos usuários. Adicionalmente, o aumento do poder computacional, devido aos avanços tecnológicos, tornou possível o suporte a perda de desempenho causada pela virtualização, permitindo sua utilização em computadores pessoais.

Dentre os diversos benefícios da virtualização, destacam-se:

i. Portabilidade: Possibilita executar um programa binário em uma plataforma diferente daquela para a qual o programa foi compilado.
ii. Isolamento: O funcionamento de uma máquina virtual não interfere nas demais máquinas virtuais hospedadas na mesma máquina física. Qualquer problema que ocorra em uma delas, seja pela execução de um programa malicioso ou um *bug* no sistema operacional, não afetará as demais.
iii. Múltiplos sistemas operacionais: É possível executar diferentes sistemas operacionais em uma mesma máquina.
iv. Migração de máquinas virtuais: Permite migrar máquinas virtuais entre diversas máquinas físicas de forma transparente para a aplicação, ou seja, esta continua executando como se estivesse na mesma máquina. Uma aplicação interessante deste conceito, a qual será analisada em mais detalhes no capítulo 4, é o balanceamento de carga entre máquinas físicas.

Com o aumento do uso de máquinas virtuais, alguns problemas surgiram, como sobrecarga de máquinas e necessidade de reparos. A migração de máquinas virtuais surgiu como uma solução, proporcionando diversos benefícios, tais como aumento de desempenho, facilidade na gerência das aplicações – pois apenas uma máquina é necessária para fornecer diversos serviços – e tolerância a falha. A migração de máquina virtual pode ser utilizada, por exemplo, para resolver a situação de manutenção de um servidor. Caso não existisse a

possibilidade de migrar as máquinas virtuais que rodam nesse servidor, as aplicações iriam parar durante o tempo de manutenção. Porém, basta realizar a migração delas e a máquina física pode ser desligada para efetuar sua manutenção (VOORSLUYS et al, 2009).

Com a dinâmica dos serviços computacionais atuais, características como alto desempenho, boa responsividade e baixo índice de indisponibilidade tornaram-se fatores primordiais para os usuários. Analisando esse fato, um dos problemas frequentemente encontrados é a má distribuição de recursos pelos serviços disponíveis. Por exemplo, duas máquinas que fornecem serviços estão ligadas, uma está com utilização de 90% de CPU (*Central Processing Unit*) e outra apenas 20%. Neste caso, uma das máquinas está sobrecarregada e seu desempenho está reduzido. A técnica de balanceamento de carga torna-se, portanto, importante para trazer melhor desempenho e confiabilidade para o usuário.

1.1 OBJETIVO DO TRABALHO

O objetivo do presente trabalho é desenvolver e implementar um algoritmo de balanceamento de carga de máquinas virtuais. Serão estudadas técnicas de migração de máquinas virtuais e monitoramento de máquinas reais, para auxiliar no balanceamento de carga.

1.2 MOTIVAÇÃO

O uso de máquinas virtuais em *datacenters* tornou-se uma realidade e, através delas, os responsáveis pelos servidores de uma organização podem maximizar a utilização de seus recursos e reduzir a equipe necessária para gerenciar a infraestrutura, possibilitando grande economia no orçamento. Portanto, o uso de técnicas de balanceamento de carga associadas a máquina virtuais tem se tornado primordial no gerenciamento de *datacenters* com o principal objetivo de otimizar dinamicamente a utilização dos recursos de acordo com a demanda.

1.3 METODOLOGIA

A partir do estudo dos conceitos básicos de virtualização, migração de máquinas virtuais e algoritmos de balanceamento de carga, é proposto um algoritmo de balanceamento de máquinas virtuais. A avaliação deste algoritmo é realizada a partir de experimentos, de forma que além do algoritmo de balanceamento de carga, será necessário implementar o monitoramento dos recursos utilizados pelas máquinas físicas e virtuais.

Para o desenvolvimento dos *scripts* de captura de dados das máquinas reais e físicas são utilizadas as bibliotecas *psutil* (PSUTIL, 2013) e *libvirt* (LIBVIRT, 2013) do python. É empregado o *git* como ferramenta de versionamento, para manter o histórico de versões do projeto e facilitar o trabalho paralelo dos autores.

Após a implementação do algoritmo os experimentos finais são realizados utilizando três computadores, conectados na mesma subrede, com suporte à virtualização e à ferramenta de virtualização instalada, além de um outro computador que será um servidor para as imagens do disco e será acessível pelas máquinas do *cluster*. Os resultados são analisados observando a carga de cada máquina física e as migrações disparadas pelo algoritmo sob diversas situações de uso das máquinas.

1.4 ORGANIZAÇÃO

O trabalho segue a seguinte estrutura: no capítulo 2 é realizada uma apresentação dos conceitos básicos de virtualização. No capítulo 3 é explicada a importância e as técnicas existentes de migrações de máquinas virtuais. No capítulo 4 são apresentados conceitos de balanceamento de carga, bem como as principais características dos algoritmos existentes. No capítulo 5 é feita uma descrição da estrutura do algoritmo de balanceamento de carga proposto. No capítulo 6 é apresentada a implementação do algoritmo. No capítulo 7 são mostrados os experimentos realizados e uma análise dos resultados obtidos. No capítulo 8 é realizada uma conclusão do trabalho e são indicados possíveis trabalhos futuros.

2 VIRTUALIZAÇÃO

Os computadores atuais oferecem alto desempenho devido a sua tecnologia avançada. Eles são compostos de diversos *chips*, cada um com centenas de milhares de transistores. Estes, por sua vez, são interconectados e combinados a dispositivos de entrada e saída e infraestrutura de rede, formando plataformas sobre as quais o *software* pode operar. A chave para gerenciar a complexidade dos sistemas computacionais está em dividí-los em níveis de abstração separados por interfaces bem definidas (SMITH e NAIR, 2005).

O uso de níveis de abstração permite ignorar ou simplificar detalhes de implementação de níveis mais baixos, simplificando, portanto, o projeto dos níveis mais altos. Por sua vez, interfaces bem definidas permitem o desacoplamento de diversas tarefas do projeto de computadores, de modo que as equipes dos diversos componentes e subsistemas possam trabalhar de forma mais independente. Apesar de suas vantagens, a utilização de interfaces bem definidas também pode ser restritiva.

Subsistemas e componentes projetados para uma determinada interface não funcionarão em um sistema que oferece uma interface diferente. Essencialmente, a virtualização trata de estender ou substituir uma interface existente de modo a imitar o comportamento de outro sistema, como retratado na figura 2.1.

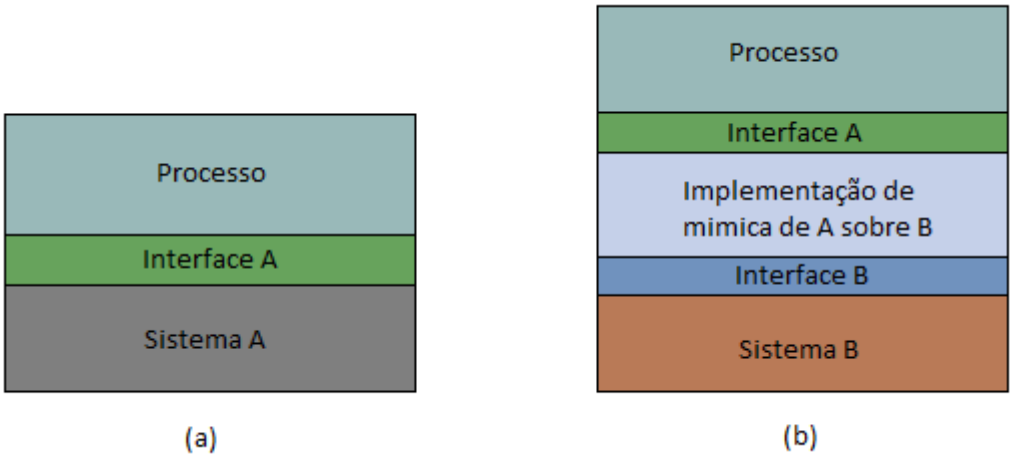

FIG. 2.1 – (a) Sistema não virtualizado (b) Sistema virtualizado

Quando um sistema ou subsistema – processador, memória, dispositivo de E/S (Entrada e Saída), por exemplo – está virtualizado, sua interface e todos os recursos acessados através dela são mapeados na interface e recursos de um sistema real, o qual efetivamente implementa-o. Consequentemente, o sistema real é transformado de modo a oferecer um sistema virtual diferente ou até mesmo um conjunto de sistemas virtuais.

Na figura 2.1-b, o processo, originalmente projetado para utilizar a interface A, pode ser utilizado em uma máquina com interface B, pois a camada de virtualização se localiza entre a interface real e o processo, traduzindo as instruções entre as duas interfaces.

Formalmente, o processo de virtualização envolve a construção de um isomorfismo que mapeia o sistema virtual (hóspede) no sistema real (anfitrião) (POPEK e GOLDBERG, 1974). Este isomorfismo, ilustrado na figura 2.2, mapeia o estado do hóspede para o estado do anfitrião (função V). Para uma sequência de operações e, a qual modifica o estado do hóspede, há uma sequência correspondente de operações e' no anfitrião que realiza uma modificação equivalente em seu estado.

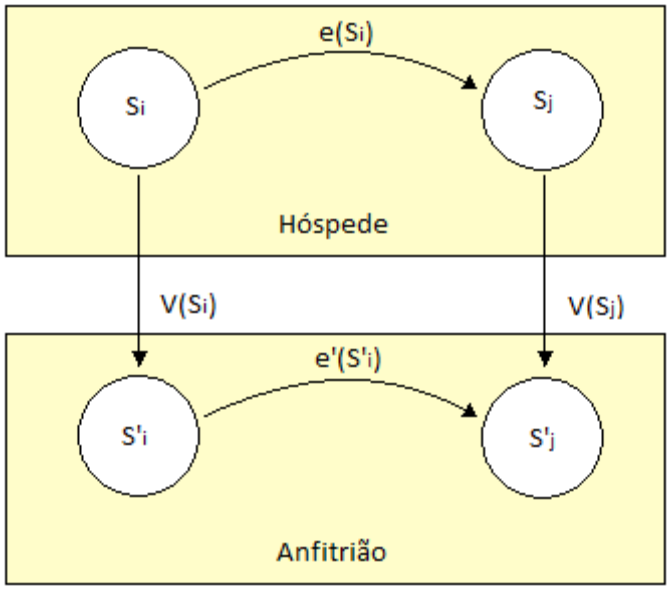

FIG. 2.2 – Formalmente, a virtualização é a construção de um isomorfismo entre um hóspede e um anfitrião; $e' \circ V(S_i) = V \circ e(S_i)$

Para entender o que é uma máquina virtual, é necessário antes entender o conceito de máquina. Na perspectiva de um processo do usuário, a máquina consiste em uma memória de endereçamento lógico que foi designada para ele, além de registradores de usuário e

instruções que permitem a execução do código do processo. Todas as operações de E/S são realizadas através de chamadas ao sistema operacional. Portanto, sob o ponto de vista de um processo, a máquina é uma combinação do sistema operacional e hardware. Já na visão do sistema operacional, a máquina subjacente suporta a execução de todo o sistema. Este, por sua vez, permite a execução de processos de usuários diversos. O sistema operacional fica responsável por alocar memória e dispositivos de E/S para os processos. Assim, a máquina, na perspectiva do sistema operacional, é implementada somente pelo *hardware*.

Do mesmo modo que há uma perspectiva do processo e uma perspectiva do sistema sobre o conceito de máquina, existem máquinas virtuais de processo e de sistema.

2.1 MÁQUINAS VIRTUAIS DE PROCESSO

Neste tipo de máquina virtual, o *software* de virtualização é colocado sobre a combinação sistema operacional/*hardware* e emula tanto as instruções de nível de usuário quanto as chamadas ao sistema operacional. Uma máquina virtual de processo é ilustrada na figura 2.3, na qual as interfaces compatíveis são ilustradas. As fronteiras são representadas com traços distintos para ilustrar as diferentes interfaces.

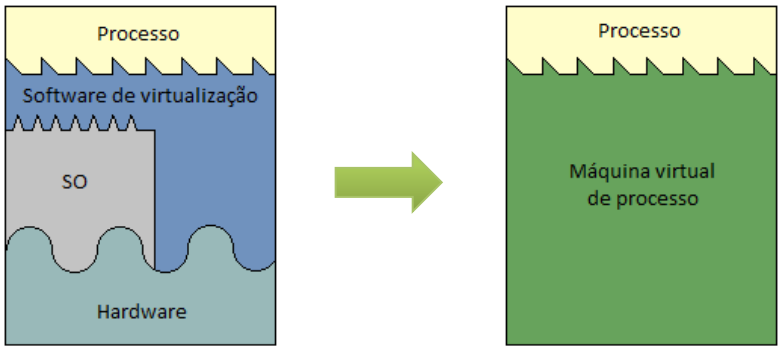

FIG. 2.3 – Em uma máquina virtual de processo, o *software* de virtualização traduz um conjunto de instruções do SO (Sistema operacional) e usuário de uma plataforma para outra

Existem diversas implementações de máquinas virtuais de processo (SMITH e NAIR, 2005). A mais comum é aquela oferecida pela multiprogramação, que está presente em todos os sistemas computacionais atuais. Até por isso, não costuma ser lembrada como um tipo de virtualização. A combinação da interface de chamada ao sistema operacional e do conjunto de instruções de usuário forma a máquina que executa um processo de usuário. A maioria dos

sistemas operacionais suporta a execução simultânea de diversos processos de usuários, onde cada um acredita ter uma máquina inteira para si. Efetivamente, o sistema operacional fornece uma máquina virtual de processo para cada aplicação executada simultaneamente.

Outra implementação, popularizada pela utilização da JVM (*Java Virtual Machine*) (LINDHOLM e YELLIN, 1999), é a máquina virtual HLL (*high-level-language*). Esta virtualização objetiva provê compatibilidade entre diversas plataformas. Assim, a máquina virtual não corresponde diretamente a uma plataforma real; em vez disso, é projetada para facilitar a portabilidade da execução de processos.

2.2 MÁQUINAS VIRTUAIS DE SISTEMA

Neste tipo de virtualização, a principal característica oferecida pelo MMV (Monitor de Máquina Virtual) é a replicação da plataforma. O problema central consiste em dividir um conjunto único de recursos de *hardware* entre os diversos sistemas operacionais hóspedes. O MMV tem acesso e gerencia todos os recursos físicos, controlando o acesso dos hóspedes a eles. Para isso, quando o sistema operacional hóspede realiza certas operações, como instruções privilegiadas que envolvem os recursos físicos compartilhados, elas são interceptadas pelo MMV e checadas para correção. Em seguida, são executadas pelo MMV em nome do hóspede, o qual desconhece todo este trabalho realizado pelo monitor de máquinas virtuais.

Na perspectiva do usuário, a maioria das máquinas virtuais oferecem praticamente as mesmas funcionalidades. Como discutido no início deste capítulo, existem diversas interfaces em um sistema computacional, levando em diferentes escolhas sobre a localização do MMV. As três principais implementações deste tipo de virtualização são descritas a seguir.

2.2.1 HIPERVISOR TIPO 1

A figura 2.4 ilustra esta abordagem de máquinas virtuais de sistema (POPEK e GOLDBERG, 1974), a qual representa a implementação original de MV. O MMV é colocado entre o *hardware* e *software* da máquina e executa no modo mais privilegiado, enquanto as máquinas virtuais, consideradas como processos de nível de usuário, rodam com menos privilégios. A grande desvantagem desta implementação é que o processador precisa ter suporte à tecnologia de virtualização VT (*Virtualization Technology*), a qual permite identificar as instruções sensíveis executadas pelo sistema operacional hóspede. Caso o

processador não tenha este suporte, as instruções sensíveis falham e o sistema não funcionará corretamente.

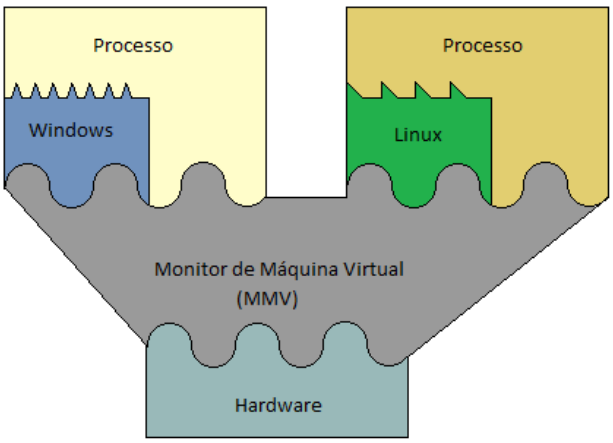

FIG. 2.4 – O *software* de virtualização fica sobre a camada de *hardware*

O Xen é um hipervisor *open source* deste tipo (XEN PROJECT, 2012) e contém os seguintes componentes principais:

- Hipervisor Xen: é o monitor de máquinas virtuais, o qual roda diretamente sobre o *hardware* e é responsável por gerenciar CPU, memória e interrupções. Ele não tem conhecimento de operações de E/S, como rede ou armazenamento.
- Máquinas virtuais hóspedes: são os ambientes virtualizados, cada um rodando seu próprio SO e processos. Cada hóspede é totalmente isolado do *hardware*; logo, também são conhecidos como *unprivileged domains* (DomU).
- Domínio de controle (*Domain* 0): é uma máquina virtual especializada que possui privilégios especiais, como a capacidade de acessar o *hardware* diretamente. Controla todo acesso às operações de E/S do sistema e interage com as outras MV. Também fornece uma interface de controle para o exterior, pela qual todo o sistema é gerenciado.

2.2.2 HIPERVISOR TIPO 2

Neste caso, o *software* de virtualização é construído sobre um sistema operacional anfitrião existente e seu processo de instalação é similar a de um programa de usuário. O *software* de virtualização depende do sistema operacional anfitrião para prover os *drivers* dos dispositivos de E/S. A desvantagem desta abordagem é que pode haver perda de eficiência

devido à adição de mais camadas de *software* entre o *hardware* e o sistema operacional virtualizado.

O primeiro hipervisor deste tipo foi o VMWare (VMWARE, 2012). Inicialmente, ele divide o código a ser executado em blocos básicos, ou seja, sequências de instruções que terminam com instruções *jump, call, trap* ou alguma outra que altere o fluxo de controle. Em seguida, o bloco é inspecionada à procura de instruções sensíveis, cada qual é substituída por uma chamada a uma rotina do VMWare que a gerencia. Também é adicionada uma última instrução de chamada a uma rotina do VMWare. Concluídas estas etapas, o bloco básico é executado diretamente pelo *hardware*. Ao término da execução, o controle retorna ao VMWare, que localiza o bloco sucessor. Se este já estiver traduzido, será executado imediatamente. Caso contrário, precisará primeiro ser traduzido para posterior execução.

Há diversas otimizações para melhorar o desempenho do VMWare. Por exemplo, se um bloco básico for chamado ao término de outro, pode-se adicionar uma instrução de *jump* ou *call* ao fim deste diretamente para o bloco básico desejado. Isto elimina a sobrecarga devido à busca pelo bloco básico sucessor. Adicionalmente, não é necessário substituir todas as instruções sensíveis nos processos de usuários, visto que o *hardware* irá ignorá-las de qualquer forma.

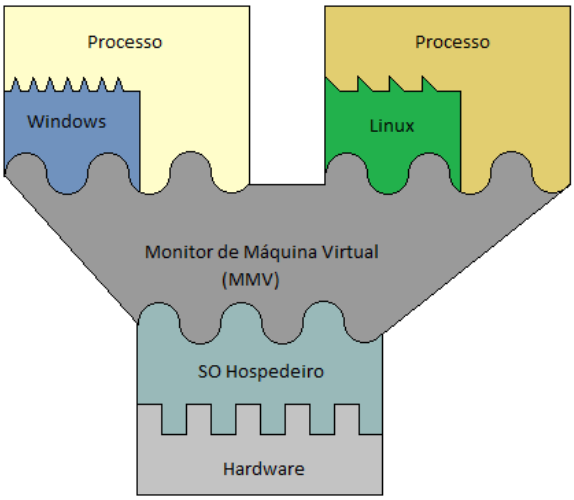

FIG. 2.5 – O *software* de virtualização fica sobre o sistema operacional hospedeiro

2.2.3 PARAVIRTUALIZAÇÃO

Nas duas implementações anteriores, os sistemas operacionais hóspedes não são modificados. Uma abordagem que vem se tornando comum é modificar o código-fonte do sistema operacional, transformando as instruções sensíveis em chamadas de hipervisor. É possível substituir todas as instruções sensíveis do SO por chamadas ao hipervisor para solicitar serviços de sistema como E/S. Um sistema operacional hóspede do qual tenham sido substituídas todas as instruções sensíveis é considerado paravirtualizado (BARHAM et al, 2003). A figura 2.6 apresenta a diferença entre a virtualização clássica e a paravirtualização.

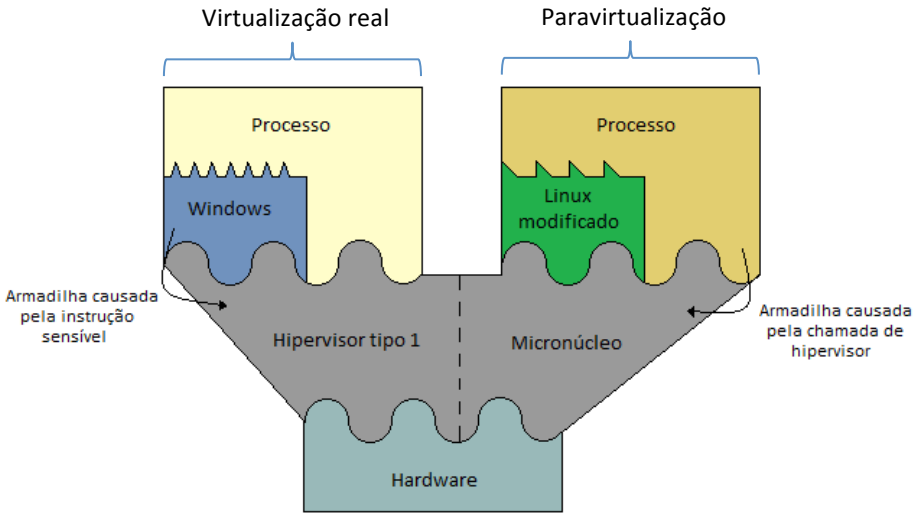

FIG. 2.6 – Um hipervisor tipo 1 controlando tanto uma virtualização real quanto uma paravirtualização

3 MIGRAÇÃO DE MÁQUINA VIRTUAL

A migração de máquinas virtuais é um conceito que possui a finalidade de melhorar o gerenciamento, o desempenho e a tolerância à falha. A ideia é transportar o processamento de uma aplicação de alguma máquina física para outra, bem como todos os processos em execução na máquina virtual e os recursos alocados a eles, modificando apenas onde o serviço se encontra fisicamente, mantendo a mesma lógica. Uma característica importante da migração é que ela seja transparente ao usuário, ou seja, que o cliente que utiliza o serviço não perceba que a aplicação hospedada em uma máquina real está passando para outra. Para atingir esse objetivo, os algoritmos de migração buscam diminuir o tempo total de migração, que é o tempo decorrido desde o momento inicial da migração até o seu término e o *downtime*, que é o tempo em que a máquina virtual está indisponível devido a algum passo da migração.

A escolha do algoritmo depende principalmente do tipo de serviço fornecido pela máquina virtual migrada. Por exemplo, um jogo virtual *online* é uma aplicação em tempo real, na qual os usuários não esperam que o jogo pare por alguns instantes para que a máquina virtual seja migrada. Nesse caso, deve ser utilizado um algoritmo que possua o menor *downtime* possível, mesmo que isso eleve o tempo total de migração. Esse tipo de migração é chamada de migração em tempo real, e um dos seus requerimentos é que os discos das máquinas virtuais estejam compartilhados na rede por meio de algum protocolo, por exemplo *Fibre Channel*, iSCSI (*Internet Small Computer System Interface*), NFS (*Network File System*), GFS2 (*Global File System 2*).

Um dos principais aspectos da migração de máquinas virtuais é a transferência da memória da máquina, que pode ser dividida em até três fases: na fase de cópia das páginas da memória pela rede, enquanto a máquina virtual continua em funcionamento. Para garantir a consistência dos dados, todas as páginas modificadas antes da próxima fase devem ser enviadas novamente. As páginas modificadas podem ser identificadas por dois métodos, no primeiro deles cada página da memória possui um *dirty bit*, o qual indica se a página foi modificada após a última leitura; o segundo deles utiliza o *bit* de escrita da entrada da tabela de páginas. A fase seguinte é de parada da máquina de origem e cópia das páginas restantes para a máquina destino, que é então ligada. A última fase é o momento onde as páginas que

são acessadas e geram uma exceção por falta de página são transferidas para a máquina de destino. (CLARK et al, 2005)

Os algoritmos de migração se baseiam nessas fases utilizando uma ou duas delas, como descrito nas próximas seções.

3.1 PARA E COPIA

Nesse algoritmo o funcionamento da máquina virtual é interrompido e todas as páginas são copiadas para o destino, iniciando uma nova virtualização. Esse é o algoritmo mais simples de ser executado e requer menos processamento. Entretanto, possui o *downtime* mais elevado, o que pode ser inaceitável para um serviço de tempo real ou até mesmo uma máquina virtual que utiliza grande quantidade de memória alocada. (CLARK et al, 2005)

3.2 CÓPIA SOB DEMANDA

Neste caso é utilizado o algoritmo de para e copia apenas para as estruturas básicas do *kernel*, para que a máquina consiga iniciar seu funcionamento no destino e, à medida que as páginas forem sendo solicitadas no seu primeiro uso, as exceções de falta de página serão geradas e essas páginas serão copiadas para a máquina física de destino. A utilização deste algoritmo proporciona um *downtime* muito menor, porém um tempo total de migração mais elevado. A maior desvantagem de utilizá-lo é o baixo desempenho após a migração, pois toda página da memória que necessite ser acessada vai precisar iniciar uma tranferência síncrona para receber o dado da antiga máquina. (CLARK et al, 2005)

3.3 PRÉ-CÓPIA

A pré-cópia possui duas fases principais. Na primeira, é determinado um tempo de intervalo entre cada transferência e inicia-se a iteração. Todas as páginas são copiadas para a máquina de destino na primeira iteração. Após o intervalo de tempo determinado, é realizada uma verificação e são transferidas apenas as páginas que foram modificadas nesse intervalo; na iteração seguinte são copiadas apenas as páginas modificadas no intervalo anterior e assim sucessivamente. Na última iteração, a máquina virtual é interrompida e ocorre a última cópia das páginas modificadas e, pelo fato de não serem muitas, o tempo de *downtime* é reduzido.

Esse algoritmo se baseia na marcação das páginas modificadas por algum método, para que possam ser distinguidas as páginas modificadas, que serão copiadas a cada iteração e

a exclusão dessa marcação a cada rodada, para evitar que ocorram transferências desnecessárias. Além disso, é importante determinar um número ótimo de iterações a ser realizado no total e também o intervalo de tempo entre cada iteração, de forma que o *downtime*, o tempo total de migração e o número de páginas a serem copiadas na última iteração sejam minimizados. (CLARK et al, 2005)

4 BALANCEAMENTO DE CARGA

Na área de computação, balanceamento de carga refere-se à técnica de distribuir uniformemente a carga de trabalho entre dois ou mais recursos físicos. Esta técnica é empregada em várias áreas da computação, desde no nível do sistema operacional, até no nível de aplicações. Por exemplo, o sistema operacional distribui o processamento entre os núcleos de processamento de um multiprocessador, através do uso de uma fila de processos para cada núcleo. Quando uma fila está vazia, o sistema operacional "rouba" processos das filas dos outros núcleos.

No ambiente da *Internet*, o balanceamento de carga é frequentemente utilizado para distribuir o trabalho entre um conjunto de servidores. Muitos serviços (como Google, Facebook, Twitter, entre outros) são implementados em *clusters* e a requisição de cada usuário é encaminhada para um nó do *cluster*. Neste caso, os recursos físicos são os nós do *cluster* e a carga é a requisição do usuário. O balanceamento de carga visa distribuir uniformemente o processamento necessário para atender às requisições, aumentando assim a taxa de atendimento de requisições e, consequentemente, o desempenho do servidor.

Neste trabalho, o foco é o balanceamento de máquinas virtuais entre as diversas máquinas físicas. Neste contexto, os recursos físicos também são os servidores, mas a carga passa a ser o processamento gerado por cada MV. Portanto, o objetivo é distribuir uniformemente as máquinas virtuais entre as máquinas físicas.

Este tipo de balanceamento ganhou ampla importância recentemente, após a difusão de máquinas virtuais em *datacenters* de todo o mundo. Ainda não há, entretanto, muitos artigos publicados sobre o assunto (WILCOX, 2009) e todos são bastante recentes. Assim, não existem algoritmos consagrados para esta área de estudo.

Basicamente, os algoritmos de balanceamento de máquinas virtuais são classificados em relação a três características principais: política, inteligência e arquitetura. A política expressa se o algoritmo possui total acesso às máquinas virtuais ou não; a inteligência mostra se o algoritmo modifica sua forma de tomar decisões ao longo do tempo e a arquitetura diz se o algoritmo é centralizado ou distribuído. A composição de um algoritmo com essas características está exposta na figura 4.1. As subseções seguintes apresentam descrições mais

detalhadas de cada característica. Os algoritmos atendem a uma das classificações de cada subseção.

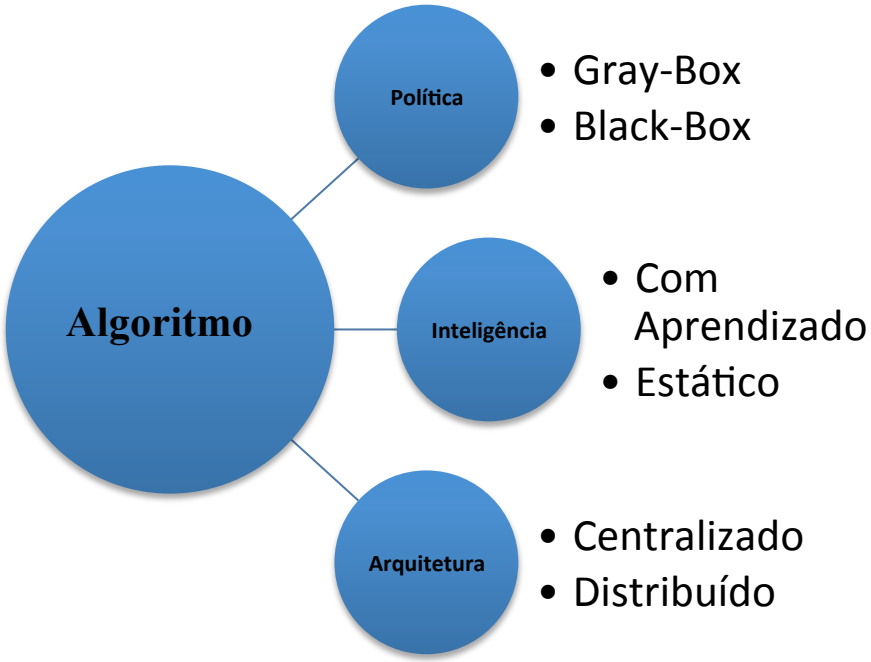

FIG. 4.1 – Classificações dos algoritmos de balanceamento de carga.

4.1 POLÍTICA - *BLACK-BOX* x *GRAY-BOX*

Na técnica de *Black-box* as decisões de balanceamento são feitas sem nenhum conhecimento das aplicações residentes na máquina virtual, mas apenas por observações externas da mesma. Todas as informações devem ser capturadas através do MMV. Este tipo de algoritmo é útil em ambientes nos quais não é possível capturar informações internas de uma MV, como em servidores que rodam aplicações de terceiros. (WOOD et al, 2007)

Já nos algoritmos *Gray-box*, além das informações externas da máquina virtual, o *Gray-box* pode capturar informações internas de uma aplicação, como *logs* e estatísticas a nível de sistema operacional, onde é possível obter com maior facilidade as informações de uso máximo. O *Gray-box* pode ser utilizado, por exemplo, em ambientes de *datacenters* corporativos, onde tanto a infraestrutura de *hardware* quanto as aplicações pertencem à mesma entidade (WOOD et al, 2007).

Em (WOOD et al, 2007), os autores monitoram o uso de CPU, memória e rede para decidir sobre as migrações. Eles implementaram ambos os tipos de algoritmo e os resultados demostraram que a segunda estratégia é superior à primeira, já que as decisões são tomadas com informações mais precisas. Entretanto, como já falado anteriormente, nem sempre é possível obter informações internas da MV, ou seja, só é possível implementar a estratégia *black-box*.

4.2 INTELIGÊNCIA - COM APRENDIZADO x ESTÁTICO

O objetivo principal dos algoritmos com aprendizado é achar o limite ótimo dos recursos de cada máquina que dispara uma migração de MV quando ultrapassado (CHOI et al, 2008). Com isso, pode-se evitar migrações desnecessárias que afetariam o desempenho geral do sistema.

Utilizando esta abordagem em (CHOI et al, 2008), os autores obtém limites de cada recurso próximos dos ideais para diversas situações de uso, bem como otimizam o número de migrações. Neste artigo, o aprendizado é feito analisando o estado do sistema antes e depois de cada migração em termos de desvio padrão da utilização. Os algoritmos estáticos, por sua vez, utilizam sempre os mesmo parâmetros, por exemplo o limite de recursos, para decisão da migração.

4.3 ARQUITETURA - CENTRALIZADO x DISTRIBUÍDO

Nos algoritmos de balanceamento de carga centralizados, todas as máquinas enviam suas informações para uma mesma máquina central, que toma a decisão sobre a distribuição ideal do sistema e controla as migrações necessárias para atingí-la. O artigo (CHOI et al, 2008) apresenta um algoritmo deste tipo.

No caso dos algoritmos distribuídos, todas as máquinas do sistema trocam informações sobre sua carga periodicamente, que são levadas em conta para a tomada de decisão de balanceamento. Cada máquina física é independente para tomar suas decisões de migrar uma MV para outra máquina. Em (ZHAO e HUANG, 2009), foi implementado um algoritmo com esta característica, visando minimizar o desvio padrão das cargas das CPUs de cada máquina física.

A vantagem dos algoritmos centralizados é reduzir o número total de migrações no sistema, já que as decisões são tomadas por uma única máquina que tem informações sobre

todo o sistema. O mesmo não ocorre nos algoritmos descentralizados, já que cada máquina decide sobre as migrações autonomamente a partir de informações parciais do sistema. Por outro lado, os algoritmos centralizados introduzem um ponto único de falha no sistema que também pode se tornar um gargalo; nos algoritmos distribuídos, evidentemente, isto não é um problema.

5 ESTRUTURA DO ALGORITMO PROPOSTO

O algoritmo de balanceamento de carga proposto leva em consideração três recursos: memória, CPU e rede. Ele será implementado de forma estática, utilizando a política *black-box* e centralizado. O algoritmo foi dividido em três fases: detecção de uma máquina física sobrecarregada, decisão de qual máquina virtual será migrada e, por último, a escolha do destino (máquina física) da MV migrada, explicadas a seguir.

5.1 1ª FASE

Nesta etapa de detecção de máquina sobrecarregada, são analisadas as cargas de todas as máquinas físicas em relação a memória, CPU e rede e, caso um dos fatores ultrapasse um limite de 90% do seu total, essa máquina será considerada sobrecarregada.

Para evitar que um pico momentâneo gere uma migração de máquina virtual desnecessariamente, deve ser analisada a carga da máquina física levando em consideração medições passadas. Com isso, a medição é realizada de acordo com a equação abaixo, para cada um dos parâmetros (memória, CPU e rede):

$$N_{i+1} = \mu * N_{instantâneo} + (1 - \mu) * N_i$$

onde $N_{instantâneo}$ é o valor capturado de uso do recurso em porcentagem na iteração i, N_i é o valor de N da iteração anterior e μ é uma constante $0 < \mu < 1$, que expressa o quanto será levado em consideração o valor instantâneo (presente) ou o histórico.

Para o cálculo do N_{i+1} utiliza-se uma média exponencial, da mesma forma como é feito em (SILBERSCHATZ et al, 2005) para o cálculo da previsão da duração do próximo *burst* de CPU no algoritmo de escalonamento SJF (*Shortest Job First*). Portanto, pode-se considerar que o N_i representa o histórico dos valores calculados, $N_{instantâneo}$ é o valor atual e N_{i+1} é uma previsão para o próximo valor instantâneo a ser capturado.

TAB. 5.1 - Comportamento do algoritmo em uma situação de pico

Tempo (t)	$N_{instantâneo}$	N_{i+1}	$N_{instantâneo}$	N_i
0	30	30,00	30	30,00
1	40	38,00	40	36,00
2	50	47,60	50	44,40
3	60	57,52	60	53,76
4	70	67,50	70	63,50
5	75	73,50	75	70,40
6	78	77,10	78	74,96
7	83	81,82	83	79,78
8	96	93,16	96	89,51
9	85	86,63	85	86,81
10	77	78,93	77	80,92
	$\mu = 0,8$		$\mu = 0,6$	

TAB. 5.2 - Comportamento do algoritmo em uma situação de sobrecarga normal

Tempo (t)	$N_{instantâneo}$	N_{i+1}	$N_{instantâneo}$	N_{i+1}
0	30	30,00	30	30,00
1	40	34,00	40	36,00
2	50	40,40	50	44,40
3	60	48,24	60	53,76
4	70	56,94	70	63,50
5	80	66,17	80	73,40
6	85	73,70	85	80,36
7	90	80,22	90	86,14
8	93	85,33	93	90,26
9	95	89,20	95	93,10
10	98	92,72	98	96,04
	$\mu = 0,4$		$\mu = 0,6$	

A tabela 5.1 apresenta a análise do valor de μ para uma situação de pico e a tabela 5.2 apresenta uma situação de sobrecarga normal. Essas duas situações mostram que os valores de $\mu = 0,8$ e $\mu = 0,4$ não seriam apropriados, devendo ser escolhido um valor dentro desse intervalo. Os dados apresentados nas tabelas 5.1 e 5.2 mostram exemplos onde o valor adotado de μ (0,6) se adapta bem às situações de sobrecarga.

Analisando a tabela 5.1, percebe-se que para $\mu = 0,8$, devido ao fato do valor instantâneo ter um peso bastante elevado, uma migração de MV é disparada de forma desnecessária no instante t=8, por causa de um pico momentâneo. Por outro lado, na segunda situação (com o valor de $\mu = 0,6$), o algoritmo consegue evitar a migração desnecessária em

uma ocorrência de pico de uso, pois o maior valor alcançado pelo N_{i+1} é de 89,51 no tempo t=8, que não ultrapassa o limite de 90%.

Avaliando a tabela 5.2, nota-se que para μ = 0,4, por ter sido considerado um peso menor para o valor instantâneo, o algoritmo demora a detectar a sobrecarga, que começa a ocorrer no valor $N_{instantâneo}$ no instante t=7, mas só é detectada pelo algoritmo no momento t=10, quando o N_{i+1} atinge o valor de 92,72. Nesse instante, o recurso já está com 98% de uso, como visto no $N_{instantâneo}$, o que é uma situação extremamente crítica. De outro modo, para o valor de μ = 0,4, observa-se que a sobrecarga é identificada pelo algoritmo no instante seguinte (t=8), onde o N_{i+1} alcança o valor de 90,26. Nesse momento, o recurso está apenas com 93% de uso, o que gera uma situação menos crítica do que no caso anterior.

O valor de μ pode ser interpretado como a agressividade do algoritmo em relação à detecção de sobrecarga. Ou seja, caso o valor seja mais próximo de 1, o algoritmo poderá fazer migrações desnecessárias em uma situação de pico, porém iria detectar mais rápido uma situação de sobrecarga normal. Já se ele for mais próximo de 0, poderá demorar muito tempo para perceber um caso real de sobrecarga, embora se comporte melhor em situações de pico. Desta forma, o valor de μ deve ser ajustado para equilibrar estes dois casos. Neste trabalho, com o valor de μ = 0,6, o algoritmo considera um peso maior para o valor instantâneo, tornando-se mais agressivo em descobrir a sobrecarga. Esse valor pode ser alterado de acordo com as necessidades e preferências do usuário do algoritmo.

Em (WOOD et al, 2007), é realizada a análise de *n* valores anteriores e, caso *k* deles excedam o valor de sobrecarga, além do valor que é calculado como previsão da próxima captura de informações, a máquina é considerada sobrecarregada. Este método busca impedir migrações em caso de pico temporário. No algoritmo implementado neste trabalho, o cálculo de $Nacumulado$ cumpre o mesmo objetivo, embora de forma mais simples.

5.2 2ª FASE

Nesta fase precisa-se escolher qual das máquinas virtuais hospedadas na máquina física sobrecarregada deverá ser migrada para aliviar sua carga. É utilizado o conceito de volume de uma máquina definido em (WOOD et al, 2007), calculado por:

$$Volume = \frac{1}{1 - cpu} * \frac{1}{1 - mem} * \frac{1}{1 - rede}$$

onde *cpu*, *mem* e *rede* são valores entre 0 e 1, que representam a porcentagem de utilização desses recursos na MV. Para o custo de migração é utilizada a função de Cobb-Douglas (HONG, 2008) e o valor é definido por:

$$Custo = Vol^{\alpha} * Img^{\beta}$$

onde *Vol* é o volume da máquina virtual, *Img* é o tamanho da imagem da memória que deverá ser migrada e α, β são constantes onde $0 < \alpha, \beta < 1$ e $\alpha + \beta = 1$, como apresentado em (HONG, 2008). O motivo da última restrição para α e β é que, caso *Vol* e *Img* dobrem, o custo de migração da MV também dobrará.

O custo de migração deve levar em consideração o volume pois, quando ele é alto, significa que esta máquina virtual está sendo muito utilizada. Portanto, deve ser evitada a migração da mesma para que não haja um período de inatividade para seus usuários.

Por outro lado, o tamanho da imagem também é importante, pois toda a imagem da MV precisa ser transferida pela rede durante uma migração, o que é custoso no caso de imagens grandes. (WOOD et al, 2007)

Em (WOOD et al, 2007), as máquinas virtuais são ordenadas de forma decrescente em relação ao valor *Vol*/*Img*, e é escolhida a primeira MV para ser migrada. Esse método propõe maximizar a transferência de volume por quantidade de *bytes*. Essa abordagem não foi adotada, para evitar migrar as máquinas com maiores volumes.

No algoritmo proposto são utilizados os valores de $\alpha = 0.4$ e $\beta = 0.6$. As tabelas 5.3 e 5.4 exemplificam situações da seleção da MV a ser migrada.

TAB. 5.3 - Escolha da MV a ser migrada com 3 MV

CPU	Rede	Memória	Volume	Imagem (MB)	Custo
0.5	0.6	0.1	5.56	120	35.11
0.1	0.08	0.05	1.27	400	40.08
0.35	0.1	0.6	4.27	200	42.95

TAB. 5.4 - Escolha da MV a ser migrada com 4 MV

CPU	Rede	Memória	Volume	Imagem (MB)	Custo
0.32	0.6	0.1	4.08	120	31.04
0.09	0.08	0.05	1.26	400	39.90
0.2	0.1	0.2	1.74	200	29.95
0.27	0.15	0.3	2.30	300	42.77

Na tabela 5.3, o menor custo é da primeira MV, pois, apesar de possuir o maior volume, o tamanho de sua imagem é muito menor em comparação com as outras. No caso da tabela 5.4, o menor custo é o da terceira MV, a qual possui um volume pequeno e uma imagem não muito grande. Dessa forma, essas duas situações exemplificam o comportamento do algoritmo e a MV escolhida para ser migrada foi condizente com o esperado.

O algoritmo irá calcular o custo de migração para cada máquina virtual e ordenar pelo menor custo. Desta forma, será analisada a máquina virtual que possui o custo mínimo e é verificado se, ao migrá-la, a máquina física que a hospeda deixa de estar sobrecarregada. A migração só irá ocorrer se liberar os recursos da máquina física de forma que esta fique com no máximo de 85% de seus recursos utilizados. Esta restrição visa impedir que seja necessária outra migração em um curto espaço de tempo. Caso a MV de menor custo não atenda a esta condição, a mesma análise é realizada com as MV seguintes, em ordem de custo.

5.3 3ª FASE

A fase final consiste em decidir qual será a máquina física destino da máquina virtual selecionada na fase anterior. A escolha mais simples seria migrar para a máquina que está com a maior quantidade de recursos disponíveis, como feito em (WILCOX, 2009). Esta abordagem, entretanto, não é ótima, pois a máquina física com mais recursos poderia suportar outra MV de maior volume em migrações futuras, enquanto as necessidades da MV migrada neste momento poderiam ser suportadas por outra máquina física com menos recursos disponíveis (KHANNA et al, 2006). Por outro lado, também não é ideal migrar para a máquina física com menos recursos disponíveis que atendam a MV, pois pode haver a necessidade de outra migração em um curto espaço de tempo.

Desta forma, o algoritmo proposto irá escolher a máquina física com menos recursos disponíveis, porém com capacidade de abrigar a MV a ser migrada e com uma margem de

segurança para evitar a sobrecarga imediata. Com isso, a máquina física deverá ser capaz de suportar a máquina virtual a ser migrada e ficar com no máximo 85% dos recursos consumidos, para que seja evitada uma sobrecarga imediata.

Caso não seja encontrada uma máquina física com recursos suficientes para suportar a máquina virtual a ser migrada, retorna-se a fase anterior para escolher uma nova MV que necessite de menos recursos.

6 IMPLEMENTAÇÃO

O algoritmo foi implementado utilizando a linguagem de programação *Python*. O monitoramento dos recursos das máquinas físicas é feito através da biblioteca *psutil* do *Python*, que fornece informações, como porcentagem de utilização de CPU, memória e rede. No caso do monitoramento das máquinas virtuais, utiliza-se a biblioteca *Libvirt*, que é uma API de virtualização escrita na linguagem C e possui uma biblioteca para *Python*. Sua principal vantagem é permitir a comunicação com os principais hipervisores existentes. Assim, embora o presente trabalho utilize o hipervisor KVM (*Kernel-based Virtual Machine*), o código poderá ser portado para outros hipervisores com pouca ou nenhuma modificação.

O hipervisor de cada máquina física presente no *cluster* monitorado é responsável por adquirir informações de uso dos seus recursos a cada 5 segundos e avaliar se a MF está sobrecarregada ou não. Ao notar uma sobrecarga, a máquina deve enviar uma mensagem à máquina central com um pedido de migração, passando as informações de uso de seus recursos e de todas as suas máquinas virtuais.

Ao receber uma mensagem de migração de alguma das máquinas físicas, a máquina central envia uma mensagem para cada máquina física do *cluster* requisitando suas informações de uso dos recursos (CPU, memória e rede), para que tenha as informações de uso atualizadas de todas as máquinas. Após receber todas as respostas, a máquina central irá buscar uma combinação de máquinas virtuais a serem migradas de modo a aliviar a máquina física sobrecarregada.

Toda a troca de mensagens entre as máquinas é feito através do protocolo UDP (*User Datagram Protocol*), pois são pacotes menores, ou seja, não sobrecarregam a rede. Além disso, por se tratar de computadores interligados por uma rede local, não há perda significativa de pacotes.

A análise de migração na máquina central é iniciada ordenando as máquinas virtuais pelo menor custo, conforme apresentado na seção 5.2. Estas são avaliadas uma a uma para ver se, primeiramente, a migração de uma máquina virtual irá aliviar a máquina física sobrecarregada e, em seguida, se há alguma máquina física que suporta essa MV. Caso não se obtenha sucesso verificando uma a uma, são analisadas combinações de máquinas virtuais, duas a duas, três a três, e assim sucessivamente, até achar uma combinação que seja possível

de se realizar a migração, ou seja, até que exista um destino para todas as MV's migradas e a máquina que está sobrecarregada seja aliviada após as migrações. Após isso, é enviada uma mensagem para a máquina sobrecarregada informando quais máquinas virtuais devem ser migradas e seus respectivos destinos. Por último, a máquina central recebe uma mensagem informando o fim da migração.

A migração de uma máquina virtual é realizada em tempo real, utilizando-se a biblioteca *Libvirt*. Portanto, conforme apresentado na seção 3, é necessário que os discos rígidos (ou imagens de disco) das MV's estejam compartilhados na rede por meio de algum protocolo. No caso desta implementação foi utilizado o NFS, embora qualquer outro protocolo possa ser utilizado sem precisar realizar nenhuma modificação no algoritmo.

Na implementação assume-se que todas as máquinas físicas possuem a mesma especificação. Ou seja, ao receber uma informação com o uso das máquinas virtuais, caso uma MV utilize 50% de um recurso na máquina A, ela também utilizaria os mesmos 50% desse recurso estando na máquina B.

Além disso, é feita uma restrição na qual cada máquina física pode receber apenas uma máquina virtual por migração. Desse modo, mesmo que o algoritmo avalie que devem ser migradas duas máquinas virtuais, cada uma terá que ser enviada para uma máquina física diferente. Apesar desta restrição, as migrações são não bloqueantes, ou seja, caso mais de uma máquina física esteja sobrecarregada, os dois processos de migração poderão ocorrer em paralelo, desde que a máquina física destino de cada MV a ser migrada seja diferente.

A figura 6.1 ilustra a troca de mensagens entre as máquinas ao longo da execução do algoritmo. O primeiro quadro indica a mensagem enviada por uma máquina física ao notar que está sobrecarregada, enviando para a máquina central os dados das suas máquinas virtuais (*VM_INFO*). No segundo quadro a máquina central envia um pedido das informações (CPU, memória e rede) de todas as máquinas monitoradas (*SEND_INFO*), que é respondido no terceiro quadro (*INFO*). Após a análise dos dados, o quarto quadro exibe o envio da mensagem de migração pela máquina central para a máquina sobrecarregada, com os dados das máquinas virtuais a serem migradas e o destino de cada uma (*MIGRATE*). O quinto quadro apresenta a máquina virtual sendo migrada e, por último, no sexto quadro, a máquina que antes estava sobrecarregada envia uma mensagem para a máquina central informando que a migração foi finalizada (*MIGRATION_FINISHED*).

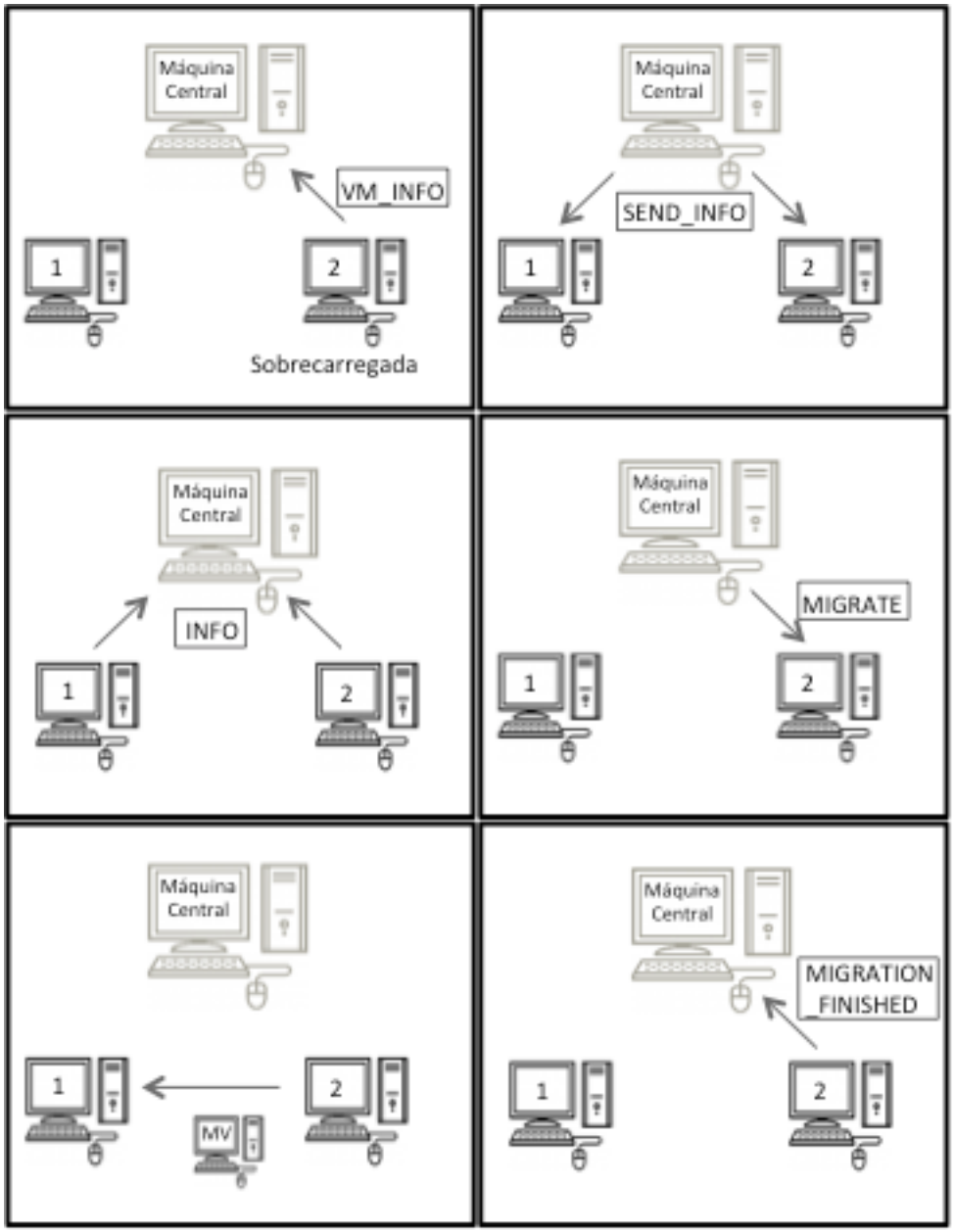

FIG. 6.1 – Sequência de envio de mensagens na execução do algoritmo.

7 EXPERIMENTOS E ANÁLISE DE RESULTADOS

Inicialmente, para a realização dos experimentos, foi realizada a preparação do ambiente. Foram preparadas quatro máquinas físicas, sendo uma a máquina central (que também funcionou como servidor NFS) e três máquinas hospedando máquinas virtuais. A máquina central possui endereço de IP (*Internet Protocol*) 192.168.91.180 e as outras três máquinas possuem IP 192.168.91.61, 192.168.91.59 e 192.168.91.39. Para facilitar, as máquinas serão referenciadas nesta seção pelo final de seu IP (no caso, 39, 59, 61 e 180). Todas as máquinas físicas possuem o processador Intel(R) Core(TM)2 Duo CPU E7500 @ 2.93GHz, 2 GB de memória RAM e uma interface de rede com capacidade de 1 Gbps.

A máquina 59 hospedava três máquinas virtuais: *centos59* (786 MB de memória), *centos59_2* (786 MB de memória) e *centos59_3* (512 MB de memória). A máquina 61 possuía duas máquinas virtuais: *centos61* (786 MB de memória) e *centos61_2* (786 MB de memória). A máquina 39 possuía apenas uma MV: *centos39* (786 MB de memória). A figura 7.1 ilustra o ambiente dos experimentos. A diferença de memória adicionada nas máquinas virtuais da máquina 59 tem o objetivo de observar o comportamento do algoritmo em relação ao volume e imagem de cada máquina virtual.

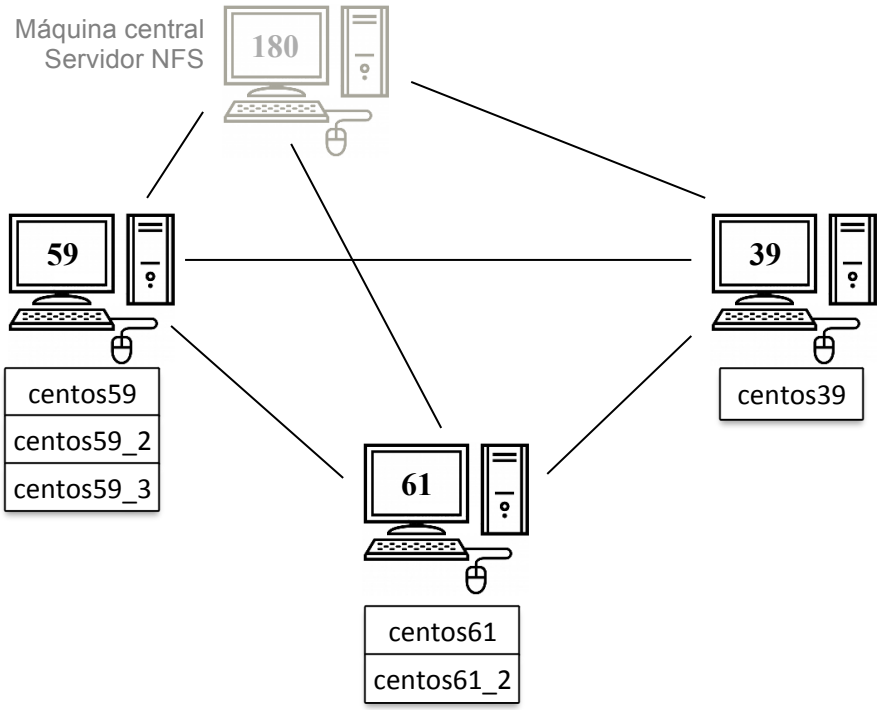

FIG. 7.1 – Os computadores utilizados no ambiente dos experimentos e suas respectivas máquinas virtuais, além da máquina central

Foram realizados três experimentos: no primeiro deles foi analisada a decisão tomada na fase 3 do algoritmo, na qual ele não deve optar por realizar a migração para a máquina que está com menor carga e sim para uma que está com o máximo de carga capaz de suportar essa MV. No segundo experimento foi feita uma grande sobrecarga momentânea de CPU (situação de pico temporário) para mostrar que o algoritmo se protege dessas situações, não gerando uma migração desnecessária. No terceiro experimento foram realizadas cargas em todas as máquinas virtuais, fazendo com que o algoritmo realize as migrações necessárias para balancear as cargas.

7.1 EXPERIMENTO 1

Neste experimento foi realizada uma sobrecarga na CPU da máquina virtual *centos59*, utilizando o programa *'stress'* (STRESS, 2013) que é um gerador de carga para sistemas POSIX. A máquina 61 estava com uma carga mediana e a 39 com pouquíssima utilização. Pode ser analisado pela tabela 7.1 que, no instante em que a máquina 59 ficou sobrecarregada, foi iniciada uma migração da máquina virtual *centos59_3* para a máquina física 61. Portanto, a MV migrada é a que possuía menor custo de migração, conforme definido na seção 5.2, e a máquina destino não era a menos utilizada, conforme determinado na seção 5.3.

A tabela 7.2 mostra o uso dos recursos nas máquinas virtuais no momento da sobrecarga, além dos cálculos do volume, imagem e custo das máquinas. A análise dessa tabela mostra que a máquina migrada (*centos59_3*) era a de menor custo.

TAB 7.1 (a) e (b) - Valores de CPU acumulado e instantâneo das máquinas 59 e 61 e a seta indicando o momento da migração

MF 59		
Horário	CPU_{n+1} (%)	CPU (%)
15:40:33	62.54	84.90
15:40:38	74.52	82.50
15:40:43	75.83	76.70
15:40:48	79.05	81.20
15:40:53	81.06	82.40
15:40:58	81.86	82.40
15:41:03	79.13	77.30
15:41:08	86.43	91.30
15:41:13	91.39	94.70
Centos59_3 migrada para MF 61		
15:41:41	84.62	82.30
15:41:46	83.29	82.40
15:41:51	80.90	79.30
15:41:56	79.76	79.00
15:42:01	81.34	82.40
15:42:06	82.04	82.50
15:42:11	80.75	79.90
15:42:16	79.22	78.20
15:42:21	81.13	82.40

(a)

MF 61		
Horário	CPU_{n+1} (%)	CPU (%)
15:40:34	40.68	40.70
15:40:39	40.75	40.80
15:40:44	40.72	40.70
15:40:49	40.77	40.80
15:40:54	42.65	43.90
15:40:59	46.04	48.30
15:41:04	45.96	45.90
15:41:09	45.98	46.00
15:41:14	53.31	58.20
15:41:19	61.77	67.40
15:41:24	65.03	67.20
15:41:29	63.21	62.00
15:41:34	62.60	62.20
15:41:39	64.88	66.40
15:41:44	66.27	67.20
15:41:49	63.59	61.80
15:41:54	62.70	62.10
15:41:59	64.68	66.00
15:42:04	66.19	67.20

(b)

TAB 7.1 (c) - Valores de CPU acumulado e instantâneo da máquina 39

MF 39		
Horário	CPU_{n+1} (%)	CPU (%)
15:40:15	1,03	1,10
15:40:20	0,95	0,90
15:40:25	0,98	1,00
15:40:30	0,87	0,80
15:40:35	1,01	1,10
15:40:40	0,88	0,80
15:40:45	1,07	1,20
15:40:50	0,91	0,80
15:40:55	0,96	1,00
15:41:00	0,99	1,00
15:41:05	0,87	0,80
15:41:10	0,95	1,00
15:41:15	0,92	0,90
15:41:20	1,03	1,10
15:41:25	1,01	1,00
15:41:30	1,00	1,00
15:41:35	0,94	0,90
15:41:40	1,10	1,20
15:41:45	0,98	0,90
15:41:50	0,93	0,90
15:41:55	0,97	1,00
15:42:00	0,93	0,90
15:42:05	1,03	1,10

(c)

TAB 7.2 – Uso das máquinas virtuais da MF 59 no momento da sobrecarga

MF 59						
MV	CPU	Memória	Rede	Volume	Imagem	Custo
centos59	29,84	49,38	0,00	2,81	388,12	54,10
centos59_2	37,20	10,40	0,00	1,77	81,74	17,67
centos59_3	16,89	9,88	0,00	1,33	50,58	11,82

7.2 EXPERIMENTO 2

Novamente foi utilizado o programa '*stress*' para gerar uma grande carga de CPU nas máquinas virtuais da máquina física 59. Após essa grande carga, o comando foi interrompido,

causando apenas um pico de CPU na MF. Essa situação ilustra um caso onde a migração não é necessária. Pode ser observado na tabela 7.3 que nos instantes 16:18:52 e 16:18:57 a máquina física 59 estava com 94.9% e 94.8% de utilização instantânea da CPU, o que foi reduzido para 82.5% em seguida. O algoritmo alcançou o valor acumulado máximo de apenas 81.87%. Isso demonstra que o algoritmo consegue evitar a realização de migrações desnecessárias, utilizando a média exponencial apresentada na seção 5.1.

TAB 7.3 – Valores de CPU acumulado e instantâneo da máquina 59. Os instantes de pico estão destacados

MF 59		
Horário	CPU_{n+1} (%)	CPU (%)
16:17:51	1.17	1.1
16:17:56	1.13	1.1
16:18:01	46.23	76.3
16:18:06	70.87	87.3
16:18:11	80.67	87.2
16:18:16	84.71	87.4
16:18:21	86.26	87.3
16:18:26	60.85	43.9
16:18:31	25.48	1.9
16:18:36	11.21	1.7
16:18:42	5.5	1.7
16:18:47	13.84	19.4
16:18:52	**62.48**	**94.9**
16:18:57	**81.87**	**94.8**
16:19:02	82.25	82.5
16:19:07	82.4	82.5
16:19:12	82.64	82.8

7.3 EXPERIMENTO 3

Neste experimento foi implementado um *script* que foi executado em todas as máquinas virtuais. Esse *script* escolhia de forma aleatória uma outra máquina virtual e copiava um arquivo entre elas através do comando *scp*. Dessa forma, havia um consumo de rede e CPU em todas as MV's.

Analisando a tabela 7.4, pode ser observado que ao longo da utilização das máquinas três migrações foram realizadas. No instante 22:21:51 é realizada a primeira migração, com origem na máquina 61. A MV migrada foi a *centos61* e, como visto na tabela 7.5 ela é a que

possui menor custo. A MF destino dessa migração foi a 39, e, apesar dela possuir menor uso do que a 59, o que tornaria ela uma não candidata a receber a migração, pode ser observado que no instante da sobrecarga, a MF 59 está com 75,80% de CPU ocupada e, pela tabela 7.5, a MV migrada possuía uso de 25,11% de CPU. Por esse motivo, a máquina 59 não poderia suportar essa MV e a máquina escolhida como destino da migração foi a 39.

A segunda migração ocorreu no instante 22:30:27 e teve como origem a MF 59. A MV migrada foi a *centos59*, a qual possuía o menor custo de migração, como apresentado na tabela 7.6. A máquina que recebeu essa migração foi a 39, que, como visto no instante da migração, possuía maior uso dos recursos em relação à máquina 61 e tinha capacidade de suportar a MV *centos59*.

A última migração ocorreu no instante 22:35:41, migrando a MV *centos59* para a máquina 61. Como mostrado na tabela 7.7, essa MV possuía o menor custo de migração e, mesmo a MF 61 sendo mais utilizada que a 59, esta última estava com uso de CPU de 73,60% e a MV migrada estava utilizando 26,39%. Portanto, a única máquina que suportaria a migração era a 61.

Pode ser observado na tabela 7.4 (c) que a carga gerada pelas máquinas virtuais foi bastante variada. Nota-se, por exemplo, no instante 22:38:03, o consumo de CPU variando de 65,1% para 15,9% e voltando para 63,6% em apenas 10 segundos. Poucas aplicações se assemelham a uma situação como essa. Uma alternativa neste caso, para manter maior estabilidade no valor acumulado calculado, seria utilizar um valor de μ menor. Desta forma, seria levado mais em consideração os valores passados no algoritmo.

TAB 7.4 (a) - Valores de CPU, memória e rede acumulado e instantâneo da máquina 59

MF 59						
Horário	CPU_{n+1} (%)	Mem_{n+1} (%)	$Rede_{n+1}$ (%)	CPU (%)	Mem (%)	Rede (%)
22:19:06	72,52	46,00	10,55	74,40	46,00	10,49
22:19:36	77,08	45,81	8,67	80,80	45,80	8,25
22:20:06	73,22	45,65	8,53	70,30	45,70	6,28
22:20:36	72,58	45,70	10,07	76,60	45,70	9,96
22:21:06	83,39	45,70	12,49	89,50	45,70	10,85
22:21:36	80,98	45,82	12,85	79,80	45,80	13,01
22:21:51	76,62	45,81	12,39	75,80	45,80	12,66
22:22:21	63,70	45,80	5,11	59,60	45,80	4,42
22:22:51	68,52	45,80	9,36	75,40	45,80	11,81
22:23:21	69,51	45,87	9,35	66,10	45,90	10,05
22:24:21	61,74	45,81	8,75	57,70	45,80	8,42
22:25:21	69,28	45,61	7,13	63,60	45,60	8,18
22:26:21	67,68	45,80	11,47	64,00	45,80	13,02
22:27:22	78,56	45,71	9,09	77,40	45,70	8,23
22:28:22	70,21	45,80	9,80	66,70	45,80	9,06
22:29:22	70,51	45,80	9,10	67,50	45,80	7,72
22:30:22	80,93	45,83	11,21	77,70	45,90	11,43
22:30:27	90,15	45,87	11,43	96,30	45,90	11,57
centos59 migrada para MF 39						
22:31:15	73,60	38,70	4,40	66,60	38,70	0,20
22:31:20	74,44	38,70	5,76	75,00	38,70	6,67
22:31:50	71,95	38,74	8,04	64,10	38,70	5,25
22:32:50	84,46	38,49	10,17	85,30	38,50	9,85
22:33:50	78,88	38,60	6,03	81,50	38,60	7,44
22:34:50	75,45	38,40	8,99	76,70	38,40	9,33
22:35:40	72,65	38,40	2,95	73,60	38,40	1,32
22:36:40	78,70	38,44	9,42	77,50	38,40	8,73
22:37:40	70,08	38,40	4,12	71,80	38,40	4,19
22:38:40	67,50	38,48	5,51	62,60	38,50	3,97
22:39:40	68,31	38,50	11,01	70,00	38,50	11,09
22:40:40	59,17	38,40	9,13	57,60	38,40	9,70

(a)

TAB 7.4 (b) - Valores de CPU, memória e rede acumulado e instantâneo da máquina 61

MF 61						
Horário	CPU_{n+1} (%)	Mem_{n+1} (%)	$Rede_{n+1}$ (%)	CPU (%)	Mem (%)	Rede (%)
22:19:06	74,98	24,38	19,32	69,10	24,40	16,72
22:19:36	65,90	24,42	15,30	64,80	24,40	15,76
22:20:06	76,36	24,46	20,10	88,90	24,50	24,49
22:20:36	72,91	24,38	18,56	73,00	24,40	19,57
22:21:06	72,40	24,03	13,67	81,50	24,00	15,47
22:21:36	73,15	24,19	13,65	79,00	24,20	14,29
22:21:41	70,06	24,32	10,29	68,00	24,40	8,05
22:21:46	80,82	24,37	15,36	88,00	24,40	18,74
22:21:51	90,17	24,39	17,19	96,40	24,40	18,40
centos61 migrada para MF 39						
22:22:42	60,34	16,54	11,52	53,50	16,50	9,77
22:22:47	51,38	16,52	10,20	45,40	16,50	9,33
22:23:17	35,98	16,20	8,69	27,70	16,20	6,68
22:24:17	46,98	16,38	9,97	38,30	16,40	6,78
22:25:17	37,61	16,05	7,87	27,70	16,00	5,32
22:26:17	51,26	16,38	13,60	42,60	16,40	12,10
22:27:17	61,99	16,29	15,42	64,80	16,30	15,84
22:28:17	31,00	16,30	6,93	31,40	16,30	6,98
22:29:17	46,62	16,19	11,00	40,10	16,20	9,39
22:30:17	27,77	16,50	4,97	30,50	16,50	5,46
22:30:27	30,84	16,50	6,56	39,40	16,50	9,09
22:31:27	20,12	16,12	5,51	12,70	16,00	4,60
22:32:27	36,13	16,30	9,09	44,60	16,30	12,59
22:33:27	38,60	16,16	9,36	38,10	16,20	10,19
22:34:27	37,76	16,39	8,39	35,30	16,40	6,97
22:35:27	51,98	19,82	16,84	51,70	20,70	17,11
22:35:43	34,62	24,06	12,99	25,70	24,80	9,33
22:35:48	53,09	25,29	18,69	65,40	26,10	22,48
22:35:58	77,75	26,79	20,25	90,90	27,30	23,45
22:36:28	68,70	27,40	17,04	62,60	27,40	14,76
22:37:28	83,01	25,72	18,16	83,50	25,60	19,27
22:38:28	83,33	24,42	17,43	93,40	24,20	18,97
22:39:28	75,09	23,59	19,29	73,10	23,60	20,74
22:40:28	78,69	23,88	20,05	76,60	23,90	19,68

(b)

TAB 7.4 (c) - Valores de CPU, memória e rede acumulado e instantâneo da máquina 39

MF 39						
Horário	CPU_{n+1} (%)	Mem_{n+1} (%)	$Rede_{n+1}$ (%)	CPU (%)	Mem (%)	Rede (%)
22:19:05	45,57	19,56	6,10	53,50	19,60	7,53
22:19:35	27,75	19,60	3,19	29,60	19,60	3,53
22:20:35	47,72	19,60	6,87	52,10	19,60	7,82
22:21:05	52,39	19,68	8,07	57,00	19,70	9,09
22:21:35	29,61	19,70	3,52	21,70	19,70	2,18
22:21:50	59,00	19,76	9,69	62,00	19,80	10,90
22:22:20	49,28	19,71	6,78	53,60	19,70	7,68
22:22:50	36,06	26,31	13,51	31,40	27,20	11,96
22:23:20	66,57	31,37	9,94	70,00	31,50	11,96
22:24:20	61,73	31,63	7,64	57,90	31,60	7,72
22:25:20	55,87	31,49	6,52	49,30	31,50	4,44
22:26:20	74,42	31,28	8,06	87,70	31,40	8,96
22:27:20	55,56	31,06	6,54	50,40	31,10	5,47
22:28:20	60,36	31,10	4,27	61,80	31,10	1,70
22:29:20	72,85	31,31	9,49	76,10	31,30	9,62
22:30:20	67,82	30,78	6,59	65,20	30,60	6,28
22:30:25	57,55	30,55	6,46	50,70	30,40	6,37
22:30:55	69,72	34,45	17,96	60,70	35,20	17,30
22:31:55	76,14	39,56	9,31	76,70	39,40	9,46
22:32:55	61,11	38,65	8,47	60,70	38,60	10,17
22:33:55	76,59	38,88	8,53	70,00	38,90	9,63
22:34:56	82,09	37,09	12,62	81,90	37,10	12,68
22:35:26	80,92	36,82	9,47	82,50	36,80	10,51
22:35:36	78,92	36,66	13,07	80,20	36,60	14,25
22:35:41	91,21	36,38	11,41	99,40	36,20	10,30
centos59 migrada para MF 61						
22:36:27	70,30	27,74	12,11	68,30	27,70	11,25
22:36:32	57,82	27,66	8,50	49,50	27,60	6,10
22:37:33	64,81	27,60	7,98	63,50	27,60	6,14
22:38:33	70,35	27,48	8,80	78,00	27,50	10,15
22:39:33	76,89	27,55	9,35	77,20	27,60	9,01
22:40:33	70,06	27,51	7,35	73,20	27,50	7,45

(c)

TAB 7.5 – Uso das máquinas virtuais da MF 61 no momento da primeira sobrecarga

MF 61						
MV	CPU	Memória	Rede	Volume	Imagem	Custo
centos61	25,11	12,43	6,81	1,63	97,71	19,03
centos61_2	31,14	12,54	7,85	1,80	98,57	19,88

TAB 7.6 – Uso das máquinas virtuais da MF 59 no momento da segunda sobrecarga

MF 59						
MV	CPU	Memória	Rede	Volume	Imagem	Custo
centos59	11,65	11,69	4,46	1,34	91,89	16,94
centos59_2	13,66	11,79	3,37	1,35	92,73	17,12
centos59_3	40,16	20,65	10,29	2,34	105,73	23,06

TAB 7.7 – Uso das máquinas virtuais da MF 39 no momento da terceira sobrecarga

MF 39						
MV	CPU	Memória	Rede	Volume	Imagem	Custo
centos39	16,19	12,75	4,35	1,43	100,24	18,31
centos61	28,71	11,85	7,61	1,72	93,20	18,88
centos59	26,39	11,05	6,68	1,63	86,90	17,74

8 CONCLUSÃO E TRABALHOS FUTUROS

Neste trabalho foi desenvolvido um algoritmo de balanceamento de carga em um ambiente de *cloud computing*, que possui como características ser *black-box*, centralizado e estático.

As análises realizadas na seção 7 mostram que os resultados dos experimentos estão de acordo com o esperado no desenvolvimento do algoritmo. Pode ser observado na seção 7.1 que a proposta estabelecida na seção 5.3 (fase 3 do algoritmo) conseguiu ser cumprida, visto que a máquina destino escolhida para receber a migração não foi a que estava com pouco uso dos recursos, reservando as máquinas menos ocupadas para receber máquinas virtuais que exigem mais recursos. Essa política pode causar uma certa ociosidade nas máquinas menos utilizadas e poderia ser utilizado para economia de energia. Ao se analisar as máquinas ociosas durante um longo período, suas máquinas virtuais podem ser migradas para que a máquina física seja desligada.

Na seção 7.2 é possível notar que o cálculo do valor acumulado, apresentado na seção 5.1 (fase 1 do algoritmo), é uma política válida, pois permitiu que o algoritmo evitasse uma migração desnecessária em um caso de pico do uso de um recurso.

Nas seções 7.1 e 7.3 a escolha da máquina virtual a ser migrada segue conforme determinado na seção 5.2 (fase 2 do algoritmo). As MV's escolhidas para a migração são aquelas que possuem o menor custo, ou seja, aquelas que irão afetar o menor número de usuários possível.

Ainda sobre o experimento realizado na seção 7.3, os valores instantâneos de uso de CPU apresentaram grande variação. Neste cenário, o valor de $\mu = 0{,}6$ não se mostrou apropriado, visto que o valor acumulado também apresenta grande variação. A alternativa é utilizar um valor de μ menor, que dê menos peso para o valor instantâneo do recurso. Conclui-se, portanto, que o coeficiente μ deve ser ajustado de acordo com o cenário em que o algoritmo se encontra.

Também é importante ressaltar que a implementação de uma arquitetura centralizada, como implementada neste trabalho, tem a vantagem de transmitir um número fixo e menor de pacotes pela rede, embora adicione o risco de um ponto único de falha (no caso da máquina central falhar). Por outro lado, em uma arquitetura distribuída, para que cada máquina não

fique com informações desatualizadas, é necessário um número mais elevado de troca de mensagens, o que poderia causar sobrecarga na rede em um ambiente de maior escala.

Uma proposta interessante para o avanço desse trabalho seria comparar o algoritmo desenvolvido com a sua implementação utilizando uma arquitetura distribuída, de modo a analisar os ganhos e perdas das duas abordagens. Após isso, poderia ser comparado esse algoritmo com outros já existentes, embora não se tenham muitas referências sobre o assunto.

Além disso, seria relevante uma análise em cenários variados do melhor coeficiente a ser utilizado no algoritmo em cada caso. Como mostrado na seção 7.3, o uso de μ=0,6 não é o mais adequado na situação daquele experimento. Portanto, em diversas situações específicas, seria importante adaptar o coeficiente para a realidade do usuário.

Por último, algumas restrições impostas na implementação poderiam ser excluídas, como o fato de só poderem ser utilizadas máquinas físicas com as mesmas especificações e não ocorrerem migrações simultâneas da mesma MF, no caso de necessitar migrar mais de uma MV. Outra sugestão de implementação é utilizar o protocolo iSCSI para compartilhamento dos discos rígidos, visto que a utilização do NFS, embora mais simples, não se mostra escalável em ambientes com um maior número de máquinas (SHARED STORAGE, 2013).

9 REFERÊNCIAS BIBLIOGRÁFICAS

POPEK, Gerald J.; GOLDBERG, Robert P., **Formal requirements for virtualizable third generation architectures**. Communications of the ACM, v.17 n.7, p.412-421, July 1974

TANENBAUM, A. S.; STEEN, M. V. **Distributed Systems: Principles and Paradigms (2nd Edition)**. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 2006

VOORSLUYS, W; BROBERG, J.; VENUGOPAL, S.; BUYYA, R. **Cost of Virtual Machine Live Migration in Clouds: A Performance Evaluation**. Proceedings of the 1st International Conference on Cloud Computing, December 01-04, 2009, Beijing, China

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

SMITH, J. E.; NAIR, R. **The Architecture of Virtual Machines**. Computer, v. 38, n. 5, pp. 32–38, 2005. ISSN: 0018-9162

LINDHOLM, T.; YELLIN, F. **The Java Virtual Machine Specification, 2nd ed**. Addison-Wesley, 1999

Linux Foundation. Xen Project. Disponível em: <www.xen.org>. Acessado em 20/09/2012.

VMware. VMware. Disponível em: <www.vmware.com>. Acessado em 20/09/2012.

BARHAM, P.; DRAGOVIC, B.; FRASER, K.; HAND S.; HARRIS, T.; HO, A.; NEUGEBAUER, R.; PRATT, I.; WARFIELD, A. **Xen and the art of virtualization.** In Proceedings of the nineteenth ACM symposium on Operating systems principles (SOSP '03). ACM, New York, NY, USA, 2003.

CLARK, C.; FRASER, K.; HAND, S.; HANSEN, J.G.; JUL, E.; LIMPACH, C.; PRATT, I.; WARFIELD, A.; **Live Migration of Virtual Machines.** Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation, p.273-286, May 02-04, 2005

WILCOX, T. C. **Dynamic Load Balancing Of Virtual Machines Hosted On Xen.** Department of Computer Science, Brigham Young University, Dissertação (Mestrado em Ciência da Computação), Abril, 2009.

WOOD, T.; SHENOY, P.; VENKATARAMANI, A.; YOUSIF, M.; **Black-box and gray-box strategies for virtual machine migration.** Proceedings of the 4th USENIX conference on Networked systems design & implementation, p.17-17, April 11-13, 2007, Cambridge, MA

CHOI, H. W.; KWAK, H.; SOHN, A. **Autonomous learning for efficient resource utilization of dynamic VM migration.** Proceedings of the 22nd annual international

conference on Supercomputing, pp. 185–194, New York, NY, USA, 2008. ACM. ISBN: 978-1-60558-158-3

ZHAO, Y.; HUANG, W. **Adaptive Distributed Load Balancing Algorithm Based on Live Migration of Virtual Machines in Cloud.** NCM '09 Proceedings of the 2009 Fifth International Joint Conference on INC, IMS and IDC, pp. 170–175, Washington, DC, USA, 2009. IEEE Computer Society. ISBN: 978-0-7695-3769-6

Google. psutil - A cross-platform process and system utilities module for Python. Disponível em: <https://code.google.com/p/psutil/>. Acessado em: 19/03/2013.

Red Hat. Libvirt: The virtualization API. Disponível em: <http://libvirt.org/>. Acessado em: 17/03/2013.

HONG, B. **Cobb-Douglas Production Function.** 2008

KHANNA, G.; BEATY, K.; KAR, G.; KOCHUT, A. **Application performance management in virtualized server environments,** in Proceedings of 10th IEEE/IFIP Network Ops and Management Symp. (NOMS 2006), 2006.

SILBERSCHATZ, A.; GALVIN, P. B.; GAGNE, G. **Operating System Concepts (7th Edition)**. John Wiley & Sons, Inc., NJ, USA, 2005

Amos Waterland. Stress. Disponível em: <http://weather.ou.edu/~apw/projects/stress/>. Acessado em: 29/05/2013.

Redhat. Shared storage example: NFS for a simple migration. Disponível em: <https://access.redhat.com/site/documentation/en-US/Red_Hat_Enterprise_Linux/6/html/Virtualization_Administration_Guide/shared-storage-nfs-migration.html>. Acessado em: 02/06/2013.