UNIVERSIDADE FEDERAL DE SANTA MARIA
CENTRO DE TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

ESPECIFICAÇÃO DE UMA ARQUITETURA PARA MIGRAÇÃO DE MÁQUINAS VIRTUAIS UTILIZANDO ONTOLOGIAS

DISSERTAÇÃO DE MESTRADO

Rafael Barasuol Rohden

Santa Maria, RS, Brasil

2015

ESPECIFICAÇÃO DE UMA ARQUITETURA PARA MIGRAÇÃO DE MÁQUINAS VIRTUAIS UTILIZANDO ONTOLOGIAS

Rafael Barasuol Rohden

Dissertação apresentada ao Curso de Mestrado Programa de Pós-Graduação em Informática (PPGI), Área de Concentração em Computação, da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para obtenção do grau de

Mestre em Ciência da Computação

Orientador: Prof. Dr. Benhur de Oliveira Stein

Co-orientador: Prof. Dr. Giovani Rubert Librelotto

Santa Maria, RS, Brasil

2015

Barasuol Rohden, Rafael

Especificação de Uma Arquitetura para Migração de Máquinas Virtuais Utilizando Ontologias / por Rafael Barasuol Rohden. – 2015.
100 f.: il.; 30 cm.

Orientador: Benhur de Oliveira Stein
Co-orientador: Giovani Rubert Librelotto
Dissertação (Mestrado) - Universidade Federal de Santa Maria, Centro de Tecnologia, Programa de Pós-Graduação em Informática, RS, 2015.

1. Máquina Virtual. 2. Migração de Máquinas Virtuais. 3. Ontologias. I. Stein, Benhur de Oliveira. II. Librelotto, Giovani Rubert. III. Título.

E-mail: rafaelrohden@gmail.com

Universidade Federal de Santa Maria
Centro de Tecnologia
Programa de Pós-Graduação em Informática

A Comissão Examinadora, abaixo assinada,
aprova a Dissertação de Mestrado

ESPECIFICAÇÃO DE UMA ARQUITETURA PARA MIGRAÇÃO DE MÁQUINAS VIRTUAIS UTILIZANDO ONTOLOGIAS

elaborada por
Rafael Barasuol Rohden

como requisito parcial para obtenção do grau de
Mestre em Ciência da Computação

COMISSÃO EXAMINADORA:

Benhur de Oliveira Stein, Dr.
(Presidente/Orientador)

Andrea Schwertner Charão, Dr. (UFSM)

Luiz Angelo Steffenel, Dr. (URCA)

Santa Maria, 23 de Julho de 2015.

A Deus.
À minha família.
À minha eterna namorada.

AGRADECIMENTOS

Gostaria de agradecer a Deus pela vida, por me dar paciência e tudo aquilo pelo qual me motiva em viver a vida à qual Ele me deu.

Aos meus pais pelo carinho com que me deram desde o início de minha vida. Aos exemplos de caráter, bondade, e de dedicação e paixão pela vida.

Ao meu irmão pelo companheirismo nas longas conversas sobre o que de melhor um filme ou seriado nos pode proporcionar intelectualmente, ou não. Também, pela paixão que temos e a alegria em falar sobre O Senhor dos Anéis. O melhor filme, do melhor livro, de todos os tempos.

À minha namorada que me trouxe um motivo a mais para lutar pelo meu futuro profissional. Que teve paciência em minhas crises em que quase desisti do Mestrado. Pelo carinho e preocupação que sempre tem comigo.

Enfim, aos meus professores pelos ensinamentos. Principalmente aos meus orientadores Benhur e Giovani.

“A mente que se abre a uma nova ideia jamais voltará ao seu tamanho original.”
— ALBERT EINSTEIN

RESUMO

Dissertação de Mestrado
Programa de Pós-Graduação em Informática
Universidade Federal de Santa Maria

ESPECIFICAÇÃO DE UMA ARQUITETURA PARA MIGRAÇÃO DE MÁQUINAS VIRTUAIS UTILIZANDO ONTOLOGIAS

AUTOR: RAFAEL BARASUOL ROHDEN
ORIENTADOR: BENHUR DE OLIVEIRA STEIN
CO-ORIENTADOR: GIOVANI RUBERT LIBRELOTTO
Local da Defesa e Data: Santa Maria, 23 de Julho de 2015.

A computação em nuvem é um novo campo na computação, sobretudo na Internet, que proporciona novas perspectivas no domínio das tecnologias de interconexões e levanta problemas na arquitetura, design e implementação de redes existentes e de *Data Centers*. Atualmente, através de tecnologia como virtualização de servidores, vem sendo largamente utilizado para disponibilização de serviços por demanda evitando que haja o espalhamento de servidores. Desta forma, os servidores são utilizados de maneira que seus recursos sejam melhores empregados para garantir a disponibilidade de recursos e serviços para os usuários, permitindo assim, que estes usuários acessem serviços baseados em suas necessidades, independentemente de onde os serviços são hospedados ou como eles são entregues. Sendo esta a característica principal da Computação em Nuvem. No entanto, em algum momento servidores podem ficar sobrecarregados e outros podem ficar mais ociosos, e a maneira para resolver isso é utilizando a migração de máquinas virtuais em tempo real, onde ocorre a migração de máquina virtual em execução juntamente com suas aplicações para outro servidor, restabelecendo, assim, o equilíbrio dos servidores. Este equilíbrio, chamado de balanceamento de carga, é uma das técnicas utilizadas pela tecnologia de migração em tempo real. Ou seja, a aplicação de migração de máquinas virtuais em tempo real tem se tornado a chave para a otimização de recursos computacionais. Assim, torna-se interessante o desenvolvimento de soluções que viabilizem a implantação desta tecnologia. Através de um ambiente virtualizado onde aplicações monitores verificam o estado de carga dos servidores é possível interagir com as máquinas virtuais realizando a migração para garantir a otimização e utilização dos recursos computacionais. Considerando isto, o presente trabalho apresenta uma arquitetura para migração de máquinas virtuais, a qual utiliza ontologias para a representação do conhecimento em um ambiente de virtualização. Para isto, foi desenvolvida, através do processo *Ontology Development 101*, uma ontologia, *OntoLM*, que representa um ambiente de virtualização de máquinas virtuais a qual propõe auxiliar a visualização do estado atual do ambiente. Para a arquitetura especificada neste trabalho foi delimitado componentes e seus respectivos fluxos de informações entre um componente e outro. Utiliza-se de ontologias como um de seus componentes. Para a exemplificação desta arquitetura foi desenvolvida uma ferramenta, *OntoMig*, em linguagem de programação JAVA, que permite executar e gerenciar as informações obtidas do monitoramento dos servidores, a população da ontologia e a migração de máquinas virtuais quando necessário.

Palavras-chave: Máquina Virtual. Migração de Máquinas Virtuais. Ontologias.

ABSTRACT

Master's Dissertation
Post-Graduate Program in Informatics
Federal University of Santa Maria

SPECIFICATION OF AN ARCHITECTURE FOR MIGRATION OF VIRTUAL MACHINES USING ONTOLOGIES

AUTHOR: RAFAEL BARASUOL ROHDEN
ADVISOR: BENHUR DE OLIVEIRA STEIN
COADVISOR: GIOVANI RUBERT LIBRELOTTO
Defense Place and Date: Santa Maria, July 23st, 2015.

Cloud computing is a new area in computing, providing new perspectives in the area of interconnect technologies and raises issues in architecture, design and implementation of existing networks and data centers. Currently through technology like server virtualization, has been widely used for providing on-demand services with avoiding the spreading of servers. In this way the servers are used so that its resources be better used to ensure the availability of resources and services for users, enabling, so these users from accessing services based on your needs, regardless of where the services are hosted, or how they are delivered. This being, the main feature of cloud computing. However, some servers become eventually overloaded and others are more idle, and the way to solve this is by using the migration of virtual machines in real time, that is, perform the migration of running virtual machine along with its applications to another server by restoring the balance of the servers. This balance, called load balancing is one of the techniques used by real-time migration technology. That is, the technology of migration of virtual machines in real time has become the key to optimizing computer resources. Thus, it becomes interesting the development of solutions that enable the deployment of this technology. Through a virtualized environment where applications monitors check the load state of the servers it is possible to interact with the virtual machines performing migration to ensure the optimization and utilization of computational resources. Considering this, this work presents an architecture for migration of virtual machines, which uses ontologies for knowledge representation in a virtualization environment. For this, was developed, through the process Ontology Development 101, an ontology, Onto-LM, which represents a virtual machine virtualization environment which offers help to visualize current state of the environment. For the specified architecture in this work was delimited components and their respective information flows between a component and another. Use of ontologies as one of its components. For examples of this architecture has been developed a tool, OntoMig, in the JAVA programming language, which allows to run and manage the information acquired from monitoring of servers, the charge of the ontology and the migration of virtual machines when needed.

Keywords: Virtual Machine. Live Migration. Ontologies..

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE APÊNDICES

LISTA DE ABREVIATURAS E SIGLAS

API	*Application Programming Interface*
APP	*Application Software*
CPU	*Central Processing Unit*
EXT	*Extended File System*
FTP	*File Transfer Protocol*
HTTP	*Hypertext Transfer Protocol*
IDE	*Integrated Development Environment*
KVM	*Kernel-based Virtual Machine*
LXDE	*Lightweight X11 Desktop Environment*
LMCI	*Live Virtual Machine Migration With Less Cost and Application Inference*
LVM	*Logical Volume Manager*
LM	*Live Migration*
NFS	*Network File System*
OWL	*Web Ontology Language*
OS	*Operating System*
OSVD	*Optimal Singular Value Decomposition*
PM	*Physical Machine*
RAM	*Random Access Memory*
RDF	*Resource Description Framework*
SWRL	*Semantic Web Rule Language*
SPARQL	*SPARQL Protocol and RDF Query Language*
SQWRL	*Semantic Query-Enhanced Web Rule Language*
SSH	*Secure Shell*
SLA	*Service Level Agreement*
TLS	*Transport Layer Security*
TCP	*Transmission Control Protocol*
URI	*Uniform Resource Indentifier*
VM	*Virtual Machine*
VSL	*Virtual Server Load*
VNC	*Virtual Network Computing*
XML	*eXtensible Markup Language*

LISTA DE SÍMBOLOS

$\wedge$	Condicional ”E”
$\rightarrow$	Resultado da expressão SWRL
$\sum$	Soma
α	Alfa
β	Beta
γ	Gamma

SUMÁRIO

1 INTRODUÇÃO

Na computação em nuvem, o cliente tem acesso à infraestrutura, a diferentes plataformas e a software em forma de serviço. O cliente acessa esse serviço e paga ou não pelo uso do mesmo. A computação em nuvem geralmente utiliza-se de técnicas de virtualização para atender sua demanda de serviços. Virtualização permite múltiplos servidores virtualizados em um único *host* físico de forma isolada e segura, permitindo, desta forma, uma otimização no uso dos recursos computacionais reduzindo o custo de implantação destes Data Centers (SAGANA; GEETHA; SUGANTHE, 2013).

Recentemente, a aplicação de migração de máquinas virtuais em tempo real, conhecido como *Live Migration* (LM), tem se tornado a chave para a otimização de recursos computacionais em *Data Centers*. LM é o processo de mover uma máquina virtual em execução juntamente com suas aplicações entre diferentes máquinas físicas sem desconectar o cliente e sem parar as aplicações que estão sendo executadas. Os recursos alocados para essa máquina virtual também são transferidos da máquina *host* de origem para máquina *host* de destino. LM é utilizado para alcançar uma melhor eficiência energética, balanceamento de carga de trabalho procurando utilizar melhor os recursos das máquinas físicas, e também auxilia para manter a alta disponibilidade dos serviços (YE et al., 2012).

A LM possui algumas motivações para realizar a migração e através de alguns objetivos permite garantir suas características pelas quais é decidido por seu uso: Migração para Eficiência Energética, Migração para Balanceamento de Carga e Migração para Tolerância a Falhas (KAPIL; PILLI; JOSHI, 2013). As técnicas, que procuram atingir esses objetivos, devem garantir o menor tempo de migração possível, sem interrupção dos serviços ou com o menor tempo de interrupção possível. O balanceamento de carga deve ser transparente ao sistema operacional convidado que está sendo migrado, ou seja, aos clientes e às aplicações que estão sendo executadas. Os parâmetros que afetam o desempenho de uma migração são: tempo total de migração (tempo de migração de uma máquina virtual do *host* origem até o *host* destino) e *downtime* (período de tempo em que a máquina virtual fica inacessível durante a migração) (LEELIPUSHPAM; SHARMILA, 2013).

Existem vários trabalhos que abordam cada um dos objetivos para a utilização de migração de máquinas virtuais. Para balanceamento de carga, Wood (2007) propõe uma técnica onde um motor de criação de perfil coleta informações sobre a utilização de recursos de todas

as máquinas virtuais e outro monitor detecta a máquina virtual que utiliza mais recursos e então migra tais máquinas virtuais de servidores altamente utilizados para servidores menos utilizados. Em Khanna (2006), é calculado o valor de L, que é um produto de custo de migração e utilização dos recursos de migração. As máquinas virtuais são migradas para o servidor físico com menor valor de L. Arzuaga (2010) propõe uma métrica chamada VSL que é obtida pela soma de recursos de máquina física multiplicado pela divisão da soma de recursos utilizados pelas VMs por capacidade de recursos do servidor físico. Essa métrica é usada para medir o desequilíbrio de carga e construir um modelo de migração de VM para atingir o balanceamento de carga.

Entretanto, para que estas técnicas sejam aplicadas é necessário ter conhecimento sobre o ambiente de virtualização, analisar os dados sobre o uso de recursos dos servidores, verificar a necessidade de realizar a migração e interagir com as VMs quando existir a necessidade de realizar uma ou mais migrações para atender um dos objetivos pretendidos.

Através de um ambiente virtualizado onde aplicações monitores verificam o estado de carga dos servidores é possível interagir com as máquinas virtuais realizando a migração para garantir a otimização e utilização dos recursos computacionais. Para isto existe uma ferramenta amplamente utilizada para gerenciamento de máquinas virtuais chamada *Libvirt*. Ela oferece uma API agnóstica de hipervisores para gerenciar, de forma segura, sistemas operacionais convidados que estejam sendo executados em um *host*. A *Libvirt* fornece a funcionalidade comum implementada pelos hipervisores suportados, tais como: iniciar, pausar, terminar e migrar uma máquina virtual (LIBVIRT, 2014).

Para representar esse ambiente virtualizado é possível utilizar-se de ontologias, que segundo Huang (2010), são uma forma de representar conhecimento consensual, e por esse motivo elas acabam por ser comumente empregadas como um método para a identificação de categorias, conceitos, relações e regras, para definir e conceituar o conhecimento em um domínio. Assim um programa pode assumir a semântica de um dado relacionamento e atuar sistematicamente através das ontologias.

Nesse sentido, a ideia do trabalho é propor uma arquitetura para migração bem como definir uma ontologia de LM que represente um ambiente de conhecimento onde esteja inserido o uso de migração em tempo real, de forma que esse conhecimento possa ser compartilhado à comunidade de áreas afins.

Não é trivial o desenvolvimento de uma ontologia, visto que envolve o entendimento e

conhecimento especializado na área em que será definido o domínio. Neste trabalho, o domínio entende-se por migração de máquinas virtuais, sendo o foco o balanceamento de carga de máquinas virtuais. Assim, pretende-se realizar um estudo para buscar esse conhecimento e a partir de então, representar esse conhecimento através de uma ontologia para que posteriormente esta possa ser processada por *software*. Foi definida uma arquitetura para exemplificar o ambiente em que a ontologia tem sua utilização e de que maneira ela interage com o ambiente proposto. Para verificação do uso da ontologia foi desenvolvido um estudo de caso e através deste, um cenário onde todo o contexto do trabalho foi unificado: ontologia para o mapeamento do ambiente virtualizado visando a migração de máquinas virtuais inserida em um arquitetura definida, e a comunicação entre o ambiente e a ontologia através de um *software*. Este *software*, com o nome de OntoMig, é desenvolvido também neste trabalho.

O trabalho está organizado da seguinte forma: no capítulo 2 é elaborada a revisão bibliográfica onde computação em nuvem, virtualização e máquina virtual, migração de máquinas virtuais e ontologia são exemplificados; no capítulo seguinte, 3, os trabalhos relacionados a esta dissertação são apresentados para justificar o desenvolvimento deste trabalho; no capítulo 4 é apresentada a ontologia proposta para migração de máquinas virtuais, no qual é apresentado o domínio, as classes, as propriedades de dados e objetos, instâncias e consultas.

Em seguida, no capítulo 5, é exemplificada a arquitetura para migração de máquinas virtuais utilizando a ontologia. São apresentadas a arquitetura e as tecnologias que as constituem, Agente monitor e Aplicação Java, e o fluxo do sistema; no capítulo 7, são apresentados o estudo de caso e os resultados. Neste capítulo o ambiente de virtualização para migração é descrito. Os servidores e as máquinas virtuais são descritas, entre outras coisas. Sobre a Aplicação OntoMig, desenvolvida para este trabalho, é apresentado o fluxo da aplicação e suas principais classes e funções. Sobre os resultados são apresentadas imagens de gráficos referente a testes realizados no ambiente; e por fim no capítulo 8 são discutidas as conclusões deste trabalho.

2 REVISÃO BIBLIOGRÁFICA

Neste capítulo são descritos os principais conceitos e relacionamentos entre as áreas que envolvem um ambiente de virtualização para migração de máquinas virtuais, e, também, as ontologias. Neste capítulo, serão abordados os seguintes assuntos: computação em nuvem; virtualização e sua aplicação em um ambiente de computação em nuvem; os conceitos de máquinas virtuais e arquitetura do sistema de máquina virtual; migração de máquinas virtuais, seus objetivos e técnicas para controle, assim como métricas e termos utilizados nesse tema; e, também, será tratado sobre ontologias como conceito, linguagens de representação, consultas com SPARQL, inferências com SWRL, consultas com SQWRL e bibliotecas de programação para ontologias.

2.1 Computação em Nuvem

Desde o momento em que a internet passou a fazer parte da vida da maioria das pessoas, esta tecnologia vem angariando cada vez mais adeptos e consequentemente mais informação é produzida. Desta forma, ela se expande para tornar-se uma plataforma para empresas, gerar negócios, e para sociedade que usufrui desta tecnologia. Assim, um novo paradigma de computação distribuída surge na vida das pessoas.

A computação em nuvem é um novo campo de pesquisa multidisciplinar, considerada a evolução e convergência de várias tendências de computação independentes, tais como a *Internet Delivery*, utilidade computacional de "pague o que usa" no sentido de que o cliente paga pelo que foi utilizado de recursos, elasticidade, virtualização, computação em grade, computação distribuída, armazenamento, a terceirização de conteúdo, segurança e web 2.0 (PALLIS, 2010).

Segundo Mell (2009) alguns dos principais aspectos da computação em nuvem são: a computação em nuvem é um modelo conveniente para permitir o acesso à rede, sob demanda a um pool compartilhado de recursos computacionais configuráveis (por exemplo, redes, servidores, armazenamento, aplicações e serviços) que podem ser rapidamente provisionados e liberados com um esforço mínimo de gestão ou serviço de interação com o provedor. Esse modelo de nuvem promove disponibilidade e é composto por cinco características essenciais (autoatendimento sob demanda, amplo acesso à rede, *pool* de recursos, rápida elasticidade e

serviço medido), três modelos de serviço (*software* em nuvem como serviço, plataforma em nuvem como um serviço e infraestrutura em nuvem como um serviço), e quatro modelos de implementação (nuvem privada, nuvem em comunidade, nuvem pública e nuvem híbrida).

Assim, para disponibilizar uma enorme gama de serviços através de grandes *Data Centers*, estes munidos de diversos servidores de diferentes sistemas operacionais e serviços, com milhares de aplicativos sendo executados, e ainda assim manter uma alta disponibilidade dos serviços com um baixo consumo de energia, foi então amplamente aplicada a tecnologia de virtualização, que permite o melhor uso dos recursos computacionais.

2.2 Virtualização

A virtualização é uma técnica que permite que vários sistemas operacionais sejam executados simultaneamente em uma única máquina física. Tornou-se um aspecto central em servidores modernos e centros de dados devido a várias vantagens, tais como ao compartilhamento flexível e eficiente dos recursos, tolerância a falhas, portabilidade e eficiência de custos (STRUNK; DARGIE, 2013).

Um dos motivos para a ascensão da virtualização é a evolução da capacidade de processamento e memória dos servidores, que permite executar máquinas virtuais em praticamente qualquer *hardware* disponível atualmente. Outro motivo é a necessidade de economia de energia, espaço e custo nos *Data Centers*. Com a virtualização é possível alcançar essas necessidades através da consolidação em um único *hardware* de servidores que no passado precisariam de suas próprias máquinas físicas.

Em um ambiente virtualizado, as máquinas virtuais que agem como máquinas físicas reais podem ser executadas em paralelo e de forma isolada uma em relação a outra e ainda compartilhar os mesmos recursos físicos. Um *middleware* de baixo nível chamado de hipervisor abstrai estas máquinas virtuais a partir do *hardware* físico e determina o uso exclusivo de recursos para cada VM (STRUNK; DARGIE, 2013). A estrutura de um ambiente de computação de virtualização é apresentado na figura 2.1.

Vale ressaltar que a implementação de máquinas virtuais de sistema ou monitores de máquinas virtuais (VMM) pode ser obtida através de duas técnicas: virtualização completa e a paravirtualização. A virtualização completa consiste em prover uma réplica (virtual) do *hadware* subjacente de tal forma que o sistema operacional e as aplicações podem executar como se tivessem executando diretamente sobre o *hardware* original. Na paravirtualização, o

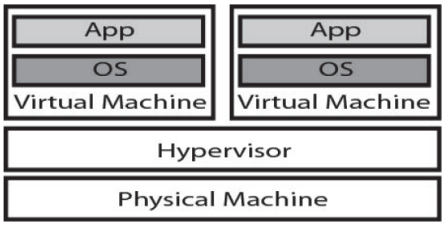

Figura 2.1 – Abstração em camadas de uma virtualização.

sistema hóspede é modificado para chamar a VMM sempre que for executada uma instrução ou ação considerada sensível. Dessa forma, o teste por instrução não é mais necessário. Além disso, na paravirtualização os dispositivos de *hardware* são acessados por *drivers* da própria VMM.

No entanto, a virtualização expõe novos desafios no gerenciamento de VMs, seus recursos e os recursos subjacentes do *host* físico. Em vez de um único mapeamento de sistema operacional para *host* físico, mais de um sistema operacional agora ocupam os recursos de um *host* físico como VMs. Cada VM é representada por um contêiner, que contém um ou mais discos virtuais e outros metadados para descrever a configuração e os limites da VM. Cada VM compartilha os recursos do *host* físico, o que requer não apenas configuração, mas uma compreensão da utilização desses recursos (para garantir uma densidade apropriada de VMs que use os recursos de forma adequada, sem sobrecarregá-los ou desperdiçá-los) (JONES, 2012).

2.3 Máquina Virtual

A simulação de instrução-por-instrução completa de um sistema de computador em um sistema diferente é uma técnica bem conhecida de computação. Muitas vezes é usada para desenvolvimento de *software* quando uma base de *hardware* está sendo alterada. Por exemplo, se um programador está desenvolvendo *software* para um novo computador de propósito especial "X" que está em construção e ainda não está disponível, ele vai provavelmente começar por escrever um simulador para o computador em uma máquina de uso geral disponível "G". (GOLDBERG, 1974).

Ainda, segundo Goldberg (1974), o simulador irá fornecer uma simulação detalhada do ambiente para fins especiais de "X", incluindo o seu processador, memória, e dispositivos de I / O. Os programas em execução no "X" podem ser arbitrários — incluindo código de exercício simulado de dispositivos de e/s, mover dados e instruções em qualquer lugar na memória simulada, ou executar qualquer instrução da máquina simulada. O simulador fornece uma camada

de *software* de filtragem que protege os recursos da máquina “G” impedindo que sejam usados indevidamente por programas em“X”. Alternativamente, uma versão especial, mais poderosa do simulador pode ser desenvolvida, que em si é um sistema de compartilhamento de tempo e oferece suporte a vários usuários. Em ambos os casos, o resultado seria a ilusão de múltiplas cópias da interface de *hardware* e *software* da máquina “X” na máquina “G”.

Nestes sistemas, grande parte do *software* para a máquina simulada executa diretamente no hardware sem interpretação de *software*. Sistemas deste tipo são chamados de sistemas de máquina virtual, as máquinas simuladas são chamadas de máquinas virtuais (VMs) e o software do simulador é chamado o monitor de máquina virtual (VMM).

Sistemas de máquinas virtuais foram originalmente desenvolvidos para corrigir algumas das deficiências das arquiteturas de terceira geração típicas e sistemas operacionais multiprogramação.

As principais características da arquitetura desses sistemas foram a organização de *hardware* com o modo privilegiado e de um modo não-privilegiado. No modo privilegiado todas as instruções estão disponíveis para o *software*, enquanto que no modo não privilegiado não são.

O sistema operacional fornece um pequeno programa residente chamado núcleo de *software* privilegiado. Programas de usuário podem executar as instruções de *hardware* não privilegiados ou fazer chamadas de supervisão para o núcleo de *software* privilegiado para ter funções privilegiadas realizadas em seu nome. O conjunto de instruções não privilegiadas, juntamente com as ligações de controle define efetivamente uma máquina estendida, o que é semelhante, mas não idêntico, ao da máquina *host* original. A máquina estendida é, em teoria, melhor engenharia humana e mais fácil de programar que a máquina *host* inicial.

O coração de um sistema VM é o *software* monitor de máquina virtual (VMM) que transforma a interface de máquina única na ilusão de muitos. Cada uma dessas interfaces (máquinas virtuais) é uma réplica eficiente do sistema de computador original, completo com todas as instruções do processador (tanto instruções privilegiados e não privilegiadas) e recursos do sistema (memória e dispositivos de Entrada e Saída). Ao executar cada sistema operacional em sua própria máquina virtual, torna-se possível executar vários sistemas operacionais diferentes (núcleos de software privilegiados) simultaneamente, como é apresentado pela figura 2.2.

De maneira mais clara Li (2010) exemplifica que na arquitetura, apresentada na figura 2.3, várias máquinas virtuais (VMs) compartilham a mesma máquina “física”. O nível mais baixo, logo acima da camada de *hardware*, o *kernel* do sistema operacional hospedeiro

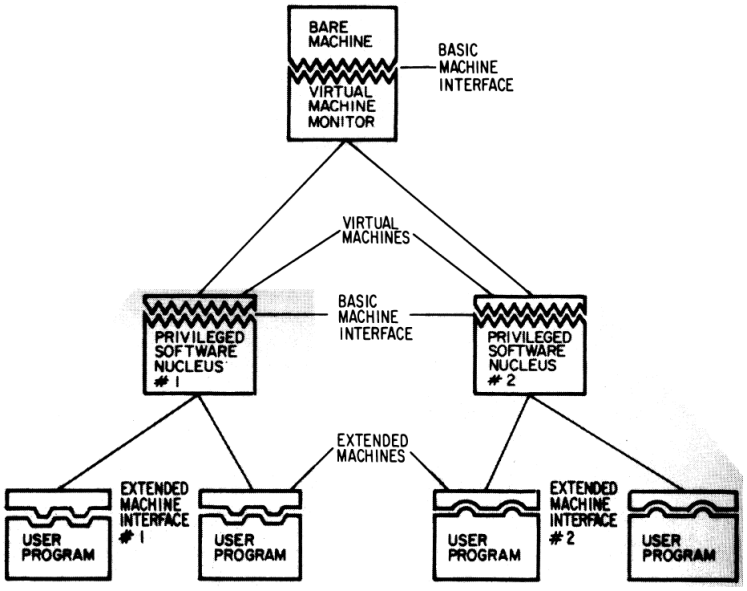

Figura 2.2 – Arquitetura de Virtualização (GOLDBERG, 1974).

ou monitor de máquina virtual (VMM) fornece alocação de recursos para máquinas virtuais. Em cada máquina virtual, várias tarefas (ou serviços) são executadas logo acima do sistema operacional, que por sua vez fornece o habitual, ou seja, um conjunto de abstrações de alto nível como suporte de rede e acesso de arquivo para aplicativos que estão sendo executados nas máquinas virtuais.

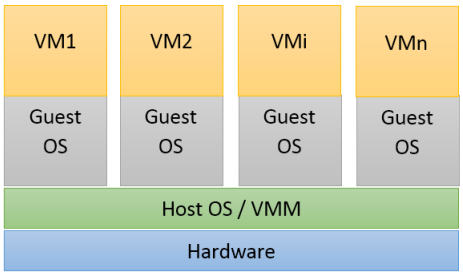

Figura 2.3 – Arquitetura do sistema de máquina virtual.

Máquina virtual permite aos usuários criar, copiar, salvar (*checkpoint*), ler e modificar, compartilhar, migrar e reverter o estado de execução da máquina com toda a facilidade de manipular um arquivo. Essa flexibilidade fornece um valor significativo para os usuários e administradores.

2.4 Migração de Máquinas Virtuais em Tempo Real

A computação em nuvem usa o conceito de virtualização. Ela permite que vários servidores virtualizados de forma isolada e com segurança possam rodar em um único servidor

físico. Muitas máquinas virtuais estão hospedadas no mesmo servidor físico para otimizar a utilização dos recursos deste servidor, reduzindo assim o custo de implantação de um Data Center. Isto possibilita também aumentar a segurança de servidores físicos em *Data Centers*. As máquinas virtuais são migradas de um servidor físico para outro, para alcançar a eficiência energética, balanceamento de carga e alta disponibilidade de servidor físico em Data Center na nuvem (LEELIPUSHPAM; SHARMILA, 2013).

Migração de Máquinas Virtuais em Tempo Real (*Live Virtual Machine Migration* ou *Live Migration*) é uma técnica onde todo o sistema operacional e seus aplicativos associados a ele são migrados, ou seja, são transferidos de uma máquina física origem para uma outra máquina física destino. As máquinas virtuais são migradas sem grandes perturbações nas aplicações em execução. Os benefícios da migração de máquinas virtuais incluem conservação de energia do servidor, balanceamento de carga física entre os servidores físicos e tolerância a falhas em caso de falha súbita (LEELIPUSHPAM; SHARMILA, 2013).

A LM está no nível de uma máquina inteira, ou seja, toda a máquina virtual é migrada, seu estado de memória, estado dos aplicativos e isso se aplica também ao *kernel* interno, migrando por exemplo, o bloco de controle TCP ativa no momento da migração. Em termos práticos, por exemplo, isso significa que se pode migrar um servidor de jogos *on-line* ou servidor de *streaming* de mídia sem a necessidade de que os clientes se desconectem (CLARK et al., 2005).

No entanto, existem parâmetros que afetam o desempenho de uma migração que devem ser considerados para a migração em tempo real de uma máquina virtual: tempo total de migração (o tempo decorrido desde o início da migração da primeira VM no *host* origem até o final da migração do último VM no *host* destino); *downtime* (o período de tempo em que a VM é totalmente suspensa durante a migração). Durante o tempo de inatividade, o estado de execução do CPU da VM e suas páginas no conjunto de trabalho são transferidos para o *host* de destino (SAGANA; GEETHA; SUGANTHE, 2013).

2.4.1 Objetivos de Migração

Diferentes técnicas para migração em tempo real podem ser utilizadas, e cada uma destas técnicas busca priorizar algumas características importantes que podem fazer com que o administrador do Data Center decida entre utilizar uma ou outra técnica. As diferentes categorias de técnicas de migração são descritas em Sagana (2013), que são:

2.4.1.1 Migração para Eficiência Energética

O consumo de energia em servidores se dá pela utilização do servidor e seus sistemas de refrigeração. Os servidores geralmente precisam de até 70% de seu consumo máximo de energia mesmo quando seu nível de utilização esteja baixo. Portanto, existe uma necessidade para técnicas de migração que conserve a energia de servidores otimizando a utilização dos seus recursos.

Como é apresentado pela figura 2.4 e a figura 2.5 um servidor físico que tem recursos sobrando, neste caso o servidor 1, é, então, um alvo para que seja utilizado seus recursos e, não havendo necessidade de alocação de novas VMs, logo é suspenso o servidor físico 2 para poupar energia.

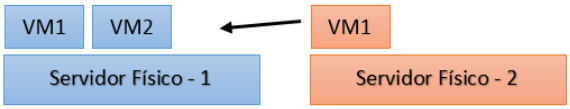

Figura 2.4 – Migração para Eficiência Energética Antes da Migração.

Figura 2.5 – Migração para Eficiência Energética Depois da Migração.

Isto ocorre quando a energia extra necessária para o servidor 1 executar a nova VM é inferior à energia que o servidor 2 consome.

2.4.1.2 Migração para Balanceamento de Carga

As técnicas de migração de balanceamento de carga visam distribuir carga entre os servidores físicos para melhorar a escalabilidade dos servidores em ambientes virtualizados. O balanceamento de carga auxilia a minimizar o consumo de recursos, implementação de *fail-over*, aumento da escalabilidade, evita gargalos e excesso de provisionamento de recursos, entre outros.

Um exemplo é apresentado na figura 2.6 e a figura 2.7 o servidor físico 2 está sobrecarregado e o servidor físico 3 está com menor utilização de recursos, para tanto Wood (2007) propõem uma técnica onde um motor de criação de perfil coleta informações sobre a utilização de recursos de todas as máquinas virtuais. Este motor *Hotspot* detecta a máquina virtual

que utiliza recursos e então o gerenciador de migração migra tais máquinas virtuais de servidores altamente utilizados para servidores menos utilizados. Assim, o gerenciador de migração emprega técnicas de provisionamento para determinar as necessidades de recursos de VMs sobrecarregadas e usa um algoritmo guloso para determinar uma sequência de movimentos ou *swaps* para migrar VMs sobrecarregadas para servidores de folga.

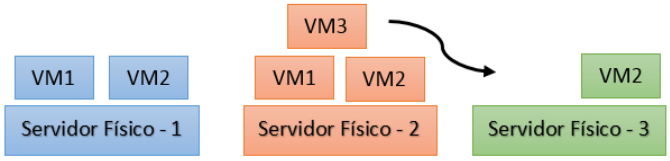

Figura 2.6 – Migração para Balaceamento de Carga Antes da Migração.

Figura 2.7 – Migração para Balaceamento de Carga Depois da Migração.

2.4.1.3 Migração para Tolerância a Falhas

A tolerância a falhas permite que as máquinas virtuais continuem o seu trabalho, mesmo se alguma parte do sistema falhar, figura 2.8 e a figura 2.9. Esta técnica migra da máquina virtual de um servidor físico para outro servidor físico baseado na previsão das ocorrências de falha. O benefício destas técnicas é a de melhorar a disponibilidade de servidores físicos e evitar a degradação do desempenho das aplicações.

Com esse objetivo as técnicas de migração migram as VMs de um servidor para outro para obter alta disponibilidade de servidores físicos. Elas preveem a falha e migram a máquina virtual para um outro servidor para impedir a degradação de desempenho das aplicações.

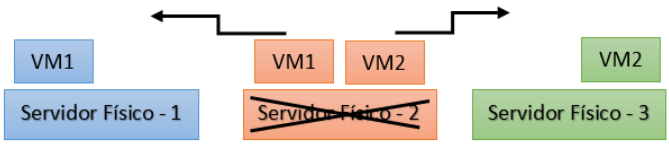

Figura 2.8 – Migração para Tolerância a Falhas Antes da Migração.

No trabalho de Fu (2009), por exemplo, é proposto um sistema em que um módulo de predição de falhas prevê falhas baseado no histórico de dados e informa ao módulo coordenador

Figura 2.9 – Migração para Tolerância a Falhas Depois da Migração.

de Máquinas Virtuais sobre o estado de saúde dos nós. O estado de condição de saúde mostra o tempo de vida do servidor físico. Ao receber as previsões de falha o módulo coordenador VM migra a máquina virtual a partir desse servidor para o servidor mais próximo que tem uma boa condição de saúde.

2.4.2 Técnicas para Controle de Migração

O objetivo de controlar a migração é reduzir o tempo de migração e o tempo de indisponibilidade durante todo o processo de migração. O tempo de migração é o tempo gasto desde o inicio até o término da migração, é o tempo total da migração. *Downtime* é o tempo que a máquina virtual que está sendo migrada ficou inativa devido à interrupção dos seus serviços. Desta maneira existem duas principais formas de controle de migração (LEELIPUSHPAM; SHARMILA, 2013):

- Técnica *Pre-Copy* e
- Técnica *Post-Copy*.

Na técnica *Pre-Copy* volumes de páginas de memória são copiadas da máquina de origem para máquina de destino enquanto os processos estão em estado de execução na máquina de origem. Os processos que estão sendo executados na máquina virtual podem solicitar alguma informação contida na página de memória a qual já foi transferida para o seu destino. Quando o processo tenta alterar algumas destas páginas de memória acaba ocasionando uma falha de página. Estas páginas são chamadas de *Dirtied Pages*, páginas sujas. Estas páginas sujas são transferidas de forma iterativa, onde toda a memória RAM é enviada na primeira iteração e nas iterações subsequentes são copiadas apenas as páginas sujas. Enfim, a máquina virtual de origem é suspensa e o estado do processador e dos processos e o restante de memórias são copiadas pra que então, na máquina destino, ela seja iniciada.

Já na técnica de migração *Post-Copy* inicialmente, o estado do processador e o mínimo de páginas de memória que são necessários para operar a máquina virtual no servidor de destino são copiados. Então, são ativadas a VM na máquina destino e concomitantemente as páginas

necessárias para o funcionamento da máquina virtual de destino são exigidas a partir da máquina virtual de origem.

Ambas as técnicas têm suas desvantagens. O tempo de migração é muito alto quando utilizada a técnica de *Pre-Copy*, devido a falhas nas páginas de memória que são acessadas no decorrer da migração exigindo um maior uso da rede. No entanto, o tempo de inatividade, ou seja, o *Downtime*, é alto no uso da técnica *Post-Copy*, porque enquanto o estado do processador é copiado, os processos e serviços ficam inativos até que a máquina virtual seja novamente ativada na máquina de destino.

2.4.3 Métricas e Termos na Migração de Máquinas Virtuais

Existem etapas da migração de máquinas virtuais em tempo real que podem gerar problemas no desempenho das aplicações e serviços que estão sendo executados.Pesquisas foram realizadas para avaliar os problemas na migração de máquinas virtuais em tempo real e foram sugeridas várias métricas de desempenho. Os resultados mostram que a sobrecarga de migração é aceitável, mas não pode ser desconsiderada, especialmente em sistemas onde a disponibilidade e capacidade de resposta são regidos por SLAs (*Service-Level Agreement*) rigorosos (KAPIL; PILLI; JOSHI, 2013).

As seguintes métricas são normalmente utilizadas para medir o desempenho de migração em tempo real, segundo Kapil (2013):

1. Tempo de Preparação (*Preparation Time*): Momento em que a migração inicia até que começa a transferência do estado da VM para a máquina de destino. A VM continua a executar e "sujar" a sua memória.

2. *Downtime*: Tempo de interrupção da máquina que está sendo migrada. Inclui a transferência de estado do processador.

3. Tempo de Retorno (*Resume Time*): Este é o tempo entre retomar a execução de VMs no destino e final da migração quando são eliminadas todas as dependências da origem.

4. Páginas Transferidas (*Pages Transferred*): Esta é a quantidade total de páginas de memória transferidas, incluindo duplicatas, em todos os períodos de tempo citados acima.

5. Tempo Total de Migração (*Total Migration Time*): Este é o tempo total de todos os tempos citados acima do início ao fim. O tempo total é importante porque afeta a liberação de

recursos em ambos os nós participantes, bem como dentro das VMs.

6. Degradação da Aplicação (*Application Degradation*): Esta é a extensão a que a migração retarda os aplicativos em execução dentro da VM.

2.5 Biblioteca Libvirt

A *libvirt* oferece uma API agnóstica de hipervisores para gerenciar, de forma segura, sistemas operacionais convidados que estejam sendo executados em um *host*. A biblioteca *libvirt* não é uma ferramenta em si, mas uma API para construir ferramentas a fim de gerenciar sistemas operacionais convidados. É construída sobre a ideia de abstração. A biblioteca oferece uma API comum para a funcionalidade comum implementada pelos hipervisores suportados. Ela foi originalmente criada como uma API de gerenciamento para Xen mas, desde então, foi ampliada para suportar diversos hipervisores (JONES, 2010).

2.5.1 Arquitetura básica

A *libvirt* existe como um conjunto de APIs projetadas para serem usadas como um aplicativo de gerenciamento. Na figura 2.10, à esquerda é apresentado um modelo tradicional de virtualização onde se tem uma máquina física, *Node*, um sistema operacional *host*, Linux *host*, e através de um hipervisor é possível virtualizar um sistema operacional convidado. E à direita é apresentado onde a *libvirt* se encaixa dentro do modelo de virtualização, se estabelecendo entre o sistema operacional *host* e o hipervisor. E o acesso a *libvirt* se dá por um aplicação de gerenciamento. Por meio de um mecanismo específico de hipervisores, a *libvirt* comunica-se com um hipervisor disponível para executar as solicitações da API.

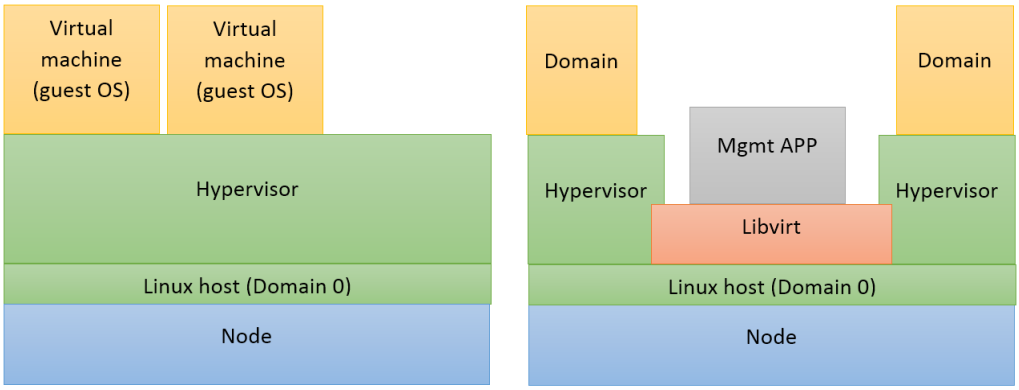

Figura 2.10 – Comparação e modelo de uso da *libvirt*.

A *libvirt* usa uma terminologia um pouco diferente. Esta terminologia é importante, pois esses termos são usados na nomenclatura da API. As duas diferenças fundamentais são que a *libvirt* chama o *host* físico de nó, enquanto que no sistema operacional convidado é chamado de domínio. A *libvirt* (e seu aplicativo) é executada no domínio do sistema operacional convidado Linux (domínio 0).

Inicialmente se tinha suporte apenas ao hipervisor Xen, mas com a atualização constante da biblioteca e o desenvolvimento da comunidade de aplicações este número aumentou. O suporte atual para hipervisores pela biblioteca *libvirt* são: KVM/QEMU; Xen; LXC Linux container system; OpenVZ; VirtualBox; VMware ESX e GSX; VMware Workstation e Player; Microsoft Hyper-V; IBM PowerVM; Parallels e Bhyve.

2.5.2 Meios de controle

Com a *libvirt*, existem dois meios de controle distintos. O primeiro é demonstrado na figura 2.10, em que o aplicativo de gerenciamento e os domínios existem no mesmo nó. Nesse caso, o aplicativo de gerenciamento trabalha por meio da biblioteca *libvirt* para controlar os domínios locais. O segundo meio de controle existe quando o aplicativo de gerenciamento e os domínios estão em nós separados. Aqui é necessária a comunicação remota. Este modo usa um *daemon* especial chamado *libvirtd* que é executado em nós remotos. Este *daemon* é iniciado automaticamente quando a *libvirt* é instalada em um novo nó e pode determinar, de forma automática, os hipervisores locais e configurar *drivers* para eles (isso será tratado em breve). O aplicativo de gerenciamento se comunica por meio da *libvirt* local com o *libvirtd* remoto através de um protocolo customizado. Para o QEMU, o protocolo termina no monitor QEMU. O QEMU inclui um console do monitor que permite inspecionar um sistema operacional convidado em execução, bem como controlar vários aspectos da máquina virtual (VM).

Para realizar o transporte dos dados e efetivar a comunicação com o host remoto a libvirt suporta uma variedade de protocolos de comunicação, sendo eles: TLS; Unix (Soquete do domínio Unix); SSH; EXT (Qualquer programa externo que pode fazer uma conexão com a máquina remota por meio externo ao âmbito da *libvirt*.); TCP; libssh2 (Transporte sobre o protocolo SSH usando libssh2 em vez do binário OpenSSH). O transporte padrão, se nenhuma outra for especificado, é o TLS (LIBVIRT, 2014).

2.5.3 Suporte a Hipervisor

Para suportar a extensibilidade para uma ampla gama de hipervisores, a *libvirt* implementa uma arquitetura baseada em *driver*, que permite que uma API comum atenda vários hipervisores subjacentes de maneira comum. Isso significa que determinadas funcionalidades especializadas de alguns hipervisores não são visíveis por meio da API. Além disso, alguns hipervisores podem não implementar todas as funções da API, que então são definidas como não suportadas no *driver* específico. Também o *libvirtd* oferece meios de acessar domínios locais a partir de aplicativos remotos.

Drivers são o alicerce fundamental para a funcionalidade *libvirt* para apoiar a capacidade de lidar com as chamadas de controladores de hipervisores específicos. Drivers são descobertos e registrados durante o processamento de conexão como parte da API "virInitialize". Cada *driver* tem uma API de registro que carrega as referências das funções específicas do *driver* para as APIs *libvirt* chamar. O que se segue é uma visão simplista do mecanismo de *drivers* para hipervisor. Considere a lista de *drivers* empilhados como uma série de módulos que podem ser conectados na arquitetura, dependendo de como a *libvirt* é configurada.

2.5.4 Aplicações que Utilizam a Libvirt

Existe uma grande variedade de aplicações que utilizam a API de gerenciamento *Libvirt*. Estas aplicações vão desde Cliente/Servidor, ferramentas de linha de comando, gerenciamento de configurações, infraestrutura como serviço, bibliotecas, aplicativos de *desktop*, até aplicações de WEB.

Para exemplificar, abaixo estão os principais programas que se utilizam desta biblioteca (LIBVIRT, 2014):

- *virt-install*: Fornece uma maneira de criar novas máquinas virtuais a partir de uma distribuição OS. Ele suporta provisionamento de imagens de CD locais e à rede através de NFS, HTTP e FTP.

- *virt-top*: Verifica CPU, memória, rede e utilização do disco de todas as máquinas virtuais rodando em um *host*.

- *virt-df*: Examina a utilização de cada sistema de arquivos em uma máquina virtual a partir do conforto da máquina *host*. Esta ferramenta espreita para os discos de hóspedes e

determina a quantidade de espaço usado. Ele pode lidar com sistemas de arquivos comuns do Linux e volumes LVM.

- *virt-manager*: Uma ferramenta de gerenciamento de desktop de uso geral, capaz de gerenciar máquinas virtuais em ambos os hipervisores localmente e acessados remotamente. Destina-se ao uso doméstico à gestão de 10 a 20 *hosts* e sua VMs.

- *virt-viewer*: Uma ferramenta leve para acessar o console gráfico associado a uma máquina virtual. Ele pode se conectar de forma segura para consoles remotos que suportam o protocolo VNC.

- *oVirt*: fornece a capacidade de gerenciar um grande número de máquinas virtuais em todo o data center de *hosts*. Ele se integra com FreeIPA para autenticação Kerberos, e no futuro, o gerenciamento de certificados.

- *virsh*: Um *shell* interativo e ferramenta programável de *scripts* em lote para as tarefas de gestão de desempenho para todos os domínios *Libvirt* gerenciados, redes e armazenamento. *Virsh* faz parte da distribuição do núcleo *Libvirt*.

2.6 Ontologias

Ontologia tem como base a filosofia onde este termo foi primeiramente desenvolvido. O termo vem do grego *ontos*, “ser”, “ente”; e *logos*, “saber”, “doutrina”, sendo assim, define-se por Aristóteles como sendo o estudo do “Ser enquanto Ser”. Guarino (1998) se refere à Ontologia, no sentido filosófico, como um sistema particular de categorias representando uma certa visão do mundo.

A computação, como sendo uma área de estudo relativamente nova quando comparada a outras áreas, absorveu o termo Ontologia e passou a utilizá-lo, no entanto com um sentido adequado a mesma. A Inteligência Artificial passou a utilizar o termo referindo-se a um artefato de engenharia, constituído por um vocabulário específico usado para descrever uma certa realidade, somando-se a um conjunto de pressupostos explícitos sobre o significado pretendido das palavras do vocabulário. Ela captura os conceitos e relações em determinado domínio e um conjunto de axiomas, que restringem a sua interpretação (GUARINO, 1998).

Atualmente, na área da computação, é amplamente citado e adotado a seguinte definição de ontologia segundo Borst (1997), a qual é consenso neste trabalho, que uma ontologia é

definida como uma especificação formal e explícita de uma conceitualização compartilhada, onde especificação formal quer dizer algo que é legível para os computadores, explícita são os conceitos explicitamente definidos, conceitualização representa um modelo abstrato de algum fenômeno do mundo real e compartilhada significa conhecimento consensual.

Nesse sentido, pelo fato de a ontologia ser uma forma de representar conhecimento consensual, ela acabada por ser comumente empregada como um método para a identificação de categorias, conceitos, relações e regras, para definir e conceituar o conhecimento em um domínio . É possível indexar dados a conceitos sem a participação humana através de técnicas de categorização automática utilizando como, por exemplo, estatística para encontrar padrões (HUANG, 2010).

A ontologia define as regras que regulam a combinação entre os termos e as relações. As relações entre os termos são criadas por especialistas, e os usuários formulam consultas usando os conceitos especificados. Uma ontologia define assim uma "linguagem" (conjunto de termos) que será utilizada para formular consultas (ALMEIDA; BAX, 2003). Desta maneira, segundo Librelotto (2008) a diferença entre ontologias e outros modelos de dados é que o seu principal objetivo é focado nos conceitos e seus relacionamentos, no qual a semântica destes relacionamentos é aplicada uniformemente.

Com relação a sua estrutura as ontologias nem sempre se apresentam da mesma maneira, mas existem características e componentes básicos comuns presentes em grande parte delas. Mesmo apresentando propriedades distintas, é possível identificar tipos bem definidos (ALMEIDA; BAX, 2003). Assim, existem os componentes básicos de uma ontologia que são segundo Knublauch (2004):

- Classes: organizadas como uma taxonomia e são as unidades básicas da ontologia;
- Relações: representam o tipo de interação entre os conceitos de um domínio;
- Axiomas: usados para modelar sentenças sempre verdadeiras;
- Instâncias: utilizadas para representar elementos específicos, ou seja, os próprios dados.

Para Guarino (1998) as ontologias ainda podem ser classificadas de acordo com o nível de dependência em relação a uma determinada tarefa ou ponto de vista, conforme mostrado na figura 2.11, na qual setas representam relações de especialização.

Sob esta perspectiva, as ontologias podem ser classificadas da seguinte maneira (STUDER; BENJAMINS; FENSEL, 1998):

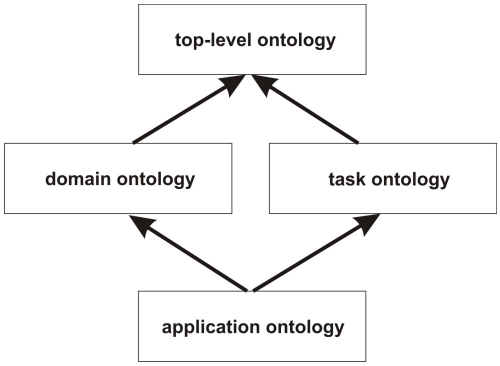

Figura 2.11 – Classificação da Ontologia.

- Domínio: utilizadas para representar o conhecimento válido para um tipo particular de domínio, como mecânica, medicina, biologia, etc. Expressam o vocabulário relativo de um domínio específico, descrevendo situações reais deste domínio;

- Genéricas: são similares às ontologias de domínio, porém os conceitos definidos neste tipo de ontologia são considerados em diversas áreas. Estes conceitos são tipicamente gerais como espaço, tempo, etc, os quais são independentes de um problema particular ou domínio. Conceitos em ontologias de domínio são frequentemente definidos como especializações de conceitos de ontologias genéricas;

- Aplicação: compreende os conceitos necessários para modelar o conhecimento requerido por uma aplicação específica. Esses conceitos correspondem, frequentemente, aos papéis desempenhados por entidades do domínio enquanto executam determinada atividade;

- Representação: este tipo de ontologia não se compromete com nenhum domínio em particular e determinam entidades representacionais sem especificar o que deve ser representado. Ontologias de domínio e genéricas descrevem seus domínios a partir de primitivas fornecidas pelas ontologias de representação.

Então, é necessário destacar que o processo de construção de ontologias não é uma tarefa trivial, tendo em vista que para a definição da mesma é necessário um conhecimento especializado de forma a não haver qualquer tipo de ambiguidade e contestações quanto a sua validade. Para tanto, como ainda não há uma ontologia que trate de um ambiente de migração de maquinas virtuais, é de grande importância neste trabalho o desenvolvimento da mesma onde é exemplificado no capítulo 4.

2.6.1 Linguagens de representação

As ontologias podem ser descritas de diversas maneiras. No entanto, para representar os conceitos e seus relacionamentos sobre determinado domínio do conhecimento, é necessário o uso de uma linguagem de representação comumente conhecida. Portanto, para a construção de ontologias é fundamental o uso de uma linguagem com semântica bem definida e expressiva para definir e criar relacionamentos entre os objetos. A linguagem OWL é uma das linguagens mais utilizadas para essa finalidade, sendo recomendada pela W3C (*World Wide Web Consortium*) (MARAN et al., 2013).

Para Web Semântica, as estruturas conceituais que definem uma ontologia fornecem a chave para que os dados possam ser processáveis por máquina. Assim a Web Semântica, tal qual as ontologias, tem como base o XML para definir esquemas de marcação personalizadas e juntamente com uma abordagem flexível do RDF para representação de seus dados (MCGUINNESS; HARMELEN, 2014). Desta forma, vale exemplificar os conceitos relacionados a XML, RDF e por fim OWL.

2.6.1.1 Representação de Dados com XML

A metalinguagem XML (*Extensible Markup Language*) descreve uma classe de objetos de dados chamados documentos XML e parcialmente descreve o comportamento de programas de computadores que os processam. Estes documentos XML são compostos de unidades de armazenamento chamadas entidades, que contêm dados parseados ou não parseados. Os Dados parseados são compostos de caracteres, alguns dos quais formam dados de caracteres, e alguns dos quais em forma de marcação. A marcação codifica uma descrição do esquema de armazenamento do documento e estrutura lógica. XML fornece um mecanismo para impor restrições e estrutura lógica sobre o layout de armazenamento (BRAY et al., 2000).

Devido aos marcadores do XML indicarem o conteúdo e a estrutura dos dados, é possível construir documentos legíveis por seres humanos e que podem ser facilmente processados por máquinas. Entre as características do XML, é relevante mencionar a sua independência dos dados e a separação entre conteúdo e apresentação. Também, é permitido criar *tags* personalizadas de forma que as informações podem ser organizadas pelo próprio autor do documento. Além disso, é baseado em texto, permitindo que ele seja editado ou criado com uma ferramenta simples.

2.6.1.2 Metadados com RDF

O RDF (*Resource Description Framework*) é uma estrutura para representar as informações na Web. Ele é uma base para o processamento de metadados fornecendo interoperabilidade entre aplicações que trocam informações compreensíveis por máquinas na Web. RDF traz facilidades que permitem o processamento automatizado de recursos da Web que podem ser usados em uma variedade de áreas de aplicação, como por exemplo: na descoberta de recursos para fornecer melhores capacidades do motor de busca; na catalogação para descrever o conteúdo e relações de conteúdo disponível em um determinado site, página ou biblioteca digital, por agentes de software inteligentes que irão, desta forma, facilitar o compartilhamento e troca de conhecimentos; na classificação de conteúdo; entre outras aplicações (LASSILA et al., 1998).

O principal objetivo do RDF é definir um mecanismo para descrever recursos que não fazem suposições sobre um domínio de aplicação particular, nem definem (a priori) a semântica de qualquer domínio de aplicação. A definição do mecanismo deve ser de domínio neutro, ou seja, não deve se ater a um domínio especifico ou ignorar algum domínio possível, sendo assim, o mecanismo deve ser adequado para descrever informações sobre qualquer domínio de aplicação.

A sintaxe apresentada pelo RDF utiliza-se de XML: um dos objetivos do RDF é torná-lo possível para especificar semântica para dados baseados em XML de forma padronizada e interoperável. Desta forma RDF e XML são complementares. Por RDF ser um modelo de metadados e apenas superficialmente resolve muitos dos problemas de codificação que o transporte e armazenamento de arquivos exige, ele necessita do apoio de XML. Também é importante compreender que esta sintaxe XML é apenas uma possível sintaxe para RDF e que formas alternativas para representar o mesmo modelo de dados RDF podem surgir.

O modelo RDF é um modelo para representar propriedades nomeadas e valores de propriedade. O modelo baseia-se em princípios bem estabelecidos de várias comunidades de representação de dados. O modelo de dados básico consiste em três tipos de objetos:

- Recursos: Todas as coisas que são descritas por expressões RDF são chamados de recursos. Um recurso pode ser uma página web inteira; como o documento HTML “http://www.w3.org/Overview.html”, por exemplo. Os recursos são sempre nomeados por URIs.

- Propriedades: A propriedade é um aspecto específico, característica, atributo ou relação

usada para descrever um recurso.

- Declarações: O conjunto de um recurso específico, juntamente com uma propriedade nomeada mais o valor dessa propriedade para esse recurso é considerado uma declaração RDF. Essas três partes individuais de uma declaração são chamadas, respectivamente, o sujeito, o predicado e o objeto.

2.6.1.3 Ontologias com OWL

A *Web Ontology Language* (OWL) é uma linguagem RDF desenvolvido pelo W3C para a definição de classes e propriedades, e também para permitir raciocínio mais poderoso e inferências sobre os relacionamentos. OWL foi construída como uma extensão para RDFS, um esquema de vocabulário mais simples, e é baseado em uma série de trabalhos anteriores no desenvolvimento de linguagens de ontologias. OWL é o padrão W3C atual para a definição de esquemas na Web Semântica, sendo que ferramentas e o suporte para API's OWL estão se expandindo rapidamente (SEGARAN et al., 2009).

OWL fornece três sublinguagens de complexidade e expressividade crescente, projetadas para o uso em comunidades específicas de implementadores e usuários, chamadas de: *OWL-Lite* (a mais simples); *OWL-DL* e *OWL-Full*. Cada sublinguagem são distintas em nível de formalidade exigida e oferecida na sua escrita, e também, pela liberdade de escrita dada ao usuário para a definição de ontologias (MCGUINNESS; HARMELEN, 2014).

A *OWL-Lite* suporta aqueles usuários que necessitam principalmente de uma hierarquia de classificação e restrições simples. Por exemplo, enquanto ele suporta restrições de cardinalidade, ela só permite valores de cardinalidade de 0 ou 1. Deve ser mais simples de fornecer suporte de ferramentas para *OWL-Lite* que seus parentes mais expressivos, e *OWL-Lite* oferece um caminho de migração rápida para tesauros e outras taxonomias.

Já a *OWL-DL* suporta aqueles usuários que querem a expressividade máxima, mantendo a integralidade computacional (todas as conclusões são garantidas para ser computável) e decidibilidade (todas as computações terminarão em tempo finito). *OWL-DL* inclui todas as construções da linguagem OWL, mas eles podem ser usados somente sob certas restrições (por exemplo, enquanto que uma classe pode ser uma subclasse de muitas classes, uma classe não pode ser uma instância de outra classe).

E por fim a *OWL-Full* é destinada a usuários que desejam máxima expressividade ea liberdade sintática do RDF sem garantias computacionais. Por exemplo, em *OWL-Full* uma

classe pode ser tratada simultaneamente como uma coleção de indivíduos e como um indivíduo em seu próprio direito. *OWL-Full* permite que uma ontologia aumente o significado do vocabulário pré-definido (RDF ou OWL).

Cada uma destas sublinguagens é uma extensão do seu antecessor mais simples, tanto no sentido do que pode ser legalmente expresso e o que pode ser validamente concluído. Isto significa que, por exemplo, uma ontologia válida em *OWL-Lite* também será válida em *OWL-DL*, mas nem toda ontologia válida em *OWL-DL* será válida em *OWL-Lite*.

2.6.2 Consultas com SPARQL

Para realizar as consultas sobre a ontologia é utilizada a linguagem SPARQL (DODDS, 2014), que é uma linguagem de consulta e protocolo de acesso de dados para a Web Semântica. SPARQL é definido em termos de modelo de dados RDF pela W3C e vai trabalhar para qualquer fonte de dados que podem ser mapeados em RDF. Ainda SPARQL pode ser usado para expressar consultas em diversas fontes de dados, sendo os dados armazenados nativamente como RDF ou visto como RDF via *middleware*. SPARQL contém recursos para consultar padrões grafos obrigatórios e opcionais, juntamente com suas conjunções e disjunções. SPARQL também suporta teste de valor extensível e consultas de restrições pelo grafos de RDF. Os resultados de consultas SPARQL pode ser conjuntos de resultados ou grafos RDF (SEABORNE; PRUD'HOMMEAUX, 2014).

SPARQL proporciona, assim, um conjunto completo de operações de consulta analíticas como JOIN, SORT, AGGREGATE para dados cujo esquema é intrinsecamente parte dos dados em vez de exigir uma definição de esquema separado. De modo específico, a consulta a seguir retorna os nomes e e-mails de todas as pessoas no conjunto de dados que estão relacionados à classe *Pessoa*:

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?nome ?email
WHERE {
  ?pessoa a foaf:Pessoa.
  ?pessoa foaf:nome ?nome.
  ?pessoa foaf:email ?email.
}
```

Outro exemplo de consulta SPARQL que responderia a seguinte pergunta dentro de uma ontologia que se representa o mapa político do Brasil: “Quais são todas as capitais do Brasil?”:

```
PREFIX abc: <http://exemplo.com/mapaPoliticoBrasil#>
SELECT ?capital ?pais
WHERE {
  ?x abc:nomecidade ?capital;
     abc:isCapitalOf ?y.
  ?y abc:nomepais ?pais;
     abc:isCountry abc:Brasil.
}
```

Nas consultas SPARQL, variáveis são representadas por “?”. Através do prefixo criado com o comando “PREFIX” é estabelecido para *abc* a URI da Ontologia onde é realizada a consulta. Nesta consulta são definidas as condições através da propriedade de Objeto *isCapitalOf* e *isCountry*, que respectivamente representam a relação da cidade com o país e a relação do país com a *String* Brasil. Ou seja, irá retornar as cidades que são capitais e estas capitais pertencem ao país “Brasil”.

2.6.3 Inferências com SWRL

A ontologia como representação de conhecimento, através de suas classes, as relações entre estas, e suas instâncias revelam o conhecimento sobre um determinado domínio. Contudo, esta não é a única capacidade de uma ontologia, classificar e agrupar conhecimento, através de motores de inferência para ontologias é possível utilizar ferramentas que raciocinam sobre as instâncias de uma ontologia ou mais esquemas de ontologia. Desta forma estas inferências podem ser usadas para responder a perguntas sobre o conhecimento explícito e implícito especificado por uma ontologia (FENSEL, 2005).

A linguagem OWL consegue relacionar conceitos entre si, como por exemplo, pode-se expressar que uma máquina virtual possui um disco rígido, no entanto esse conhecimento de que a máquina virtual possui um disco rígido é baseado em sua relação. Para criar, de fato, um conhecimento genérico sobre esta relação, de tal forma, que esta relação seja válida para todas as máquinas virtuais, é necessário uma expressividade mais ampla. Para esse propósito é que o SWRL surgiu.

SWRL é uma linguagem de regra que oferece extensões para regras mais complexas e eficientes que OWL. Propõe-se de tal forma que ele pode aproveitar as classes OWL e os indivíduos dentro de suas definições de regras (POLLOCK, 2009). Estas regras, segundo Horrocks (2014), têm a forma de uma implicação entre um antecedente e consequente. O significado pretendido pode ser lido como: sempre que as condições especificadas no antecedente pretendido forem atendidas, então as condições especificadas na consequente também devem estabelecer significado.

Assim, a linguagem SWRL pode ser utilizada juntamente com a OWL, aumentando o poder de expressividade sobre aquele conhecimento abordado na ontologia. A partir das regras criadas é possível inferir novos conhecimentos sobre a ontologia. Estas regras ao serem executadas irão afetar diretamente toda a estrutura da ontologia, pois quando uma regra encontra as condições especificadas ela gera um novo conhecimento e, a partir de então, esse conhecimento é aplicado à ontologia.

2.6.4 Consultas com SQWRL

SQWRL (*Semantic Query-enhanced Web Rule Language*) é construído sobre a linguagem de regra SWRL. Ele leva um antecedente de regra padrão SWRL e efetivamente, trata-o como uma especificação padrão para uma consulta. Ele substitui a regra consequente com uma especificação de recuperação (O'CONNOR; DAS, 2009).

Assim, ela permite a criação de consultas em uma ontologia de forma semelhante ao SQL. Estas consultas retornam as informações especificadas nas restrições e então esses resultados podem ser utilizados por um sistema qualquer, diferentemente da linguagem SWRL, ela não aplica o resultado à ontologia.

A linguagem SQWRL por ser baseada em SWRL, ela utiliza de recursos internos da SWRL como uma forma de extensão da linguagem, ou seja, utiliza suas bibliotecas e assim pode usar suas regras para fazer consultas em OWL. Usando destes recursos, é definido um conjunto de operadores que pode ser usado para construir especificações de recuperação de informação, assim, editores SWRL existentes podem ser utilizadas para gerar e editar consultas SQWRL (O'CONNOR; DAS, 2009).

O operador principal do SQWRL é *sqwrl:select*. É preciso um ou mais argumentos, que são normalmente variáveis usadas na especificação do padrão da consulta, e então é que se constrói uma tabela usando os argumentos como as colunas da tabela. Por exemplo, a consulta

a seguir recupera todas as pessoas em uma ontologia com uma idade conhecida que é de menos de 9, juntamente com suas idades:

$$Pessoa(?p) \wedge temIdade(?p, ?i) \wedge swrlb : lessThan(?i, 9) \rightarrow sqwrl : select(?p, ?i)$$

Esta consulta retornará aos pares de indivíduos e idades com uma linha para cada par. Resultados podem ser ordenados usando os métodos *orderBy* e *orderByDesCending*. Por exemplo, uma consulta para retornar uma lista de pessoas ordenadas por idade pode ser escrita:

$$Pessoa(?p) \wedge temIdade(?p, ?a) \rightarrow sqwrl : select(?p, ?a) \wedge sqwrl : orderBy(?a)$$

Existem diversos outros operadores para a construção de consultas SQWRL, como *sqwrl:orderBy*, *sqwrl:count*, *sqwrl:avg* entre outros. Logo, pela possibilidade de utilizar os recursos do SWRL, o SQWRL torna-se uma poderosa linguagem para busca de informações em OWL, possibilitando desta forma maior expressividade semântica em OWL.

2.6.5 Bibliotecas de Programação para Ontologias

Para que seja possível desenvolver aplicações de computador que se utilizem de ontologias, se faz necessário utilizar-se de interpretadores que realizam a tarefa de mediar os dados entre a ontologia e a aplicação que irá usufruir estas informações. Nesse sentido, a W3C mantém uma lista de bibliotecas[1], ou seja, APIs entre outras ferramentas para trabalhar com ou através de ontologias. Nesse guia, somente para a linguagem de programação Java, existem pelo menos, 45 bibliotecas. Para este trabalho não cabe discussão sobre todas elas, mas a seguir são destacadas algumas proeminentes da lista.

A biblioteca OWL API[2] é uma ferramenta para criar, manipular e serializar ontologias OWL para linguagem de programação Java. A biblioteca disponibiliza um conjunto de classes e métodos para manipulação de ontologias, deste a criação de uma ontologia até a leitura de um arquivo OWL existente. É possível trabalhar com os indivíduos, deletar, adicionar e alterar restrições nas propriedades de dados e objetos. A ferramenta também realiza relações com os tipos de dados da ontologia. Indivíduos podem ser relacionados entre si através de procedimentos especiais na OWL API.

[1] http://www.w3.org/2001/sw/wiki/Tools
[2] http://owlapi.sourceforge.net/index.html

Corese[3] é uma ferramenta para trabalhar com grafos em RDF. Ela fornece RDF, RDFS, consultas e atualizações com SPARQL, regras e inferências SPARQL. Possui como objetivo estruturar as ontologias em grafos e realizar pesquisas. Estas pesquisas são implementadas através de um conjunto de consultas em SPARQL. Corese aceita qualquer consulta, mas também possui funções especiais que adicionam mais funcionalidades a SPARQL, como buscas semânticas aproximadas.

Jastor[4] é um gerador de código Java de código aberto que emite Java *Beans* a partir de uma ontologias no formato OWL permitindo de forma prática o acesso seguro a modelos e eventos em RDF armazenada em um modelo de *Jena Semantic Web Framework*. Com esse gerador é possível gerar automaticamente interfaces, implementações e outros códigos referentes às classes, propriedades e relacionamentos de uma ontologia.

Jena[5] é um projeto de código aberto desenvolvido nos laboratórios da HP. Na manipulação de ontologias Jena trabalha com as linguagens RDF e OWL. A ontologia pode ser instanciada em memória ou persistir em bancos de dados relacionais como MySQL, Oracle e PostgreSQL. Esta ferramenta foi criada para facilitar o desenvolvimento de aplicativos que usam linguagens e modelos de informações da web semântica.

[3] http://wimmics.inria.fr/corese
[4] http://jastor.sourceforge.net/
[5] https://jena.apache.org/

3 TRABALHOS RELACIONADOS

Neste capítulo são brevemente apresentados alguns trabalhos relacionados ao tema de migração de máquinas virtuais. Nestes trabalhos, os autores buscaram exemplificar diferentes maneiras de definir qual máquina virtual deve ser migrada e para qual máquina física será o destino, buscando atender os objetivos de eficiência energética, balanceamento de carga e/ou tolerância a falhas. Sendo assim, utilizam-se diferentes técnicas e mecanismos para trabalhar com migração de VMs.

3.1 LVMCI

Segundo Zhang (2012), a maioria dos estudos existentes partem da definição de que a tomada de decisão de que máquinas virtuais alvo devem ser migradas para máquinas físicas adequadas em termos de utilização do recurso. No entanto, ações de migração podem degradar o desempenho do aplicativo migrado devido ao consumo de CPU e a largura de banda extra exigida durante a migração.

Nesse trabalho, os autores desenvolveram um sistema de seleção de VMs com menor custo de migração e menor interferência de desempenho de aplicativo, chamado LVMCI (*Live Virtual machine Migration with less Costs and application Interference*). O autor enfatiza que parte do desafio em migração de máquina virtual em tempo real é saber qual VM deve ser migrado e para que máquina física deve ser alocada esta VM.

O sistema LVMCI possui componentes que monitoram e coletam informações referentes a cada VM e também coletam parâmetros de cada máquina física. Estas informações são referentes ao uso de CPU e disco rígido e são coletadas a cada período de tempo definido pelo usuário. Estes dados, então, alimentam o componente relatório que transmite periodicamente estas estatísticas para o componente de plano de controle. Enfim, um ponto de migração é responsável por avisar o componente de tomada de decisão. Este último, formado pelos componentes de custo-consciente, que decide qual VM deve ser migrado, e interferência-consciente, que determina para qual máquina física será o destino.

O componente de custo-consciente é responsável por calcular o custo de migração de acordo com o um modelo de custo projetado. Ele mantém duas listas: uso intensivo de CPU das VMs e uso intensivo de entrada e saída das VMs. Cada lista é decrescente sobre custo de migração. Já o componente interferência-consciente é responsável por verificar a interferência

de desempenho de acordo com o modelo de interferência proposto.

3.2 Estratégia Baseada em Predição

O trabalho de Li (2012) propõe uma estratégia de migração em tempo real para máquina virtual, que utiliza de algoritmo de previsão de desempenho. Essas predições são feitas de acordo com a utilização média da CPU, memória, rede, entrada e saída de dados, e largura de banda. Então a estratégia de migração desse trabalho utiliza-se da previsão de desempenho e faz uma série de decisões sobre a migração, como: se uma migração deve ser acionada, que máquina virtual deverá ser migrada, e onde é o *host* de destino da máquina virtual.

Nesse trabalho, é utilizada a informação de monitores de *host* como uma fonte de dados referente aos recursos que estão sendo utilizados pelas máquinas virtuais e informações sobre a máquina *host*. Então o algoritmo de predição de desempenho OSVD (*Optimal Singular Value Decomposition*) é aplicado para prever um valor desempenho futuro, e também são analisadas as tendências de desempenho, para que, com análise global de várias informações de desempenho, a previsão do comportamento seja capaz de reduzir a migração dinâmica desnecessária, e melhorar a estabilidade do sistema.

O método OSVD utilizado é baseado em SVD (*Singular Value Decomposition*), onde, em álgebra linear, a decomposição em valores singulares (SVD) é uma fatoração de uma matriz real ou complexa, promovendo a diagonalização de matrizes ortogonais, com muitas aplicações úteis em processamento de sinais e estatísticas. Assim, para a previsão de desempenho em migração de máquina virtual, SVD pode ser visto como uma técnica para derivar um conjunto de informações de carga não correlacionadas.

3.3 Modelo Otimizador de Energia

No trabalho de Wei (2011) é apresentado um método abrangente que combina o modelo de migração consciente de energia e o modelo de despacho de carga. Com base na análise do processo de *pre-copy* durante a migração e o conhecimento que a energia consumida é linearmente proporcional ao volume de dados copiados, eles puderam estabelecer o modelo de migração consciente de energia que restringe o desempenho exigido determinando, desta forma, o melhor candidato VM à migração.

O *framework* é composto pelo Modelo de Despacho de Carga, que é responsável por car-

regar as máquinas físicas com trabalhos até encontrar um ponto ideal de utilização dos recursos, e o Modelo de Co-Migração, que é responsável pela seleção da VM. Esta precisa ser migrada da máquina física sobrecarregada, trabalhando em conjunto com o modelo anterior que enquanto isso carrega uma nova carga de trabalho adicional equilibrando a máquina sobrecarregada.

3.4 Virtual-Machines-MIB

É apresentado no trabalho de Hillbrecht (2012) a *Virtual-Machines-MIB*, a MIB (*Management Information Base*), dirigido à gestão de máquinas virtuais através de SNMP (*Simple Network Management Protocol*) – protocolo utilizado para monitorar o *status* dos equipamentos de rede e servidores. A *Virtual-Machines-MIB* visa definir uma interface padrão para o gerenciamento de máquinas virtuais, permitindo a gestão de vários monitores de máquinas virtuais, como Xen, KVM e VMWare. A *Virtual-Machines-MIB* permite executar operações de controle, como criar, apagar, reiniciar, ligar, fazer uma pausa e desligar máquinas virtuais. É também possível utilizar a solução proposta para modificar o nome de uma máquina virtual, a quantidade de RAM, o processador central virtual e de unidades de armazenamento virtuais. Estes agentes SNMP são baseados em agente NET-SNMP domínio público, que foi estendido para suportar Virtual-Machines-MIB usando API *libvirt*.

3.5 Considerações Sobre os Trabalhos Relacionados

Os artigos mencionados indicam que muitas pesquisas trazem à tona a importância da escolha correta de uma máquina virtual no momento de realizar uma migração. A escolha incorreta pode acarretar a má distribuição das VMs entre as máquinas *hosts*, ou seja, acaba desequilibrando a carga, por isso uma das características desejáveis com a utilização de *Live Migration* é manter o equilíbrio da carga. Os diferentes trabalhos mencionados utilizam algoritmos entre outros métodos para realizar migrações. Esse aspecto é relevante para este trabalho que tem como objetivo propor uma arquitetura para migração bem como definir uma ontologia de LM que represente um ambiente de conhecimento onde esteja inserido o uso de migração em tempo real, de forma que este conhecimento possa ser compartilhado à comunidade de áreas afins. Por isso, em relação aos trabalhos relacionados este trabalho busca contemplar de maneira mais ampla a migração de VMs, e não de forma específica e exclusivamente através de algoritmos.

Destaca-se em Kapil (2013) a importância na decisão de qual máquina será migrada e para qual *host*. Ele aborda a questão de disponibilidade de serviço como um dos desafios da migração de máquinas virtuais, e coloca que a disponibilidade de recursos pode ajudar a fazer a melhor decisão sobre quando migrar VM e como alocar recursos. Estes recursos disponíveis para *Live Migration* têm um impacto sobre o tempo de migração. Nesse sentido a ontologia poderia auxiliar na disponibilização dos recursos atualmente disponíveis.

Os trabalhos apresentados foram comparados ao trabalho proposto, nos aspectos de "Utilização de Sistema Semântico", "Utilização de Arquitetura Completa", "Conhecimento Compartilhado (Ontologia)" e "Sistema de Monitoramento". O resultado desta comparação é a Tabela 3.1, que mostra um comparativo entre os trabalhos relacionados e a proposta desta dissertação.

Tabela 3.1 – Comparativo entre trabalhos relacionados

Trabalho	**Utilização de Sistema Semântico**	**Utilização de Arquitetura Completa**	**Conhecimento Compartilhado (Ontologia)**	**Sistema de Monitoramento**
(ZHANG et al., 2012)	–	X	–	X
(LI et al., 2012)	–	–	–	X
(WEI; LIN; KONG, 2011)	–	–	–	X
(HILLBRECHT; BONA, 2012)	–	X	–	X
Proposta Apresentada	X	X	X	X

Pode-se compreender, a partir da tabela acima, que a abrangência da proposta dessa dissertação é maior quando comparada aos trabalhos relacionados, no que diz respeito a arquiteturas para migração de máquinas virtuais com relação a Compartilhamento de Conhecimento e Utilização de Sistema Semântico, uma vez que ontologias permitem que o sistema possa assumir a semântica de um dado relacionamento e atuar sistematicamente através dela. Com relação a Arquitetura Completa é dito no sentido de que os trabalhos que não contemplam esse item, fixam exclusivamente sua solução na elaboração de um algoritmo para migração, ou exclusivamente em um métrica específica, entre outras coisas. Por isso, essa dissertação contempla de forma mais ampla o tema de migração de VMs. No que diz respeito a Conhecimento Compartilhado é referente ao uso da ontologia propriamente dita.

4 ONTOLOGIA PARA MIGRAÇÃO DE MÁQUINAS VIRTUAIS EM TEMPO REAL

Neste capítulo é apresentada a ontologia modelada para representar um ambiente virtualizado onde a migração em tempo real de máquinas virtuais está inserida. Esta ontologia foi denominada de Onto-LM (*Ontology for Live Migration*). Para o desenvolvimento desta ontologia optou-se pela utilização da linguagem *OWL-DL*, e para sua construção optou-se pela utilização da ferramenta *Protégé* (HORRIDGE et al., 2009). Esta ferramenta é comumente utilizada na construção de ontologias por possuir recursos de importação e exportação em diversos formatos, proporcionando o reuso, além disso possui *plugins* para visualização e manipulação de ontologias, que permitem, por exemplo, realizar raciocínios sobre a ontologia.

Na construção da Ontologia Onto-LM seguiu-se o processo denominado *Ontology Development 101* (NOY; MCGUINNESS et al., 2001). Esse guia é um processo tradicional formado por um conjunto de passos e dicas para a determinação de entidades e seus relacionamentos entre si. A figura 4.1 apresenta as entidades que representam o fluxo definido pelo método *Ontology Development 101*.

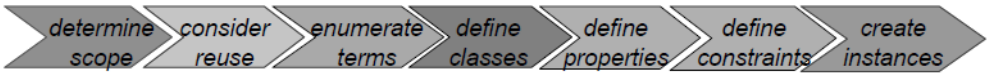

Figura 4.1 – Fluxo de desenvolvimento proposto pelo guia *Ontology Development 101* (NOY; MCGUINNESS et al., 2001)

Assim, o fluxo para a construção da ontologia se dá: a) pela determinação do escopo de atuação da ontologia, ou seja, definição do seu domínio; b) pela enumeração dos termos relacionados ao domínio, possíveis classes; c) pela definição das classes e suas hierarquias; d) pela determinação das propriedades das classes, sendo estas de dados, as quais caracterizam as entidades, e de objetos, cuja função é criar relações entre as classes; d) pela definição das restrições das propriedades e e) pela criação as instâncias da classes. Sendo que os itens listados acima podem considerar o reuso, ou seja, agregar outras ontologias que tenham uso no contexto pretendido.

A ontologia desenvolvida neste trabalho faz parte da camada de Infraestrutura de *Software* da Computação em Nuvem e alocado na subcamada Recursos Computacionais. No trabalho de Youseff (2008), para uma compreensão completa do campo da computação em nuvem, e uma adoção mais rápida por parte da comunidade científica, foi proposta uma ontologia desta

área que demonstra uma dissecção da nuvem em cinco camadas principais: *Cloud Application* (ex: SaaS – *Software as a Service* (*Software* como Serviço)); *Cloud Software Environment* (ex: PaaS – *Platform as a Service* (Plataforma como Serviço)); *Cloud Software Infrastructure*, onde estão: *Computational Resources* (ex: IaaS – *Infrastructure as a Service* (Infraestrutura como Serviço), *Storage* (ex: DaaS - *Data-Storage as a Service* (Banco de Dados como Serviço)), Communications (ex: CaaS - *Communication as a Service* (Comunicação como Serviço))); *Software Kernel* e *Firmware/Hardware* (ex: HaaS - *Hardware as a Service* (*Hardware* como Serviço)).

Especificamente a terceira camada (Infraestrutura de *Software* da Computação em Nuvem), ainda segundo o autor, fornece recursos fundamentais para outras camadas de nível superior, que por sua vez podem ser utilizados para a construção de novos ambientes de *software* em nuvem ou aplicações em nuvem. É por esse motivo que esta ontologia se encaixa nesta camada. E, ainda, na subcamada Recursos Computacionais (*Computational Resources*), uma vez que as máquinas virtuais (VMs) são a forma mais comum para o fornecimento de recursos computacionais para usuários da nuvem nessa camada. Assim, a ontologia aqui desenvolvida parte do conceito de *Cloud*, desce um nível de hierarquia em IaaS e inicia sua construção a partir de *Computing Resources* (Recursos Computacionais), baseado na Ontologia para Nuvem IaaS de Han (2010). A seguir, enfim, são exemplificadas as etapas de construção da ontologia.

4.1 Definição do Domínio

Dentro do contexto relacionado neste trabalho e exemplificado até então, a ontologia tem como objetivo representar um ambiente virtualizado onde a migração de máquinas virtuais em tempo real esteja inserido. Esta representação do ambiente visa relacionar os elementos que envolvem uma migração em tempo real, ou seja, as máquinas virtuais e suas características, as máquinas físicas e suas características e a relação desses elementos entre si. A ontologia também busca identificar potenciais máquinas para migração, uma vez que o processo de definição de alvos para migração e alvos destinos é uma etapa importante na migração.

Após a definição do domínio de atuação da ontologia, foi considerado o reuso de ontologias com a ideia de aproveitar os conhecimentos já pré-estabelecidos, no entanto não foram encontradas na revisão bibliográfica ontologias que abrangessem o contexto pretendido, ambiente virtual para migração de máquinas virtuais. Desta forma, optou-se pela criação de representação desse contexto específico de migração de máquinas virtuais.

Assim, para a criação de uma representação de contexto especifico foi conduzido uma busca por temos relacionados ao contexto pretendido. Essa busca se deu por artigos relacionados à migração de máquinas virtuais em tempo real. Nesses trabalhos, que deram embasamento teórico e que constam nas referências da seção 2, obteve-se termos e conhecimento sobre o ambiente, que, então, pode-se utilizar na elaboração de um ontologia que se engloba esse domínio.

A próxima etapa na definição da ontologia proposta foi elencar os principais termos do domínio e representá-los na ontologia através de classes, propriedades e indivíduos. Esse processo é descrito a seguir.

4.2 Termos e Classes

Neste trabalho, o domínio é o que envolve o ambiente de migração de máquinas virtuais, ou seja, os conceitos que envolvem esse ambiente. Então foram definidas as principais classes da ontologia, também chamadas de superclasses, pois definem as principais áreas de contexto que foram sendo refinadas durante a modelagem da ontologia. Elas representam elementos de contexto que podem influenciar o ambiente de migração.

Nesse sentido, os termos principais que foram elencados para delimitar o escopo da ontologia são: *Machine*, *Virtual Machine* (LEELIPUSHPAM; SHARMILA, 2013), *Physical Server* (LEELIPUSHPAM; SHARMILA, 2013), *Operating Sistem*, *Networking*, *Software*, *Hardware*, *PhysicalTarget*, *VirtualTarget*. Esses termos representam o ambiente computacional. Este trabalho é um passo inicial nesse domínio, visto que ainda não existem ontologias que abranjam esse escopo, por isso, os termos definidos têm o objetivo de exemplificar o ambiente de migração somente, no entanto, isso não impede de ser expandido futuramente.

Em um ambiente de virtualização temos máquinas virtuais e máquinas físicas, que podemos chamar de servidores. Tanto as máquinas físicas quanto as virtuais possuem características comuns como: endereço de rede, memória *ram*, disco rígido, processador e programas sendo executados. Nesse ambiente as propriedades destas características iram variar com o tempo e essa variação irá determinar as máquinas que serão migradas e as máquinas físicas de destino. Assim, as classes definidas basicamente são os mesmos termos descritos anteriormente, adaptados de Leelipushpam (2013): *Machine*, *VirtualMachine*, *PhysicalServer*, *OperatingSystem*, *Networking*, *Software*, *Hardware*, *Target*, *PhysicalTarget*, *VirtualTarget*.

Dentre estas classes é importante destacar que nem todas possuem instâncias, visto que foram definidas como classes abstratas e serão utilizadas para organização da hierarquia on-

tológica, e assim nenhum indivíduo poderá pertencer somente a elas. Para dar um exemplo desse tipo temos a superclasse *Target* e a superclasse *Machine*. Na superclasse *Target* qualquer máquina alvo devera ser classificada como *PhysicalTarget* ou *VirtualTarget*, ou seja, a máquina obrigatoriamente será um alvo independente da sua classificação física ou virtual. Na superclasse *Machine* as máquinas são classificas como *PhysicalMachine* ou *VirtualMachine*.

Já as classes concretas, que são as que possuem uma ou mais instâncias, reúnem todos os indivíduos que possam ser classificados nesse contexto que abrange a superclasse. Então as demais subclasses deste trabalho são concretas e, assim, a partir da definição de superclasses e subclasses é criado um sistema hierárquico na ontologia.

Até então, com estas classes é possível concluir que: existem máquinas que podem ser físicas ou virtuais, e estas máquinas podem estar relacionadas com uma rede, com um sistema operacional, com um hardware e um software, e estas podem ser alvos para serem migrados ou alvos destino.

A seguir é descrito o sistema hierárquico da ontologia definida neste trabalho, seguido de suas propriedades. As propriedades em uma ontologia podem ser de dois tipos: propriedades de dados, as quais são responsáveis por caracterizar as instâncias de cada classe; e propriedades de objeto que definem os relacionamentos existentes entre os indivíduos de cada classe. Na hierarquia da ontologia as subclasses herdam automaticamente as propriedades definidas na sua classe mãe.

4.3 Hierarquia

O escopo da ontologia é especialista, desta forma o conhecimento abrangido é, também, especializado dentro de um domínio, este o de migração em tempo real. Por esse motivo a hierarquia apresenta poucas classes e subclasses, como mostra a figura 4.2. Mas nada impede que esta hierarquia seja expandida.

Toda ontologia parte de uma classe principal, e geralmente se usa a classe *Thing* para representar esse início do conhecimento, ou seja, a ontologia representa alguma “coisa” independente do que seja essa “coisa”. Dessa forma a ontologia parte de *Thing* e, assim, o conhecimento passa a ser estabelecido e é expandido com o delineamento das ideias e do que se deseja representar. Na hierarquia acima, apresentadas na figura 4.2 as subclasses *PhysicalServer* e *VirtualMachine* são filhas de *Machine*. Da mesma forma *PhysicalTarget* e *VirtualTarget* são filhas de *Target*. E as demais classes estão no mesmo nível hierárquico, no entanto isso não impede

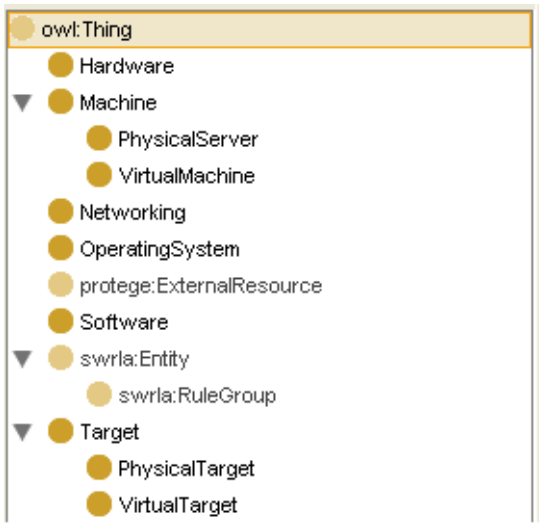

Figura 4.2 – Hierarquia das Classes da ontologia Onto-LM.

que, por exemplo, *Machine* tenha um relacionamento com *Hardware*.

4.4 Propriedades de Dados

As propriedades de Tipo de Dados são informações que dão forma às classes, isto é, as características utilizadas para descrevê-las e diferenciar cada indivíduo pertencente a esta classe. Por exemplo, a figura 4.3 mostra algumas das propriedades deste trabalho. A propriedade *disk* armazena a informação referente à quantidade de espaço de um disco rígido, e esta propriedade está relacionada com a classe Hardware, ou seja, toda classe Hardware possui como característica o armazenamento da informação do espaço em disco.

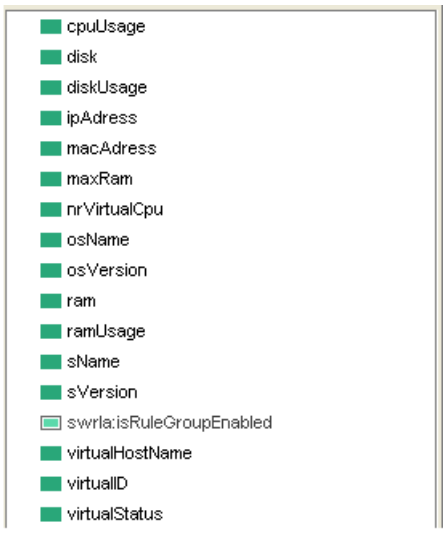

Figura 4.3 – Propriedades de Tipos de Dados.

Para representar o ambiente de migração foram criadas então algumas propriedades comuns ao mesmo, como, *ipAddress* e *macAddress* para identificação das máquinas físicas e virtuais dentro de uma rede, também para permitir a comunicação e possibilitar a migração. Outras características comuns a uma máquina também foram incorporadas, como, *ram* para armazenar o tamanho da memória disponível. Algumas características são distintas, tais como, *virtualHostName*, *virtualID*, *virtualStatus* e *nrVirtualCpu* são propriedades que pertencem às máquinas virtuais, pois não cabem às máquinas físicas.

No entanto, também existem propriedades comuns às VMs e às máquinas físicas: *cpuUsage*, *diskUsage*, *ramUsage* são propriedades que pertencem à classe *Machine*, ou seja, elas são herdadas pelas suas subclasses *VirtualMachine* e *PhysicalMachine*. Estas propriedades permitem analisar o uso destes recursos computacionais, ou seja, quanto uma máquina está usando de memória, qual o percentual do uso do processador, e qual a porcentagem do uso do disco rígido. Desta forma, pode-se ter o mínimo de informação disponível para verificar a situação atual do ambiente e decidir se é necessário realizar alguma migração e que máquinas irão migrar e para que destino.

Na Tabela 4.1 estão apresentadas as classes e suas propriedades de dados sendo elas: o nome da propriedade, os tipos de dados e seus valores representativos.

Tabela 4.1 – Propriedade de Dados da Ontologia

Classe	**Propriedade**	**Tipo de Dados**	**Valores**
Machine	cpuUsage	inteiro	0–100
	diskUsage	inteiro	0–100
	ramUsage	inteiro	0–100
Virtual Machine	virtualHostName	string	vmx, vmy
	virtualID	inteiro	0–n
	virtualStatus	string	Executando, Parado, Pausado
	nrVirtualCpu	inteiro	1–n
Hardware	disk	inteiro	Valores em Gigabytes
	ram	inteiro	Valores em Gigabytes
Software	sName	String	Nome de *Software*
	sVersion	String	Versão do Sofware
Networking	ipAddress	String	000.000.000.000
	macAddress	String	xx–xx–xx–xx–xx–xx
OperatingSystem	osName	String	Linux, Windows, Mac OS
	osVersion	String	Ubuntu 15.04, Windows 8, Leopard

Com a tabela acima, compreende-se os valores inteiros de 0-100 como valores que podem variar entre 0 e 100. Esses valores correspondem a porcentagem de uso da CPU, (*cpuUsage*), Disco Rígido (*diskUsage*) e Memória RAM (*ramUsage*). Sendo os valores *String*

correspondentes a valores literais que podem variar dependendo das informações que serão armazenadas.

4.5 Propriedades de Objeto

As propriedades Objetos de uma ontologia têm como função criar a relação entre as classes. Na figura 4.4 são apresentadas algumas propriedades de objeto deste trabalho. Para exemplificar o uso dessas relações na ontologia, considera-se uma máquina virtual que pertence à classe *VirtualMachine* ela possui um *hardware* que a caracteriza. Assim, através da propriedade *hasHardware* é relacionado a máquina virtual a uma classe que possui atributos que especificam o seus componentes de *hardware*.

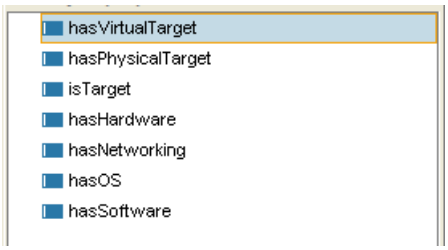

Figura 4.4 – Propriedades de Objetos.

Já a propriedade *hasOS* representa, neste trabalho, que uma *Machine* está relacionada com *OperatingSystem* no sentido de que ela possui/tem um relacionamento com *OperatingSystem*. E assim sucessivamente, *hasSoftware* significa que *Machine* possui/tem um relacionamento com *Software*.

Na Tabela 4.2 estão apresentadas as classes e suas propriedades de objeto sendo elas: o nome da propriedade, o relacionamento que esta classe possui relacionando na coluna seguinte para qual classe esse relacionamento acontece, e se existe alguma relação inversa.

Tabela 4.2 – Propriedade de Objetos da Ontologia

Classe	**Relacionamento**	**Classe**	**Relação Inversa**
Machine	hasNetworking	Networking	–
	hasHardware	Hardware	–
	hasOS	OperatingSystem	–
	hasSoftware	Software	–
	isTarget	Target	isa
PhysicalTarget	hasPhysicalServer	PhysicalServer	isa
VirtualTarget	hasVirtualTarget	VirtualMachine	isa

Com a tabela acima, compreende-se que a classe *Machine* possui diversos elementos.

Cabe ressaltar que a *Machine* é um *Target* assim como *Target* é uma *Machine*, ou seja, existe uma relação inversa entre estas classes. Da mesma maneira, ocorre com as classes *PhysicalTarget* e *PhysicalServer*, e *VirtualTarget* e *VirtualMachine*.

4.6 Restrições

As restrições limitam e descrevem o conjunto de valores possíveis que as propriedades podem assumir, como por exemplo uma *string*, um inteiro, ponto flutuante e entre outros. Também através da restrição é possível especificar a classe a que a instância pertence, aplicar cardinalidades, valores máximos e mínimos etc. Em OWL existem três principais tipos de restrições de propriedades: restrições de quantificador, restrições de cardinalidade e restrições de valor (HORRIDGE et al., 2004).

Para um determinado indivíduo, o quantificador efetivamente coloca restrições sobre as relações que o indivíduo participa. Ele faz isso, ou especificando que deve existir pelo menos um tipo de relacionamento, ou especificando o tipo de relações que podem existir (se existirem). Esses quantificadores são: *someValuesFrom* quando os indivíduos de uma classe devem possuir pelo menos um valor de um determinada classe, e *allValuesFrom* significa que essa propriedade sobre uma classe em particular tem uma restrição de alcance local associado a ele.

As restrições de cardinalidade são usadas para definir sobre o número de relacionamentos que um indivíduo pode participar de uma determinada propriedade. As cardinalidades podem ser definidas como *minCardinality* e *maxCardinality*. Nesse sentido, para o trabalho é aplicada uma restrição de cardinalidade. A restrição aplicada é de que toda máquina sendo física ou virtual ela possui obrigatoriamente um *hasHardware*, uma *hasNetworking*, um *hasOS*, e no mínimo um *hasSoftware*. Também uma disjunção é aplicada, a de que uma instância de máquina virtual não pode pertencer também a uma máquina física.

4.7 Instâncias

As instâncias são de fato as informações. A ontologia é alimentada através da criação de instâncias e nesse processo as informações vindas de um banco de dados, de um arquivo XML, entre outros meios, são instanciadas e a partir de então, passíveis de processamento.

No ambiente deste trabalho todas as classes devem ser instanciadas e abrangendo todas as informações necessárias para que seja possível a realização de consultas e inferências sobre

as informações, e para que, enfim, sejam tomadas decisões sobre o resultado destas consultas.

Características das instâncias:

- *Hardware*: disk (inteiro em GB), *ram* (inteiro em GB);
- *PhysicalServer*: ramUsage (inteiro em porcentagem), cpuUsage (inteiro em porcentagem), diskUsage (inteiro em porcentagem), hasNetworking (instância de *Networking*), hasOS (instância de *OperatingSystem*), hasHardware (instância de *Hardware*), hasSoftware (instância de *Software*);
- *VirtualMachine*: ramUsage (inteiro em porcentagem), cpuUsage (inteiro em porcentagem), diskUsage (inteiro em porcentagem), hasNetworking (instância de *Networking*), hasOS (instância de *OperatingSystem*), hasHardware (instância de *Hardware*), hasSoftware (instância de *Software*), maxRam (inteiro em GB), *virtualHostName* (*string*), virtualID (inteiro), virtualStatus (*string*), nrVirtualCpu (inteiro), valor de carga do servidor (inteiro);
- *Networking*: ipAddress (*string*), macAddress (*string*);
- *OperatingSystem*: osName (*string*), osVersion (*string*);
- *Software*: sName (*string*), sVersion (*string*);
- *PhysicalTarget*: hasPhysicalTarget (instâncias de *PhysicalServer*);
- *VirtualTarget*: hasVirtualTarget (instâncias de *VirtualMachine*).

4.8 Consulta

As consultas em uma ontologia são realizadas utilizando SQWRL, o qual fornece operações de forma análoga ao SQL para recuperar informações da ontologia (COP, 2012).

Através das consultas obtemos informações referentes às instâncias baseadas nas regras criadas para as consultas. Estas regras são executadas e retornarão os dados obtidos. Por exemplo, no que cabe a ontologia criada neste trabalho, pode-se obter a seguinte consulta para descobrir qual máquina virtual está utilizando mais que 70% da memória RAM:

$$VirtualMachine(?vm) \wedge ramUsage(?vm, ?ur) \wedge swrlb : greaterThan(?ur, 70) \rightarrow sqwrl : select(?vm, ?ur)$$

E para descobrir qual máquina física estaria disponível para receber a máquina virtual a ser migrada poderíamos realizar a seguinte pesquisa, para obter, por exemplo, qual máquina física está utilizando menos que 30% da memória RAM:

$$PhysicalServer(?fm) \wedge ramUsage(?fm, ?ur) \wedge swrlb : lessThan(?ur, 30) \rightarrow sqwrl : select(?fm, ?ur)$$

Também através de consultas podem ser realizadas inferências, ou seja, a descoberta de conhecimento, no entanto, elas não modificam a ontologia, somente inferem o conhecimento e retornam o que foi encontrado.

4.9 Inferências

Diferente das consultas, as inferências resultam em conhecimento e estes são aplicados à ontologia de forma que as instâncias afetadas pela inferência são reclassificadas ou desclassificadas de acordo com a regra inferida.

Foi criado uma classe chamada *MachineTarget*, onde o resultado das buscas citadas na seção anterior ao invés de retornar as informações colocadas na função *sqwrl:select* serão inferidas através das regras SWRL, e, então, as informações inferidas serão aplicadas, sendo assim, a ontologia é reclassificada.

Para exemplo, poderia se dizer que se uma máquina virtual com mais de 70% de uso de memória RAM pertence a classe *MachineTarget*, ou seja, esta máquina virtual passa a ser uma candidata a migração. Esta inferência poderia ser feita com a seguinte regra SWRL:

$$PhysicalServer(?vm) \wedge ramUsage(?vm, ?ur) \wedge swrlb : greaterThan(?ur, 70) \rightarrow PhysicalTarget(?vm)$$

Desta forma, quando aplicada a regra, o resultado é a vinculação automática de uma instância de uma *PhysicalServer* na classe *PhysicalTarget* sem a necessidade de que isso seja

definido manualmente, ou seja, com esta regra todas as máquinas que atenderem às características estipuladas pelo SWRL passarão a ser consideradas máquinas alvo. Esta mesma regra deve ser aplicada para o conjunto de máquinas virtuais para classificar quais VMs estão mais sobrecarregadas para que a VM que está sobrecarregando o servidor seja migrada.

Então, com as máquinas virtuais alvo determinadas pode ser aplicada uma nova regra, novamente sobre as máquinas alvos, para determinar dentre as máquinas já classificadas como candidatas para migração qual VM está mais sobrecarregada.

Um outro exemplo de inferência pode ser feita. É possível ser utilizado um índice de carga dos servidores. Esse índice pode ser calculado dividindo o uso de recursos utilizados pelas VMs no servidor pelo total de recursos que o servidor disponibiliza. Desta forma, poderia se ter um índice que seria aplicado a cada servidor, bem como a cada máquina virtual. Logo, a inferência poderia ser utilizada sobre esse índice:

$$PhysicalServer(?vm) \wedge hasIndice(?vm, ?i) \wedge swrlb:greaterThan(?i, valor\_do\_indice) \rightarrow PhysicalTarget(?vm)$$

Ou seja, ao invés de ficar presa a um único recurso, através do índice a migração aconteceria de forma mais adequada, pois a medida de sobrecarga faria mais sentido, uma vez que a memória esteja próxima de seu uso completo isso não significaria necessariamente um problema. Por isso, com a utilização de um índice teríamos uma medida mais justa.

4.10 Considerações do Capítulo

Este capítulo apresentou detalhadamente a ontologia criada para representar um ambiente virtualizado para migração de máquinas virtuais. Do mesmo modo, descreveu as classes, suas propriedades e restrições, e discutiu sobre consultas SQWRL e possível inferência através de regra SWRL.

A ontologia faz parte da metodologia proposta pelo trabalho e é de fundamental importância para a arquitetura de migração de máquinas virtuais, visto que os dados descritos na ontologia descrevem todos os conceitos envolvidos nesse tipo de ambiente. Essas informações serão utilizadas por sistemas de virtualização que implementem a arquitetura proposta, a qual é vista no capítulo seguinte.

5 ARQUITETURA PARA MIGRAÇÃO DE MÁQUINAS VIRTUAIS UTILIZANDO ONTOLOGIA

Este capítulo apresenta a arquitetura proposta para um sistema de migração de máquinas virtuais utilizando ontologias como ferramenta para mapear o ambiente de virtualização. Esse conhecimento sobre o ambiente, ao se apresentar mapeado através da ontologia, permitirá então a aplicação de consultas e inferências sobre esta ontologia a partir de módulos do sistema responsáveis pela detecção de possíveis máquinas físicas e virtuais sobrecarregadas. A aplicação de consultas é baseada em métricas definidas previamente para indicar as máquinas físicas e virtuais sobrecarregadas que, estas últimas, poderão ou não ser migradas de acordo com as métricas estabelecidas.

A arquitetura que é apresentada nesta dissertação tem como referência alguns trabalhos já realizados por outros autores como Zhang (2012), Wei (2011) e Ye (2012). Esses autores desenvolveram *frameworks* para migração de máquinas virtuais e apresentam a mesma base de arquitetura. As diferenças encontram-se nas tecnologias utilizadas e na lógica de composição de alguns módulos.

Basicamente, o sistema é composto por uma aplicação de monitoramento, *script* Monitor, configurado em cada servidor com a finalidade de interagir com a aplicação OntoMig. Esses *scripts* Monitores respondem à aplicação OntoMig suas requisições de informações referentes ao uso dos recursos dos servidores. A aplicação OntoMig faz as requisições aos *scripts* Monitores sobre os recursos que estão sendo utilizados pelos servidores e pelas máquinas virtuais, e então realiza a atualização constante das informações contidas na ontologia. Essas informações são atualizadas com os dados de todo o ambiente virtualizado vindo dos *scripts* Monitores, e, enfim, seriam processados pela aplicação OntoMig, que manipula ontologias através da leitura dos dados, da inserção de elementos e valores, da pesquisa ou inferência. Após o processamento das informações e da ontologia, essa mesma aplicação realiza a migração, caso necessário.

A figura 5.1 apresenta uma visão geral da arquitetura deste trabalho, onde os componentes e o fluxo das informações estão destacados respectivamente por formas geométricas e setas. Como é possível visualizar, as VMs que estão sob o Servidor Físico (PM) possuem um Monitor VM, responsável pela coleta de informações sobre os recursos da VM, sendo estas informações solicitadas por outro Monitor que se encontra no Servidor Físico, responsável pela coleta de dados sobre os recursos da máquina física e da solicitação para o Monitor VM. As informações

vão da VM para o Servidor Físico, e dele para a aplicação OntoMig (desenvolvida para este trabalho), a qual solicita periodicamente os dados sobre o uso de recursos. Esta aplicação processa os dados recebidos, e utilizando-se de APIs, a aplicação carrega a ontologia para a memória e então popula com tais informações já processadas. Uma vez populada a aplicação consulta a ontologia e verifica a necessidade de migração de máquina virtual.

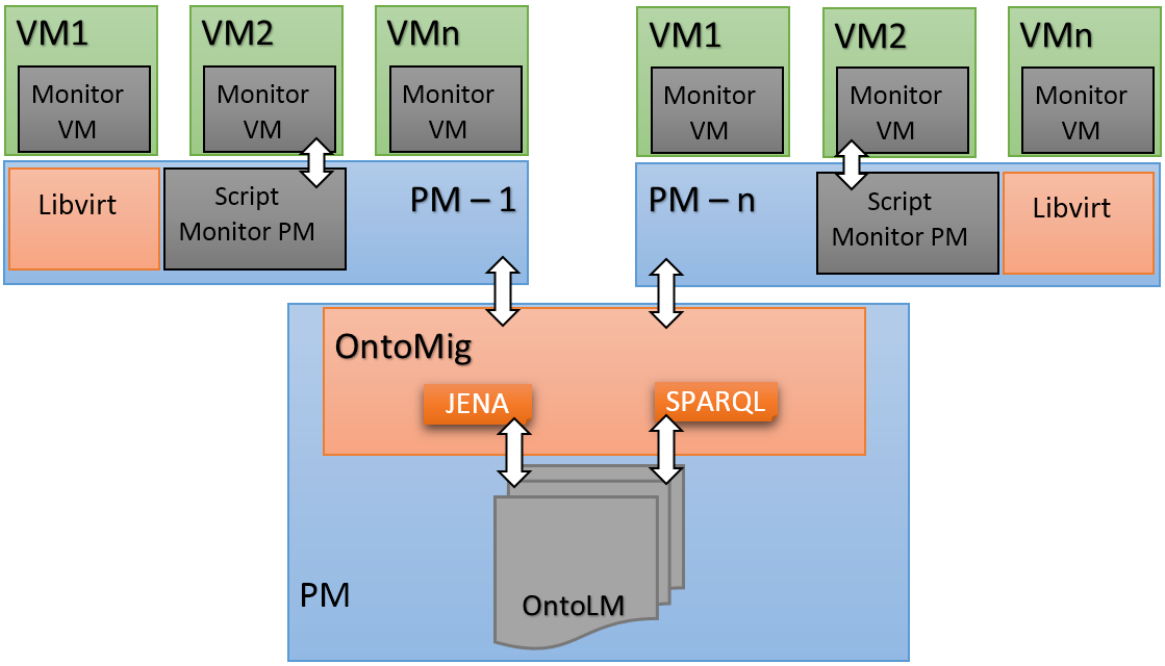

Figura 5.1 – Arquitetura do Sistema de Migração de Máquinas Virtuais Utilizando Ontologia.

A arquitetura possui 3 processos principais de funcionamento que são: monitorar a utilização dos recursos utilizados pelas VMs e Servidores Físicos; carregar e popular a ontologia; e verificar a necessidade de realização de migração de máquina virtual, havendo necessidade então é executada a migração.

Para o monitoramento da utilização dos recursos das VMs e dos servidores físicos é utilizado *Shell script*, o qual é configurado consecutivamente em cada máquina virtual e física. Esses *scripts* executam comandos que retornam informações referentes aos recursos que estão sendo utilizados pelas máquinas, como por exemplo as porcentagens de uso da CPU e memória RAM. Essas informações são requeridas pela aplicação principal, OntoMig.

De posse das informações advindas dos *scripts*, os próximos processos são carregar e popular a ontologia. Carregar a ontologia significa trazer o modelo OWL, que representa os conceitos estabelecidos nesta ontologia através de suas marcações OWL, para a memória. Estando o modelo na memória é possível processá-lo. Processá-lo entende-se por atribuir instâncias à ontologia, atribuir propriedades e valores, realizar pesquisas e inferências. Uma vez o

modelo carregado para a memória, então, é populado com informações referentes ao ambiente virtualizado que se constitui pelas máquinas virtuais e físicas.

O ato de popular a ontologia se dá pela ferramenta OntoMig. Ao popular a ontologia a ferramenta alimenta o modelo com informações sobre o ambiente computacional. A ontologia populada permite, a visualização do estado de utilização dos recursos computacionais naquele momento especifico. A partir disso, é possível obter informações necessárias para verificar a necessidade de migração, através desse mapeamento do ambiente.

Para verificar a necessidade de realização de migração de máquina virtual, última fase do sistema, é necessário, antes, estabelecer uma métrica para determinar quando é considerada pertinente a migração de máquinas virtuais. Essa métrica é estabelecida na seção 5.3. A métrica é calculada no mesmo instante em que é populada a ontologia, para cada máquina física e virtual. Desta maneira, após esse processo, além do estado atual do ambiente com relação aos recursos, se tem os valores de carga de cada máquina. Havendo esse índice de carga é possível delimitar parâmetros para realização de migrações.

Para exemplificação destes processos, delineando os detalhes de cada processo, são detalhadas a seguir cada módulo e seu respectivo funcionamento.

5.1 Monitoramento dos recursos

Primeiramente é necessário obter os dados do ambiente como um todo, ou seja, obter os recursos que estão sendo utilizados pelas máquinas físicas e virtuais. A figura 5.2 apresenta o fluxo que as principais atividades executam neste processo de monitoramento. Agentes Monitores neste trabalho é o *script* monitor. Esse monitor executa comandos que buscam dados referente a memória RAM, armazenamento e CPU. Sobre armazenamento, os comandos obtêm dados de todos os dispositivos montados no diretório */dev/* . Para a memória RAM os dados são referentes ao seu total, e quanto esta em uso, e a partir disto se faz os caculos de porcentagem de uso e de memória livre. E para CPU é utilizado arquivos do sistema operacional Linux que armazenam dados sobre as atividades realizadas pelo processador em determinado tempo.

Para obter essas informações é utilizado *Shell script* (COOPER, 2014) sobre as máquinas físicas e o acesso remoto para obter estas informações acontecem através de SSH (*Secure Shell*) (OPENSSH, 2014). mas outros métodos poderiam ser utilizados neste item. Tais como SMNP, que é um protocolo padrão para monitoramento e gerenciamento de redes. A sigla SNMP é um acrônimo para “Simple Network Management Protocol” ou “Protocolo Simples

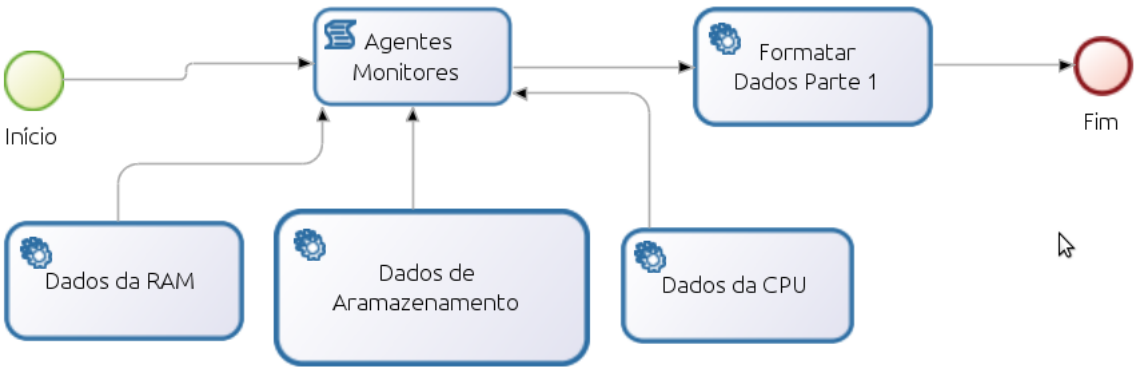

Figura 5.2 – Processo de monitoramento da arquitetura.

de gerenciamento de redes". Também ferramentas como *iostat tool* e *sar* ambas utilizadas por Zhang (2012) que fazem monitoramento de recursos, no entanto é optado neste trabalho a utilização de *Shell script* por ser uma ferramenta simples entretanto eficaz, para o desenvolvimento de *scripts*, e está pré-instalado nos sistemas operacionais Linux, evitando assim a instalação de novos recursos.

Esse *Script* Monitor escrito em *Shell script* utiliza-se de arquivos como */proc/cpuinfo* e */proc/stat* para descobrir informações sobre CPU onde através desses arquivos é possível verificar a quantidade de tempo que o processador gasta trabalhando. Ao fazer esta pesquisa em dois momentos diferentes é possível realizar um cálculo da diferença entre as duas medidas feitas e então se tem a porcentagem de uso de processamento. Também através de comandos como *free* e *df* para solicitar ao sistema operacional informações referentes, respectivamente, à memória e ao disco rígido. Sobre a memória são obtidas informações sobre: total de memória, uso de memória, espaço livre de memória, e as porcentagens de uso e espaço livre de memória. Também sobre os discos rígidos é requisitado ao sistema operacional informações sobre: tamanho total de disco, espaço utilizado e espaço livre, e porcentagem de uso. A figura 5.3 apresenta o *script* utilizado.

Nas linhas 4, 7 e 8 através do comando *free* é obtido do sistema operacional informações referentes ao uso de memória, e nas linhas 11 e 14 é calculada a porcentagem do uso de memória com os dados já obtidos nos comandos anteriores. É utilizado nesses comandos *grep*, um aplicativo para linha de comando de sistemas Linux que faz buscas no conteúdo dos arquivos procurando linhas que respeitem a expressão regular mencionada, e *awk*, também um aplicativo Linux em linha comando que habilita mais opções para processar textos e conteúdos em arqui-

```
1  #!/bin/bash
2
3  # MEMORIA == total memoria (OK)
4  MEM_TOTAL=$(free -m | grep Mem | awk '{ print $2 }')
5
6  # MEMORIA == memoria usada e livre em MB (OK)
7  MEM_USANDO=$(free -m | grep cache: | awk '{ print $3 }')
8  MEM_LIVRE=$(free -m | grep cache: | awk '{ print $4 }')
9
10 # MEMORIA == porcentagem do uso de memoria (OK)
11 MEM_USANDO_PORC=$(((($MEM_USANDO*100)/$MEM_TOTAL))
12
13 # MEMORIA == porcentagem do uso de memoria livre (OK)
14 MEM_LIVRE_PORC=$(((($MEM_LIVRE*100)/$MEM_TOTAL))
15
16 # DISCO == verifica uso de disco
17 DISCO_H_STRING=$(df -h | grep /dev/ | awk '{ printf("%s %s %s %s %s \n", $1, $2, $3, $4, $5) }')
18 DISCO_H_INTEIRO=$(df -h | grep /dev/ | awk '{ printf("%s %s %s %s %s \n", $1, $2, $3, $4, $5) }')
19
20 # PROCESSADOR == calculo da porcentagem de uso do processador (OK)
21 NUMCPUS=`grep ^proc /proc/cpuinfo | wc -l`;
22 FIRST=`cat /proc/stat | awk '/^cpu / {print $5}'`; sleep 1;
23 SECOND=`cat /proc/stat | awk '/^cpu / {print $5}'`;
24 USED=`echo 2 k 100 $SECOND $FIRST - $NUMCPUS / - p | dc`;
25
26 echo $MEM_TOTAL
27 echo $MEM_USANDO
28 echo $MEM_LIVRE
29 echo $MEM_USANDO_PORC
30 echo $MEM_LIVRE_PORC
31 echo $DISCO_H_STRING
32 echo $USED
```

Figura 5.3 – *Shell Script* de monitoramento.

vos. Nas linhas 17 e 18 são obtidas informações referentes ao uso de disco rígidos através do comando *df*, onde através de *flags* e utilizando *grep* e *awk*, é possível selecionar exatamente os campos que se precisa do retorno do comando.

Para obter informações sobre o processador, na linha 21 procura-se em */proc/cpuinfo* o número de processadores que a máquina tem, posteriormente na linha 22 no arquivo */proc/stat* é encontrado o tempo de atividade e inatividade do processador. Nesse arquivo encontram-se informações que identificam a quantidade de tempo que o CPU gastou realizando diferentes tipos de trabalho e, também, o tempo em que o processador ficou ocioso. Para descobrir a porcentagem de uso é necessário fazer a leitura em duas unidades de tempo diferentes, assim primeiro na variável *FIRST* é obtido a quantidade de centisegundos acumulados em que o processador passou fazendo nada, ou seja, inativo. Da mesma maneira, na variável *SECOND* é conseguido a mesma informação, no entanto, com intervalo de 1 segundo de diferença da variável FIRST. Enfim, na linha 24 é realizado o cálculo de porcentagem utilizando a calculadora de precisão arbitrária *dc* (*Desk Calculator*). No cálculo são definidas duas casas decimais de precisão, *2 k*, e então é feito o cálculo subtraindo as duas medidas para descobrir o tempo de inatividade naquele segundo e o complemento para 1 segundo será o quanto ele ficou trabalhando.

Por fim, as últimas linhas do *script* realizam a tarefa de formatar os dados em uma sequencia de saídas que seguem a sequência lógica de execução. Cada linha da saída é um dado, na primeira linda de saída se tem a memória total, seguido de: memória usada, memória livre, memoria usada em porcentagem, memória livre em porcentagem. Para dados sobre discos rígidos estão todos reunidos em uma única variável, ou seja, a saída da linha 31 é uma linha inteira com informações de todos os dispositivos montados em */dev/*. Estes dados sobre os discos não são executadas individualmente, em um único comando todos os discos são listados. E por fim, é dado como saída a linha 32 com a porcentagem de uso do processador.

Vale ressaltar que este método utilizado para o monitoramento dos recursos do ambiente virtual não é genérico, uma vez que estes comandos escritos em *Shell script* não podem ser utilizados em máquinas virtuais com sistema operacional Windows como convidado. Esta é uma limitação deste trabalho, pois foi elaborando em um ambiente virtual baseada em sistema Linux. O objetivo é apresentar uma proposta de arquitetura onde elementos são apresentados. Não é objetivo deste trabalho definir quais os melhores elementos, e sim uma visão geral sobre a forma como foi desenvolvido neste trabalho.

5.2 Carregar e Popular a Ontologia

O processo de carregar e popular a ontologia é realizado através da utilização da ferramenta OntoMig, desenvolvida para este trabalho com a linguagem de programação Java. Suas etapas de trabalho e suas camadas de aplicação que envolvem o fluxo de processos realizados através da ferramenta, são descritos no capítulo 6. A figura 5.4 mostra, em resumo, as atividades realizadas neste processo de carregar e popular a ontologia.

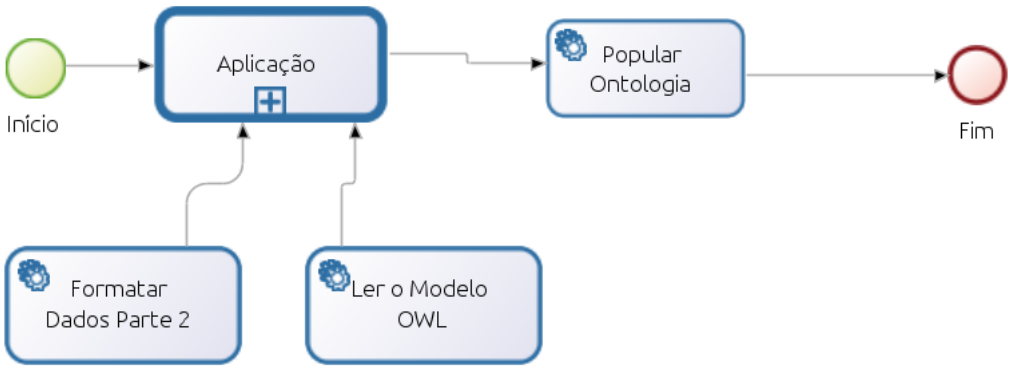

Figura 5.4 – Processo de Carga na Ontologia.

Primeiro a aplicação formata os dados. Os dados que são obtidos pelo monitor são pré-

formatados e são enviados sequencialmente de acordo com a figura 5.3. Ao receber os dados do monitor, a aplicação cria uma estrutura, baseado em orientação a objetos, para armazenar os dados e relacionar cada objeto a seus respectivos dados. Logo, se tem objetos para armazenar dados da memória, de dispositivos de armazenamento e da CPU. Para que assim, torne mais sistemático os dados para manipulação.

Então, após os dados formatados a ontologia deve ser lida pela aplicação. Ler a ontologia significa literalmente ler o arquivo OWL gerado pelo Protégé, e então carregar estas informações na memória. Este processo é necessário para manipular a ontologia durante todo o período em que seja necessária utilizá-la. A ontologia é lida uma única vez e todas as atividades realizadas na ontologia são feitas na memória, permite então uma maior rapidez em sua manipulação.

Manipular a ontologia, significa ler seus valores, como os indivíduos e suas propriedades, e também atribuir valores a eles. Então, nesta etapa, após a leitura, os dados formatados são atribuídos à ontologia. Ou seja, o ambiente virtualizado é mapeado para a ontologia. Informações referente aos recursos utilizados pelos servidores estarão contidas na ontologia para posterior processamento.

Para trabalhar com Ontologias, ou seja, com arquivos OWL na linguagem de programação Java, é possível através da utilização de APIs. Existem diversas bibliotecas disponíveis para trabalhar com ontologia em Java. A biblioteca Jena (JENA, 2014) através de sua API para ontologia, tem como objetivo fornecer uma interface de programação consistente para desenvolvimento de aplicativos com ontologia, independente de qual idioma de ontologia que se está usando em seus programas. A API de ontologia de Jena é uma linguagem neutra: os nomes de classe Java não são específicos para a linguagem subjacente. Por exemplo, a classe *OntClass* pode representar uma classe OWL ou uma classe RDF. Para representar as diferenças entre as várias representações, cada uma das linguagens de ontologia tem um perfil, que lista as construções autorizadas e os nomes das classes e propriedades.

O perfil está ligado a um modelo de ontologia, que é uma versão estendida do Modelo de classe Jena. O modelo básico permite o acesso às declarações em uma coleção de dados RDF. *OntModel* estende isso adicionando suporte para os tipos de construções que se espera estarem em uma ontologia: Classes (em uma hierarquia de classes), propriedades (em uma hierarquia de propriedade) e indivíduos.

O próprio serviço de validação de arquivos RDP mantida pelo Consórcio W3C[6], utiliza um interpretador Jena para validar a estrutura e extrair tuplas destes arquivos. Também existem vários outros projetos relacionados com Jena[7]. Desta forma, existe uma grande comunidade de desenvolvedores utilizando a ferramenta, assim garante uma grande confiabilidade na mesma. Por isto foi optado para este trabalho a API Jena como sendo a ferramenta mais adequada.

Então, através da biblioteca Jena em conjunto da linguagem Java, é possível realizar a leitura de arquivo OWL, modelo da ontologia, e realizar o processamento desse arquivo. Processamento esse que envolve a leitura do arquivo, a escrita do arquivo, consultas e navegação nos modelos, e por fim, a realização de operações nos modelos. Operações que envolvem a adição de propriedades, valores de propriedades, restrições e instâncias ou indivíduos.

5.3 Realizar a migração

Nesta etapa, a aplicação possui a ontologia mapeada, com as informações do uso de recursos computacionais do ambiente virtual. A partir disso, é necessário consultar a ontologia e verificar a partir de métricas se há necessidade da migração de VMs, isto é, se o ambiente está em desequilíbrio de carga. A figura 5.5 mostra as atividades principais desta parte da arquitetura, responsável pela realização das consultas e migração de máquinas virtuais.

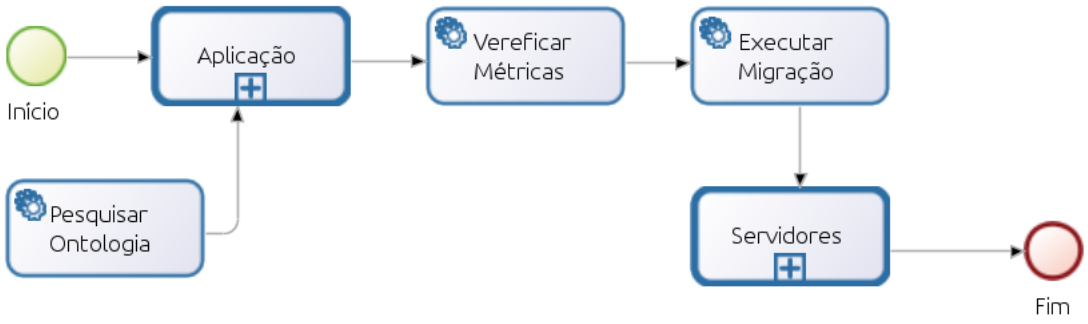

Figura 5.5 – Processo de verificação de métricas e migração.

A partir da carga da ontologia, a aplicação tem disponível o mapeamento das informações dos servidores. Então, é possível realizar as consultas ou inferências sobre o conhecimento e verificar, pelo resultado, qual o estado atual de carga dos servidores. Este processo é importante pois aqui a aplicação visualiza o estado do ambiente para então decidir se realiza migração ou aguarda novos dados dos monitores. O estado do ambiente sinaliza naquele momento espe-

[6] http://www.w3.org/RDF/Validator/
[7] http://jena.apache.org/about_jena/contributions

cífico as condições dos servidores em relação aos seus recursos, por isso, com estas informações é possível verificar se existe alguma indicação de desequilíbrio.

Com estas informações a aplicação permite a execução de métricas estabelecidas, e a verificação da possibilidade de migração. Percebendo a necessidade, a aplicação executa a rotina de migração. Esta rotina comunica-se com *deamons* responsáveis por estabelecer comunicação com os hipervisores. Assim, a migração é realizada através destes *deamons*.

Para realizar a migração de máquinas virtuais, neste trabalho, a aplicação OntoMig utiliza-se da biblioteca *Libvirt*, que é uma interface de programação de aplicativo para criar ferramentas de gerenciamento de máquinas virtuais compatíveis com vários VM's (LIBVIRT, 2014).

Além da API *Libvirt* é necessária a utilização de um servidor NFS (*Network File System*) para compartilhar as imagens que são geradas a partir da instalação das VM's para que todas as máquinas físicas possam ver esse diretório. NFS é um sistema de arquivos distribuídos desenvolvido inicialmente pela *Sun Microsystems*, a fim de compartilhar arquivos e diretórios entre computadores conectados em rede, formando assim um diretório virtual. Segundo Sandberg (1985), NFS foi projetado para tornar os recursos do sistema de arquivos compartilhados em uma rede de máquinas não homogêneas mais fácil. E também, fornecer um caminho para tornar arquivos remotos disponíveis para programas locais.

Para realizar a migração de máquinas virtuais existem duas opções para o transporte dos dados pela rede: transporte nativo do hipervisor e transporte por tunelamento da *Libvirt*. No primeiro, normalmente têm os menores custos computacionais, minimizando o número de cópias de dados envolvidos. No entanto, o transporte de dados nativos vai exigir etapas extras de configuração de rede específicas do hipervisor pelo administrador. Através do tunelamento do transporte não é exigida nenhuma configuração de rede adicional para além do que já é exigido para *libvirtd* no acesso remoto, SSH, bem como permite o uso de criptografia. Neste trabalho foi optado pelo uso do transporte através de tunelamento uma vez que a biblioteca *Libvirt* permite recursos simples como o tunelamento evitando grandes configurações nos servidores.

Ainda é necessário definir a métrica para realizar a migração de máquinas virtuais. Existem diversos trabalhos que utilizam diferentes formas para decidir sobre migração. No trabalho de Khanna (2006) as máquinas virtuais são dispostas por ordem decrescente dos recursos utilizados e os servidores físicos são organizados em ordem crescente a partir de seu custo de migração. A carga é equilibrada por migrar máquina virtual com alta carga de recursos para

servidor físico com baixo custo de migração.

Para este trabalho foi optado pela métrica definida por Arzuaga (2010) que propõem uma técnica que atribui peso para todos os recursos utilizados pelos servidores. Os recursos utilizados por máquinas virtuais são claramente previstos apenas pela camada de hipervisor. O *Virtual Server Load* (VSL) é a métrica que é calculada usando os recursos utilizados na camada hipervisor.

Uma propriedade desejável de uma métrica de carga do servidor virtualizado é que ele pode ser utilizado em sistemas heterogêneos. Sistemas de *Cluster* no mundo real geralmente são homogêneos, então para que essa métrica possa ser aplicada a uma variedade de configurações de sistema, é preciso quantificar a carga de um servidor de uma forma que não dependa de unidades de recursos fixos. Uma métrica sem unidade permite a comparação direta de diferentes servidores no sistema, independentemente de seus componentes internos.

Para efeitos da definição de tal métrica, dada por Arzuaga (2010), seja S o conjunto de servidores físicos e VM_{Host} o conjunto de máquinas virtuais em execução no servidor físico $Host$, $(Host \in S)$. Em seguida, a carga do servidor virtualizado (VSL_{Host}) pode ser expressa como:

$$VSL_{Host} = \sum_{resource} W_{resource} * \frac{\sum_{v \in VM_{Host}} v_{resource} usage}{Host_{resource} capacity}$$

Onde o $resource \in \{$*CPU*, *memória*, *disco*$\}$ e $W_{resource}$ é um peso associado a cada $resource$. Note-se que esta medida de carga não tem unidades e começa em 0 quando o uso de recursos das VMs é zero. O valor máximo depende dos valores de peso atribuído ($W_{resource}$), nesse caso quando as VMs consomem todos os recursos disponíveis. VSL_{Host} varia de forma dinâmica, dependendo das VMs em execução no sistema físico ($Host$). Esse fato torna a medida adequada para uma variedade de características do sistema, tais como uma métrica para balanceamento de carga. Para $v_{resource} usage$ tem-se o uso de recurso da máquina virtual e $Host_{resource} capacity$ tem-se pela capacidade disponivel de recurso da máquina host.

Esta carga dos servidores virtualizados é calculada para cada servidor no *Data Center*. Com base no valor VSL as máquinas são migradas de servidores muito utilizados para os servidores menos utilizados.

6 APLICAÇÃO ONTOMIG

Este capítulo apresenta a aplicação desenvolvida para a arquitetura proposta. Na arquitetura especificada neste trabalho, a aplicação tem como objetivos: gerenciar os monitores de recursos computacionais dos servidores e das VMs; trabalhar com o conhecimento estabelecido através da ontologia, ou seja, manipular o modelo OWL; verificar a necessidade de migração através de regras determinadas; estabelecer comunicação com os servidores e as máquinas virtuais para realizar as migrações, quando necessárias.

Esta aplicação foi desenvolvida em Java (2014) e foi denominada de OntoMig. A aplicação OntoMig possui 2 camadas: aplicação e negócio. Na camada de aplicação estão as classes responsáveis por interagir com a camada de negócio, e realizar algumas tarefas. Nessa camada tem-se classes como *OntoHostInfo*, responsável por buscar informações sobre as máquinas *hosts* e exibir em uma janela para o usuário, e *OntoPesquisar*, responsável por buscar máquinas que estejam dentro de critérios estabelecidos pelo usuário nessa janela e retornar em formato de texto. Do mesmo modo, a classe *OntoPopular* com a função de popular a ontologia com as informações advindas dos *scripts* monitores. A figura 6.1 apresenta a relação entre essas camadas do sistema.

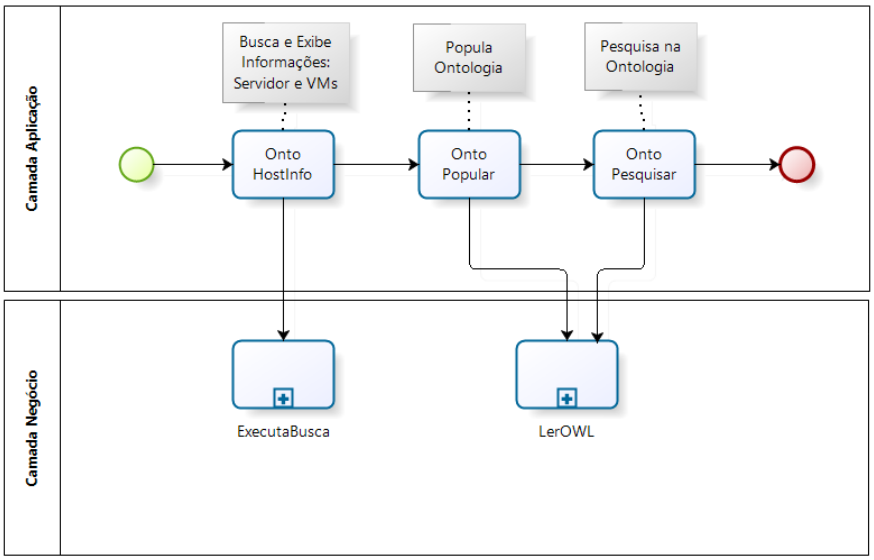

Figura 6.1 – Relação entre as camadas de Aplicação e Negócio.

A camada de aplicação possui três principais janelas de interação com o usuário, onde o usuário pode realizar separadamente as funções de: buscar informações sobre uso dos recursos dos servidores e VMs; popular a ontologia; e pesquisar na ontologia. Na primeira janela de

interação o sistema utiliza da classe *ExecutarBusca* e nas janelas seguintes utilizam a classe *LerOWL*.

Na camada de negócio estão as classes responsáveis por trabalhar com os arquivos OWL, tais como: ler o arquivo e trazer o modelo para a memória; executar os *scripts* nas máquinas para adquirir os dados dos recursos utilizados; e também é onde tem-se as classes que irão armazenar as informações das máquinas. A classe *ExecutaBusca* é responsável por conectar via SSH e executar os comandos para obter informações de recursos, já a classe *LerOWL* tem como função a leitura de arquivos OWL e carregar o modelo para memória, bem como oferece recursos para atribuir valores e resgatar valores de propriedades permitindo popular a ontologia. A figura 6.2 apresenta o diagrama de classes da camada de negócio da aplicação.

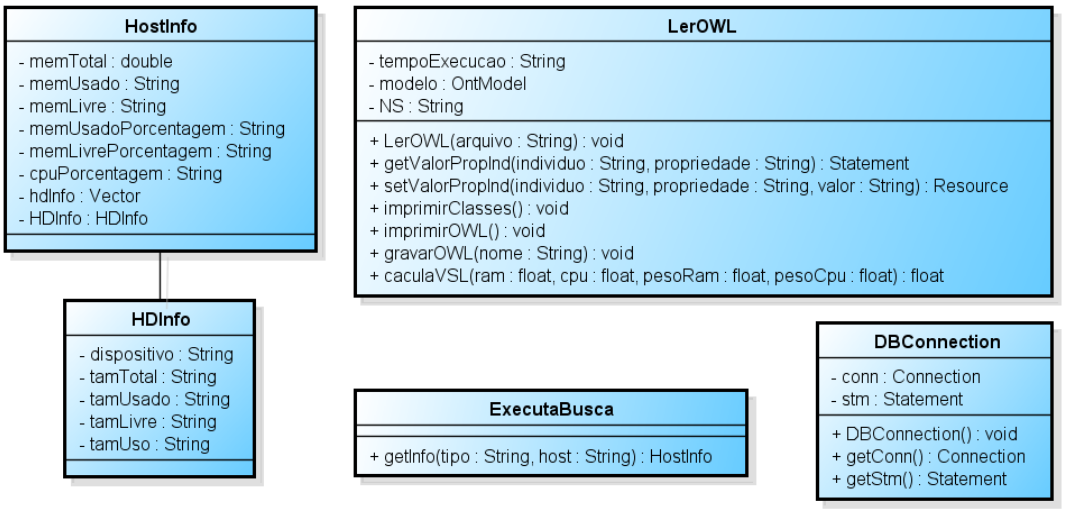

Figura 6.2 – Principais Classes da Aplicação OntoMig.

A classe *HDInfo* é utilizada para armazenar os dados, obtidos através dos *scripts* monitores, referentes aos dispositivos de armazenamentos. Esses dados são formatados e a partir disto é criado um objeto para cada dispositivo, quando existir mais que uma unidade de armazenamento. A classe *DBConnection* é utilizada para conexão com banco de dados. No banco de dados foram armazenados: tempo de espera para requisição de informações do estado dos servidores; valor limite da carga dos servidores considerada para migração das VMs; e o peso dos recursos. A aplicação OntoMig permite que o usuário altere cada uma destas informações e salve no banco de dados.

Para isso, na aplicação OntoMig foi utilizada a ferramenta SQLite. SQLite é um mecanismo de banco de dados SQL embutido. Ao contrário da maioria dos outros bancos de dados SQL, o SQLite não tem um processo para um servidor separado. SQLite lê e escreve direta-

mente em arquivos comuns de disco. Um banco de dados SQL completo com várias tabelas, índices, gatilhos e *views*, ficam contidos em um único arquivo em disco (SQLITE, 2015).

Esta aplicação possui 5 (cinco) etapas de trabalho, que são as seguintes:

1. Ler modelo OWL (Ontologia);
2. Requisitar informações aos agentes;
3. Popular a ontologia;
4. Realizar consultas;
5. Realizar a migração.

As 4 primeiras etapas são executadas a cada um minuto. Ou seja, a cada minuto a aplicação executa a leitura da ontologia, requisita informações aos agentes, popula a ontologia e realiza consultas. A última etapa é executada quando, após o resultado das consultas, existir necessidade de realizar alguma migração.

6.1 Ler modelo OWL (Ontologia)

Com relação ao primeiro item de atividade da aplicação OntoMig, a leitura do modelo OWL, que representa a ontologia, é realizada quando a aplicação é iniciada e mantida em memória para manipulação da ontologia até que a aplicação seja encerrada. A API permite três formas de armazenar o modelo lido do arquivo: *in-memory*, SDB ou TDB. Na primeira opção, *in-memory*, o modelo é alocado na memória. SDB usa um banco de dados SQL para o armazenamento e consulta de dados RDF. E TDB é um componente de Jena, para consulta e armazenagem RDF através de *datasets*. Para este trabalho foi optado pela utilização do método *in-memory* pela simplicidade, uma vez que o modelo permanece em memória, o processo para sua manipulação é então simplificado comparado com o processo de abrir uma conexão no banco de dados ou manipular arquivos diretamente.

A API Jena permite a leitura do arquivo OWL independente do perfil que esse arquivo seja definido na construção, basta especificar o modelo para o método que irá criar o modelo em memória. Nesse caso a ontologia foi definida a partir do perfil *OWL-DL*, então para o método de leitura especifica-se *OWL_DL_MEM*.

6.2 Requisitar informações aos agentes

Para requisitar as informações, o programa acessa as máquinas através de SSH e então executa os *scripts* monitores nas máquinas. Então esses *scripts* obtêm como retorno as informações em ordem pré-definida, apresentado na seção 5.1, e a aplicação OntoMig formata esses dados. A partir de então é possível trabalhar esses dados juntamente com o modelo OWL que está armazenado em memória.

6.3 Popular Ontologia

A aplicação é dividida em camadas onde cada uma possui classes específicas para cada tarefa que deverá ser realizada. Na camada de negócio a classe *LerOWL* tem como função a leitura de arquivos OWL e carregar o modelo para memória, bem como oferece recursos para setar valores e resgatar valores de propriedades permitindo popular a ontologia.

Esta classe possui método para atribuir valor a uma propriedade de indivíduos, método para resgatar valor de uma propriedade de indivíduos. Esse primeiro método é utilizado para realizar o mecanismo de popular ontologia através da passagem dos parâmetros: nome do indivíduo, nome da propriedade e valor da propriedade, assim, com estas informações de forma composta é possível inserir valor em uma propriedade. Então com as informações coletadas pelo *script* monitor várias inserções são realizadas até que a ontologia esteja totalmente populada com as informações necessárias.

Também, existe o método para imprimir na tela o modelo OWL atualmente em memória para realizar verificações quando necessário, como verificar se as inserções das informações estão sendo feitas corretamente pela aplicação. Ainda, há um método para gravar o modelo em arquivo OWL. Para saída de arquivo existem 4 possibilidades disponibilizadas pela API, “RDF/XML”, “RDF/XML-ABBREV” e também “N-TRIPLE”, e “TURTLE”. “RDF/XML” produz uma saída regular razoavelmente eficiente, mas não é legível. Em contraste, “RDF/XML-ABBREV”, produz saída legível sem levar em conta a eficiência. No entanto para o desenvolvimento da ontologia foi utilizado “RDF/XML-ABBREV” por ser legível. Nesse sentido, para manter o padrão definido anteriormente foi optado pela mesma utilização no método de escrita do arquivo.

A aplicação OntoMig permite, também, a persistência da ontologia em banco de dados através da API SDB. A ideia não é trabalhar com versionamento de estado do ambiente virtu-

alizado, mas com a persistência do estado anterior que pode ser utilizado para comparação e verificação da medida atual realizada pelo *script* monitor e a medida feita anteriormente. Esta comparação pode ser feita com a ideia de verificar, por exemplo, se a situação *x* ou *y* permanece ou se a situação do ambiente alterou seu estado.

6.4 Realizar consultas sobre a Ontologia

Para realizar as consultas sobre a ontologia é utilizada a linguagem SPARQL (DODDS, 2014), que é uma linguagem de consulta e protocolo de acesso de dados para a Web Semântica. SPARQL é definido em termos de modelo de dados RDF pela W3C e vai trabalhar para qualquer fonte de dados que podem ser mapeados em RDF. Ainda SPARQL pode ser usado para expressar consultas em diversas fontes de dados, sendo os dados armazenados nativamente como RDF ou visto como RDF via *middleware*. SPARQL contém recursos para consultar padrões grafos obrigatórios e opcionais, juntamente com suas conjunções e disjunções. SPARQL também suporta teste de valor extensível e consultas de restrições pelo grafos de RDF. Os resultados de consultas SPARQL podem ser conjuntos de resultados ou grafos RDF (SEABORNE; PRUD'HOMMEAUX, 2014).

Abaixo tem-se um exemplo de consulta utilizando SPARQL dentro da linguagem Java. No código está sendo atribuído a uma variável *String* a consulta que posteriormente é realizada por um método. O processo de realização da consulta se dá pela criação da consulta pelo método *QueryFactory.create(queryString)*, para então ser criada a consulta para execução onde é passado como parâmetro a consulta criada anteriormente, mais o modelo da ontologia que está em memória, *QueryExecutionFactory.create(query, lerOWL.getModelo())*. E por fim, a consulta é executada com *qe.execSelect()* e o seu retorno armazenado em um *ResultSet*:

```
String queryString = "PREFIX foaf:" +
" <http://www.owl-ontologies.com/OntologyToMigration.owl#>" +
" SELECT ?vm ?ram ?disk" +
" WHERE {" +
" ?vm a foaf:VirtualMachine." +
" ?vm foaf:ramUsage ?ram." +
" ?vm foaf:diskUsage ?disk." +
" FILTER (?ram > valor)}";
```

Nesta consulta apresenta-se a semelhança da estrutura com uma estrutura de pesquisa SQL (*Structured Query Language*) (CHAMBERLIN; BOYCE, 1974), linguagem de pesquisa declarativa padrão para banco de dados relacional (base de dados relacional), que tornou-se um padrão adotado pela grande maioria de gerenciadores de banco de dados pela sua simplicidade e facilidade de uso. Nesse sentido, o SPARQL adota por usar uma estrutura semelhante para facilitar a seu uso. Do mesmo modo, pela familiaridade que programadores já possuem com o SQL, então a aprendizagem do SPARQL se torna mais rápida.

6.5 Realizar a migração

Nesta etapa foi utilizado o *software virsh*. Esse *software* faz parte da distribuição *Libvirt*. É uma ferramenta de interface de linha de comando para gerenciar convidados e hipervisor através da *Libvirt*.

Uma vez populada a ontologia com as informações sobre os recursos das máquinas físicas e virtuais do ambiente, se faz necessário a verificação da necessidade de migração. Essa verificação é feita pelo índice de carga dos servidores calculado pela métrica estabelecida, VSL. Baseado no limite do índice VSL estipulado pelo usuário da ferramenta OntoMig, é então realizada a migração de um servidor carregado para outro menos carregado.

A ferramenta OntoMig então executa o comando que ira migrar a máquina virtual a partir do acesso SSH à máquina física que está sobrecarregada.

```
virsh migrate VM_X qemu+ssh://desthost/system
```

Esse comando irá informar à biblioteca *Libvirt* do servidor, em que o comando foi executado, de que a VM de nome “VM_X” deve ser migrada para o destino informado na URL “qemu+ssh://desthost/system”.

6.6 Considerações do Capítulo

Este capítulo apresentou uma proposta de arquitetura para um sistema de migração de máquinas virtuais utilizando ontologias como ferramenta para mapear o ambiente de virtualização. A arquitetura que aqui foi apresentada neste capítulo tem como referência alguns trabalhos já realizados por outros autores como Zhang (2012), Wei (2011) e Ye (2012). Esses autores desenvolveram *frameworks* para migração de máquinas virtuais e ambos apresentam a mesma base de arquitetura.

A implementação desta arquitetura, através da utilização da ontologia desenvolvida neste trabalho, possibilita o desenvolvimento de aplicações que: (a) necessitam de informações do ambiente virtualizado; (b) permitam realizar migrações de máquinas virtuais a partir de um ambiente virtualizado; (c) garantem a disponibilidade de recursos nos serviços disponíveis em um ambiente de computação em nuvem.

A fim de validar a arquitetura proposta neste trabalho, no próximo capítulo é apresentado um estudo de caso que apresenta uma situação real do funcionamento de um sistema de migração de máquinas virtuais desenvolvido com base na arquitetura proposta.

7 ESTUDO DE CASO E RESULTADOS

Para validar a funcionalidade da proposta deste trabalho, da mesma forma exemplificar o funcionamento da mesma, foi realizado um estudo de caso. Este estudo de caso tem como objetivo apresentar o uso de ontologia para migração de máquinas virtuais a partir da arquitetura proposta, assim, através de um ambiente virtualizado onde aplicações monitores verificam o estado de carga dos servidores é possível interagir com as máquinas virtuais realizando a migração para garantir a otimização e utilização dos recursos computacionais.

7.1 Ambiente de Virtualização para Migração

Para realizar o estudo de caso é apresentado, então, um cenário. Nesse cenário é aplicado o enfoque em Técnicas de Migração para Balanceamento de Carga. Neste trabalho foi utilizado o cálculo de carga do servidor virtualizado (VSL - *Virtualized Server Load*) mencionado em Arzuaga (2010) na seção 2.2. O objetivo é demonstrar o uso da ontologia para migração de máquinas virtuais a partir da arquitetura desenvolvida, e para tanto foi arquitetada uma situação em que é necessário balancear a carga dos servidores.

Para formar esse ambiente de virtualização foram configurados para esse estudo de caso dois servidores: Servidor 1 (PM1) é um servidor Ubuntu 12.04 x64 em um computador Dell XPS com 6GB de RAM, Processador Intel i5 2.4GHz, com um HD de 500GB; Servidor 2 (PM2) é servidor Ubuntu 12.04 x64 em um computador Dell Latitude com 2GB de RAM, processador Intel Dual Core 1.8GHz, com um HD de 120GB. Como o ambiente é pequeno foi configurado no primeiro servidor o serviço NFS para o compartilhamento das imagens das máquinas virtuais, ou seja, todas as máquinas virtuais criadas nos dois servidores estarão sendo compartilhadas pelo servidor NFS, uma vez que a memória e os processos é que são migrados.

A aplicação desenvolvida irá ser executada no Servidor 1, a partir do qual serão realizadas as requisições sobre os recursos que estão sendo utilizados pelos servidores. Localmente é necessário o acesso SSH com as máquinas virtuais, ou seja, estas máquinas devem estar devidamente configuradas com serviços SSH. Para que aplicação acesse o outro servidor também é realizado via SSH, e assim também acontece a relação do Servidor 2 e suas máquinas virtuais. Desta maneira os *scripts* são executados todos via SSH.

No Servidor 1 foram configuradas duas máquinas virtuais, denominadas de: *vm12* e *vm14*. Ambas as máquinas foram configuradas com o sistema operacional Lubuntu 12.04. Lu-

buntu é um sistema operacional rápido e leve. O núcleo do sistema é baseado em Linux e Ubuntu e utiliza como ambiente gráfico para *desktop* o LXDE (*Lightweight X11 Desktop Environment*) e uma seleção de aplicações leves (LUBUNTU, 2014). Ambas as VMs possuem as seguintes configurações: 768 MB de RAM; 8GB de HD e 2 CPUs lógicos. No Servidor 2 foi configurado somente uma máquina virtual, denominada de: *vm16*. A configuração desta VM é idêntica à do Servidor 1. Maiores detalhes sobre a configuração do ambiente de virtualização, instalações e comandos, estão descritos no Apêndice A.

Uma aplicação foi desenvolvida para realizar o gerenciamento do ambiente de virtualização e utilizar-se das informações fornecidas pela ontologia para migração das máquinas virtuais. Na figura 7.1 é apresentado o fluxo da aplicação *OntoMig* onde as etapas do sistema são apresentadas sequencialmente.

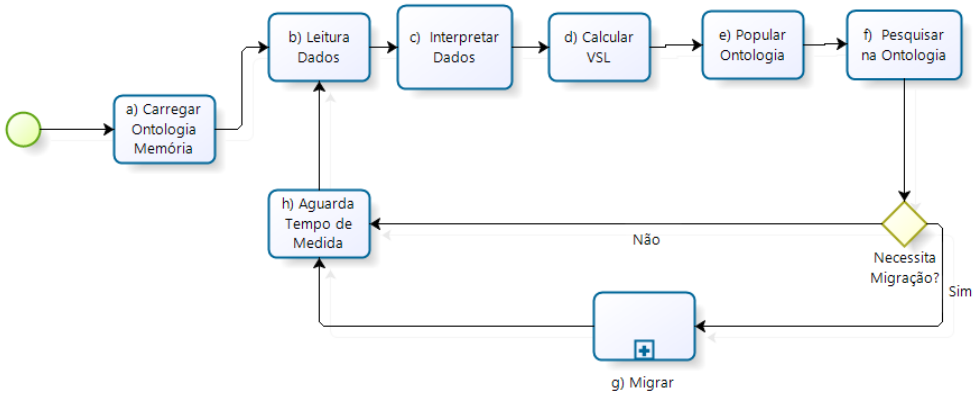

Figura 7.1 – Fluxo da Aplicação OntoMig.

Para realizar a migração de máquinas virtuais a aplicação OntoMig utiliza-se da biblioteca *Libvirt*, que possui duas maneiras de controle distintos, local e remotamente. No caso deste trabalho como se tem dois servidores e existem VMs em *host* separados do aplicativo de gerenciamento, logo, aqui é necessária a comunicação remota. Ou seja, esse modo usa um

daemon especial chamado *libvirtd* que é executado no *host* remoto.

Algoritmo 1: Processo de monitoramento e migração de VMs

```
Entrada: Ontologia, Dados dos Servidores e VMs
Saída: Nenhuma saída
início
    Leitura Ontologia (arquivo OWL);
    enquanto verdadeiro faça
        Eb = ExecutarBusca();
        enquanto existir máquinas faça
            Informacoes = Eb.buscar("dominio@ip");
            CalcularVSL(Informacoes);
            PopularOntologia(Informacoes);
        fim
        ServidorAlvo = PesquisarOntologia(VSL);
        se ServidorAlvo = verdadeiro então
            VMAlvoMaiorVSL = PesquisarOntologia(VSL, ServidorAlvo);
            ServidorBaixoVSL = PesquisarOntologia(VSL);
            Migrar(VMAlvoMaiorVSL, ServidorBaixoVSL);
        fim
        Dormir(60s);
    fim
fim
```

A partir do momento em que o sistema é iniciado o primeiro evento é lançado. A primeira leitura da ontologia (a) demora em torno de 3 (três) segundos, mas uma vez estando na memória não há necessidade de uma nova leitura já que o modelo fica armazenado em memória. A partir de então um segundo evento é disparado e o sistema entra em um *loop* contínuo de tarefas a realizar. Sequência lógica do processo de monitoramento e migração das VMs é apresentado por Algoritmo 1.

A leitura de dados (b) é realizada pela execução dos arquivos contendo *scripts* de busca de informações sobre o recurso utilizado pelos servidores e recursos utilizados pelas máquinas virtuais. Os dados vêm formatados de forma sequencial separados por quebra de linha, ou seja, esses dados precisam ser lidos e interpretados (c). Esta interpretação dos dados é efetuada por uma classe chamada *ExecutaBusca* que realiza as tarefas (b) e (c) sequencialmente. Internamente esta classe organiza os dados e o retorno das informações são dois Objetos: *HostInfo* que contém *HDInfo*.

Uma vez tendo as informações é necessário antes de popular a ontologia calcular o valor de carga dos servidores (d) e das VMs. Esse cálculo é realizado utilizando a métrica definida na subseção 5.3. Com a medida de carga de todas as máquinas é possível verificar

o quanto elas estão carregadas e delimitar um valor em que pode ser considerado importante realizar a migração. Para este trabalho fazem parte do cálculo de carga de virtualização o uso de processador e o uso da memória.

Para realizar o cálculo do VSL é utilizado um método *calculaVSL* da classe *LerOWL* onde o retorno é um valor o qual irá ser utilizado como VSL. O cálculo é realizado passando como parâmetro os valores do uso de CPU e o uso de RAM com os devidos pesos que cada recurso irá possuir. Para validar, tem-se um exemplo: em um caso em que o uso do processador está em 95% e o uso de memória está em 85%, e o peso dos recursos consecutivamente, valeriam $0,5$ e $0,5$, então se tem:

$$VSL = (0,5) * \frac{95}{100} + (0,5) * \frac{85}{100}$$

O resultado desta operação é $0,9$. Ou seja, o servidor tem uma carga de $0,9$. Como a soma dos pesos dos recurso é 1, então, quando os recursos estão no máximo do uso, tem-se um VSL de 1, e quando nenhum recurso está sendo utilizado tem-se um VSL de 0.

A etapa seguinte é a população da ontologia (e). Para realizar esta tarefa o sistema dispõe dos métodos *getValorPropInd* e *setValorPropInd*. Esses métodos da classe *LerOWL* permitem, respectivamente, resgatar valores de propriedades de indivíduos ou classes e gravar valores de propriedades na ontologia. Enfatiza-se que esse processo é executado rapidamente, em média $0,022$ segundos, uma vez que esta tarefa é realizada em memória.

Com a ontologia populada, ela então, está com o ambiente virtualizado mapeado em memória. A partir de agora, consultas podem ser realizadas sobre a ontologia (f), através da linguagem SPARQL, bem como executar inferências. A pesquisa é realizada baseada no VSL, o qual mostra o quanto um servidor está carregado:

```
String queryString = "PREFIX foaf:" +
" <http://www.owl-ontologies.com/OntologyToMigration.owl#>" +
" SELECT ?pm ?vsl" +
" WHERE {" +
" ?pm a foaf:PhysicalMachine." +
" ?pm foaf:vsl ?vsl." +
" FILTER (?vsl > valor_limite)}";
```

Uma vez que o VSL ultrapassa o valor limite desejável, então se tem um candidato sobrecarregado, nesse sentido procura-se realizar a migração para outro servidor que não esteja

sobrecarregado. Para verificar qual máquina virtual deve ser migrada é realizada uma pesquisa sobre a carga das VMs nesse servidor. Não havendo servidores e máquinas virtuais sobrecarregadas o fluxo do sistema irá para o modo de espera (h) até a próxima leitura de dados dos servidores, assim iniciando todo processo novamente.

Encontrada uma VM que necessite ser migrada então o fluxo do sistema passa para a sua etapa de migração (g). Nesta etapa a aplicação OntoMig utiliza-se do *software virsh*. Uma vez que este *software* faz parte da distribuição *Libvirt* logo não há necessidade de nenhuma instalação extra. Logo abaixo é apresentado o comando utilizado para realizar a migração.

```
virsh migrate nome_da_VM qemu+ssh://usuario@servidor/system
```

Realizada a migração o sistema passa para a etapa de espera (h) por uma nova leitura de dados sobre os recursos dos servidores, assim iniciando todo processo novamente. Esse é o ciclo completo da aplicação desenvolvida para demonstrar o uso de ontologia na migração de máquinas virtuais a partir da arquitetura proposta nesse mesmo trabalho. Cabe ressaltar que este trabalho não tem o objetivo de avaliar diferentes métricas de balanceamento de carga de servidores. Da mesma forma, este trabalho não tem o objetivo de avaliar o desempenho da migração nesse sistema.

A aplicação foi desenvolvida de tal forma que cada uma das etapas realizadas automaticamente pelo sistema possa ser ativa individualmente pelo usuário através dos seus itens no menu. No menu *Programa* possui um item de menu *Critério* onde pode ser configurado o tempo de espera do ciclo de tarefas, que por padrão é de 60 segundos, ou seja a cada 60 segundos o sistema inicia o fluxo a partir da etapa (b). Nesse item de menu pode ser configurado, também, o peso da CPU e da memória RAM, utilizados no cálculo do VSL, e definir o valor limite de carga. Logo, nesse item tem-se as configurações iniciais.

Em *Host* estão os itens de menu que possibilitam a busca de informações nos servidores e retornam em formato legível ao usuário. Desta forma, o usuário pode verificar o estado dos servidores individualmente independente do processo automático da ferramenta. A figura 7.2 apresenta uma consulta realizada sobre o Servidor 1 sob um dado momento. Estas mesmas informações podem ser consultadas para as VMs locais e remotas e *host* remoto.

No menu *Ontologia* é possível popular a ontologia manualmente. Diferentemente do processo automático, o processo manual grava em um arquivo a ontologia populada e esse arquivo é armazenado em um diretório chamado *arquivosOntologia*. Deste modo, o usuário

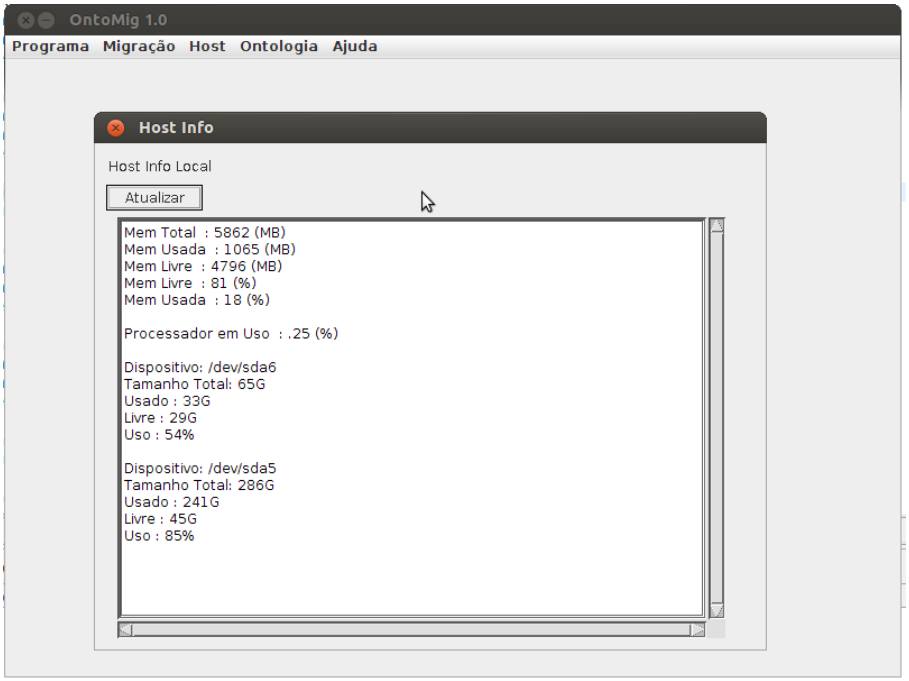

Figura 7.2 – Informações coletadas de uma máquina HOST.

pode armazenar um histórico de suas consultas que poderão servir para motivos de verificação de estados de carga dos servidores.

7.2 Resultados Obtidos

Para verificar o funcionamento da arquitetura utilizando a ontologia e a atividade relacionada à aplicação desenvolvida, realizou-se um teste em um ambiente virtualizado de forma controlada, ou seja, não é um ambiente real de uma empresa ou de um *Data Center*, mas sim, um ambiente reduzido e controlado para viabilizar os testes sem variáveis que poderiam vir a produzir resultados inadequados para pesquisa. Isso não significa, que os testes não poderiam ser realizados em ambientes reais, uma vez que as diferenças estão na quantidade de servidores e nos números de acessos aos serviços pelos usuários. No entanto, para o objetivo deste trabalho, optou-se por um ambiente controlado pois atende as necessidades para realização dos testes.

O ambiente virtualizado foi explanado nas seções anteriores. Para trabalhar com o uso de CPU na VM e produzir o aumento do uso deste foi utilizado o *Benchmark SysBench* (KOPYTOV, 2014), também utilizado no trabalho de Zhang (2012), ele é uma ferramenta de *benchmark* modular, multiplataforma e *multi-threaded* para avaliar parâmetros de sistema operacional que são importantes para um sistema executando um banco de dados sob carga intensa,

no entanto ele possui *benchmarks* individuais e um dos itens que ele avalia é a CPU. Quando executado com a carga de trabalho da CPU, *Sysbench* irá verificar números primos fazendo divisão padrão do número por todos os números entre 2 e a raiz quadrada do número. Se qualquer número dá um restante de 0, o próximo número é calculado. Desta maneira, esses cálculos irão fazer com que a CPU seja colocada para trabalhar.

Para trabalhar com a memória RAM foi utilizada a biblioteca *stdlib.h* na linguagem de programação C, a qual possui 4 funções para alocação dinâmica de memória: *malloc()*, *calloc()*, *free()* e *realloc()*. A função *malloc()*, a qual foi utilizada, reserva um bloco de memória com tamanho especificado em *bytes* e retorna um ponteiro de endereço de memória para o primeiro *byte*. Por exemplo, o comando abaixo irá alocar 100 elementos inteiros:

```
ptr=(int*)malloc(100*sizeof(int));
```

O teste foi realizado da seguinte forma: as máquinas virtuais foram executadas em ambos os servidores 1 e 2. No Servidor 2 a *vm16* foi definida para ser a máquina a qual é aplicada o *benchmark* de CPU, uma vez que esse Servidor tem menos recursos e proporcionaria o alcance ao limite do VSL de forma mais rápida para os testes. Através do *benchmark* foi executado o seguinte comando, os valores de referência podem ser configurados com o número de *threads* simultâneas e o número máximo de vezes que ele verifica se ele o número é primo:

```
sysbench --num-threads=2 --test=cpu --cpu-max-prime=30000 run
```

O limite estipulado para o VSL foi de 0.90, onde o servidor é considerado com nível alto de carga. Não é pretensão deste trabalho descobrir ou discutir qual o melhor índice. A figura 7.3 apresenta a evolução do uso de memória e CPU das máquinas virtuais. O ambiente foi monitorado por um determinado tempo e foi extraído desse monitoramento a faixa de tempo um pouco antecedente a migração e um pouco após a migração para que fosse possível visualizar o estado da evolução do uso de CPU e RAM. Na figura 7.3 são apresentadas 8 amostras do monitoramento, assim para cada série de VMs (*vm12*, *vm14* e *vm16*) é uma medida do estado atual do ambiente.

É possível perceber o crescente uso da memória RAM e a variação do uso da CPU. Uma vez que aos poucos foram sendo alocados maior números de *bytes* da memória RAM, sendo assim cada vez mais utilizado seu recurso, no entanto, quanto ao uso de CPU, até a quinta (5^0) medida pouca variação do processador é apresentada. Na sexta (6^0) medida apresenta

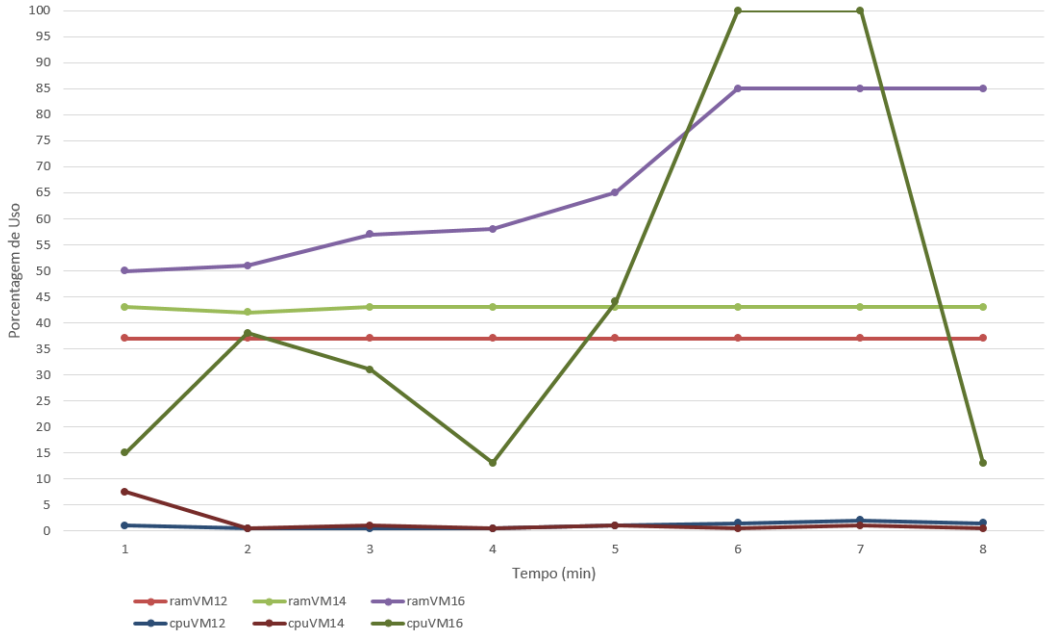

Figura 7.3 – Uso da CPU e de Memória RAM pelas VMs.

a execução do comando de avaliação do processador pelo *benchmark*, assim, elevando o uso de 100% da CPU. Desta maneira, o índice do VSL alcança o ponto de corte, apresentado na figura 7.4. A aplicação então identifica esse aumento e inicia a migração para o Servidor menos carregado.

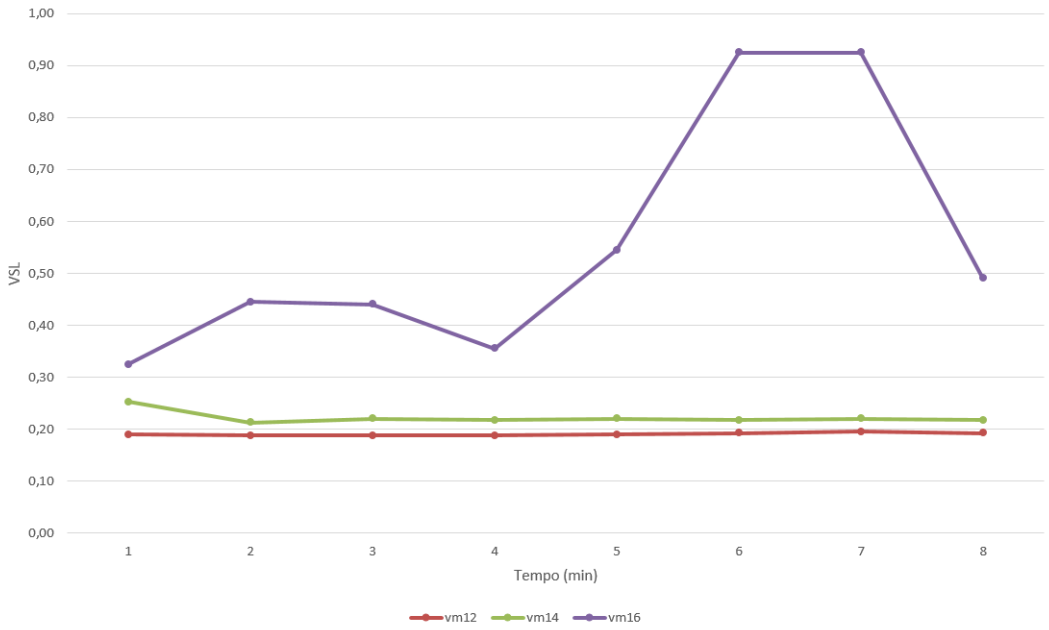

Figura 7.4 – VSL das VMs em cada série de Medida pela Aplicação OntoMig.

Na sétima (7^0) medida percebe-se que a máquina *vm16* ainda está processando o teste de CPU, pois seu VSL continua alto, na figura anterior o processamento na CPU continua 100%.

Vale ressaltar que nesta medida a migração já havia ocorrido. Identificado a necessidade de migração a aplicação realizou a migração. Os servidores pertencem à mesma rede, logo devido a pouca memória e rápida conexão a migração ocorreu entre uma medida e outra, ou seja, 1 min foi o suficiente para que o processo fosse finalizado. Como apresentado pela figura 7.5, o servidor *PM2* reduz drasticamente o uso de processador, ao contrário do servidor *PM1* que tem o aumento de processamento pois o teste do *benchmark* continua sendo executado após a migração e este é finalizado no *PM1*.

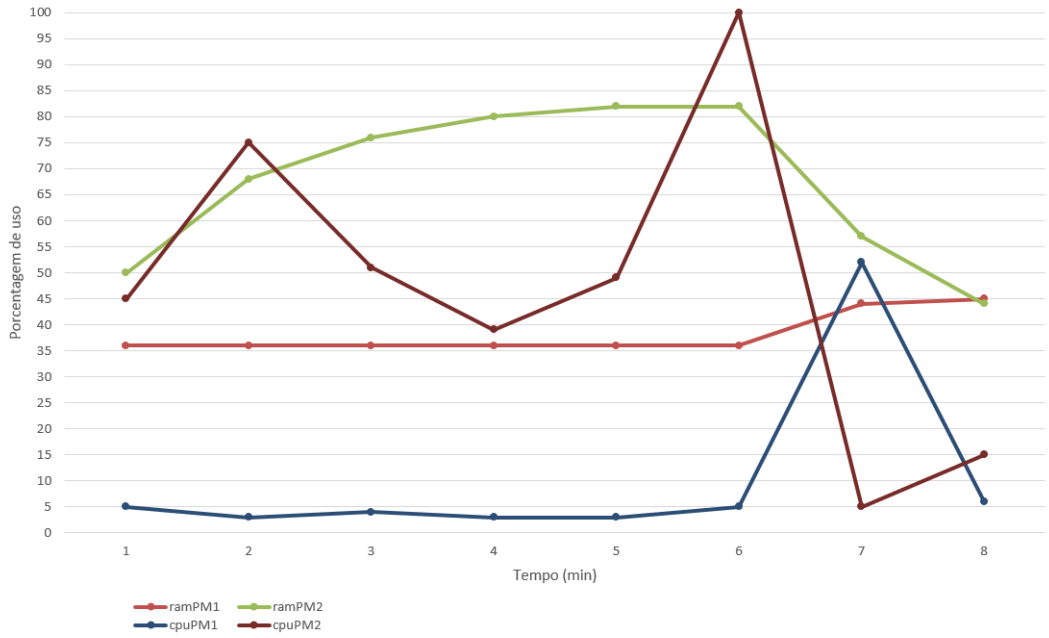

Figura 7.5 – Uso da CPU e de Memória RAM pelos Servidores 1 (PM1) e 2 (PM2).

Como mostra a figura 7.5, o processamento de PM1 sobe para em torno de 50%, e porque não para 100%? Isso é devido ao fato, de que a *vm16* possui duas unidades lógicas de processamento e o servidor 1 possui 4 unidades de processamento, sendo assim, quando a vm passou a ser executada nesse servidor ela utilizou recursos de somente 2 processadores.

Desta maneira, como mostra a figura 7.6, os servidores são equilibrados em relação ao seu índice de carga VSL. O servidor 1 tem um aumento brusco uma vez que agora está com 3 VMs rodando, no entanto, por possuir mais processamento e maior quantidade de memória RAM permanece com o valor de carga abaixo do índice de migração. Na oitava (8^0) medida o valor de carga do servidor 1 reduz pois o processamento do *benchmark* é finalizado. No entanto o VSL fica acima da sexta (6^0) medida pois por sustentar 3 VMs, ele acaba tendo que ceder mais memória para a máquina virtual *vm16* que antes estava noutro servidor.

Com relação aos tempos obtidos neste trabalho, vale ressaltar que para as medidas e

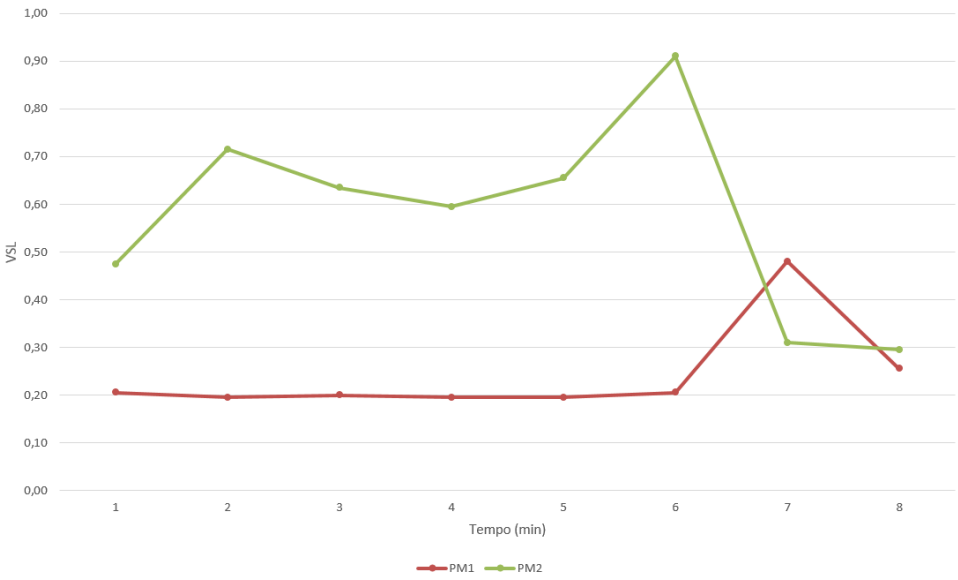

Figura 7.6 – VSL do Servidor 1 (PM1) e Servidor 2 (PM2)

em relação ao tempo de migração, ambos são subjetivos. Uma vez que, não é objetivo deste trabalho analisar esses tempos nem os respectivos significados dos mesmos.

Neste trabalho a ontologia possui poucos indivíduos e poucas relações entre estes por ser um conhecimento especifico para ambiente de migração. Por esse motivo acaba não refletindo no desempenho. Mas ao trabalhar inferências com ontologia, quanto mais indivíduos, classes e relações definidas mais lento é esse processo. Quando a biblioteca Jena executa uma inferência todas as relações são inferidas sobre todos os indivíduos existentes e este processo é lento. Não é possível garantir neste trabalho se a utilização da ontologia é melhor do que os métodos apresentados pelos trabalhos relacionados.

No entanto se alcançou um grande material com a apresentação da arquitetura e dos resultados. A arquitetura apresenta uma nova forma de realizar o mapeamento e caracterização do ambiente virtual através da ontologia, e os demais módulos apresentados, mesmo que não genéricos, uma vez que o ambiente esta sob sistema operacional Linux, possuem recursos básicos para o monitoramento de recursos computacionais e para o gerenciamento das máquinas virtuais quanto à migração.

Além disso, mostrou-se um estudo de caso com a arquitetura proposta. Apresentando, assim, que é aplicável a utilização desta arquitetura proposta apesar dos limites de não ser uma proposta genérica. E ainda há a necessidade de verificar o desempenho desta proposta com relação aos trabalhos relacionados. Ainda assim, é uma nova ideia dentro deste contexto e com a vantagem de estabelecer um padrão através da caracterização do conhecimento com a

ontologia. Uma vez que não esta claro ainda as relações e os conceitos relacionados a migração de máquinas virtuais, e também, não há métricas padronizadas para uso genérico no equilíbrio de carga dos servidores. Por isso, a ideia proposta aqui inicia uma nova fase para a padronização dos conceitos no domínio de migração de máquinas virtuais.

7.3 Considerações do Capítulo

Neste capítulo foi apresentado o ambiente de virtualização para migração no qual foram configurados dois servidores Linux para virtualização de três máquinas virtuais *vm12*, *vm14* e *vm16*. Foi especificada a configuração dos servidores e das VMs, respectivamente.

Também foi exemplificado sobre a aplicação OntoMig responsável pela formatação das informações recebidas do monitor de recursos dos servidores, pela população da ontologia para mapeamento do ambiente virtualizado, pela pesquisa na ontologia para identificar quando um servidor ultrapassar o limite definido para o valor de carga, VSL. Da mesma maneira, é de responsabilidade da aplicação chamar os métodos que realizam a migração da máquina virtual que está sobrecarregando o servidor identificado.

Enfim, foi apresentado o resultado do estudo de caso. Foi realizado um teste no ambiente onde os servidores estão com as VMs ativas e foi executado na *vm16* um *benchmark* para elevar o uso da CPU e, através da função *malloc()* da linguagem C, foram executados alguns comandos para elevar o uso de memória. Enfim, foi migrada uma máquina virtual e gerados gráficos para apresentar os resultados.

8 CONCLUSÃO

A virtualização permite múltiplos servidores virtualizados em um único *host* físico de forma isolada e segura, permitindo, desta forma, uma otimização no uso dos recursos computacionais reduzindo o custo de implantação de Data Centers.

No entanto, em algum momento servidores podem ficar sobrecarregados e outros podem ficar mais ociosos, e uma maneira para resolver isso é utilizando a migração de máquinas virtuais em tempo real, onde ocorre a migração de máquina virtual em execução juntamente com suas aplicações para outro servidor, restabelecendo, assim, o equilíbrio dos servidores. Neste sentido a aplicação de migração de máquinas virtuais em tempo real, conhecido como *Live Migration*, tem se tornado a chave para a otimização de recursos computacionais em *Data Centers*.

Assim, torna-se interessante o desenvolvimento de soluções que viabilizem a implantação desta tecnologia. Através de um ambiente virtualizado, onde aplicações monitores verificam o estado de carga dos servidores, é possível interagir com as máquinas virtuais realizando a migração para garantir a otimização e utilização dos recursos computacionais.

Para representar este ambiente virtualizado é possível utilizar-se da ontologia, pelo fato de que a ontologia possui uma forma de representar conhecimento, desta forma, ela acabada por ser comumente empregada como um método para a identificação de categorias, conceitos, relações e regras, para definir e conceituar o conhecimento em um domínio.

Considerando isto, neste trabalho foi desenvolvida uma ontologia que propõe representar um ambiente virtualizado para migração de máquinas virtuais e auxiliar a visualização do estado atual deste ambiente. A ontologia é importante para o sistema de migração de máquinas virtuais, visto que os dados descritos na ontologia descrevem os conceitos envolvidos neste tipo de ambiente. Essas informações serão utilizadas por sistemas de virtualização que implementem a arquitetura proposta neste trabalho.

Na arquitetura definida neste trabalho, foram elencados elementos necessários para a implementação de tecnologias que permitam o desenvolvimento de soluções que possam interagir com o ambiente virtualizado e realizar migrações. Nesta arquitetura foram denominados, então, monitores responsáveis pela captação das informações sobre os recursos dos servidores. Tecnologias como Jena, para população da ontologia, SPARQL para consulta em ontologias, assim como, SWRL para criação de regras para inferências. Essas últimas, permitem que um

programa possa atuar sistematicamente através das ontologias.

Para completar este trabalho, foi desenvolvida uma ferramenta que realiza o monitoramento dos servidores e das VMs, e utiliza-se da API Jena e de ferramentas como SPARQL para popular e pesquisar em ontologias. Desta forma a ferramenta completa a arquitetura exemplificada no trabalho. Para tanto, foi arquitetado um ambiente virtualizado para viabilizar teste da arquitetura. Os testes demonstraram o funcionamento da arquitetura.

Para trabalhos futuros, é interessante aumentar o ambiente de teste uma vez que o teste realizado neste trabalho foi em um ambiente controlado e não em um *Data Center* real onde diferentes situações poderiam vir a ocorrer. Da mesma forma, a arquitetura poderia ser aprimorada. Um trabalho futuro poderia ser adicionar componentes de banco de dados para armazenamento de histórico de estado do ambiente virtualizado para estudo e desenvolvimento de algoritmos de predição para trabalhar com aprendizagem. Logo o sistema poderia ”aprender” quais máquinas são mais propícias a ficarem sobrecarregadas, por exemplo.

REFERÊNCIAS

ALMEIDA, M. B.; BAX, M. P. Uma visão geral sobre ontologias: pesquisa sobre definições, tipos, aplicações, métodos de avaliação e de construção. **Ciência da Informação**, [S.l.], v.32, p.7–20, 12 2003.

ARZUAGA, E.; KAELI, D. R. Quantifying Load Imbalance on Virtualized Enterprise Servers. In: FIRST JOINT WOSP/SIPEW INTERNATIONAL CONFERENCE ON PERFORMANCE ENGINEERING, New York, NY, USA. **Proceedings...** ACM, 2010. p.235–242. (WOSP/SIPEW '10).

BORST, W. N. **Construction of Engineering Ontologies for Knowledge Sharing and Reuse**. 1997. Tese (Doutorado em Ciência da Computação) — , Enschede.

BRAY, T. et al. **Extensible Markup Language (XML) 1.0 (Second Edition)**. Acessado em Fevereiro/2014, http://www.w3.org/TR/2000/REC-xml-20001006.

CHAMBERLIN, D. D.; BOYCE, R. F. SEQUEL: a structured english query language. In: ACM SIGFIDET (NOW SIGMOD) WORKSHOP ON DATA DESCRIPTION, ACCESS AND CONTROL, 1974., New York, NY, USA. **Proceedings...** ACM, 1974. p.249–264. (SIGFIDET '74).

CLARK, C. et al. Live Migration of Virtual Machines. In: ND CONFERENCE ON SYMPOSIUM ON NETWORKED SYSTEMS DESIGN & IMPLEMENTATION - VOLUME 2, 2., Berkeley, CA, USA. **Proceedings...** USENIX Association, 2005. p.273–286. (NSDI'05).

COOPER, M. **Advanced Bash-Scripting Guide - An in-depth exploration of the art of shell scripting**. Acessado em Março/2014, http://tldp.org/LDP/abs/html/index.html.

COP. **SQWRL**. Acessado em Novembro/2013, http://protege.cim3.net/cgi-bin/wiki.pl?WikiHomePage.

DODDS, L. **Introducing SPARQL**: querying the semantic web. Acessado em Março/2014, http://www.xml.com/lpt/a/2005/11/16/introducing-sparql-querying-semantic-web-tutorial.html.

FENSEL, D. **Spinning the Semantic Web**: bringing the world wide web to its full potential. [S.l.]: Mit Press, 2005.

FU, S. Failure-Aware Construction and Reconfiguration of Distributed Virtual Machines for High Availability Computing. In: IEEE/ACM INTERNATIONAL SYMPOSIUM ON CLUSTER COMPUTING AND THE GRID, 2009., Washington, DC, USA. **Proceedings...** IEEE Computer Society, 2009. p.372–379. (CCGRID '09).

GOLDBERG, R. P. Survey of Virtual Machine Research. **Computer**, Los Alamitos, CA, USA, v.7, n.9, p.34–45, Sept. 1974.

GUARINO, N. **Formal Ontology in Information Systems**: proceedings of the 1st international conference june 6-8, 1998, trento, italy. 1st.ed. Amsterdam, The Netherlands, The Netherlands: IOS Press, 1998.

HAN, T.; SIM, K. M. An ontology-enhanced cloud service discovery system. In: INTERNATIONAL MULTICONFERENCE OF ENGINEERS AND COMPUTER SCIENTISTS. **Proceedings...** [S.l.: s.n.], 2010. v.1, p.17–19.

HILLBRECHT, R.; BONA, L. C. E. d. A SNMP-Based Virtual Machines Management Interface. In: IEEE/ACM FIFTH INTERNATIONAL CONFERENCE ON UTILITY AND CLOUD COMPUTING, 2012., Washington, DC, USA. **Proceedings...** IEEE Computer Society, 2012. p.279–286. (UCC '12).

HORRIDGE, M. et al. A Practical Guide To Building OWL Ontologies Using Protégé-OWL Plugin and CO-ODE Tools Edition1.0. **The University Of Manchester**, [S.l.], 2004.

HORRIDGE, M. et al. A Practical Guide To Building OWL Ontologies Using Protégé 4 and CO-ODE Tools Edition1. 2. **The University Of Manchester**, [S.l.], 2009.

HORROCKS, I. et al. **SWRL**: a semantic web rule language combining owl and ruleml. Acessado em Abril/2014, http://www.w3.org/Submission/SWRL/.

HUANG, M. On the concept of geographic ontology-from the viewpoints of philosophy ontology, information ontology and spatial ontology. In: INTERNATIONAL CONFERENCE ON GEOINFORMATICS, Beijing. **Anais...** IEEE Computer Society, 2010. p.1–5.

JAVA. **Java SE Documentation**. Acessado em Março/2014, http://www.oracle.com/technetwork/java/javase/documentation/index.html.

JENA. **Jena Ontology API**. Acessado em Março/2014, http://jena.apache.org/documentation/ontology/.

JONES, M. T. **Anatomia da biblioteca de virtualização libvirt**. Acessado em Fereiro/2014, http://www.ibm.com/developerworks/br/linux/library/l-libvirt/.

JONES, M. T. **Gerenciando VMs com o Virtual Machine Manager**. Acessado em Fereiro/2014, http://www.ibm.com/developerworks/br/cloud/library/cl-managingvms/.

KAPIL, D.; PILLI, E.; JOSHI, R. Live virtual machine migration techniques: survey and research challenges. In: ADVANCE COMPUTING CONFERENCE (IACC), Ghaziabad. **Anais...** IEEE Computer Society, 2013. p.963–969.

KHANNA, G. et al. Application performance management in virtualized server environments. In: NETWORK OPERATIONS AND MANAGEMENT SYMPOSIUM, 2006. NOMS 2006. 10TH IEEE/IFIP. **Anais...** [S.l.: s.n.], 2006. p.373–381.

KNUBLAUCH, H. Ontology-Driven Software Development in the Context of the Semantic Web: an example scenario with protege/owl. In: INTERNATIONAL WORKSHOP ON THE MODEL-DRIVEN SEMANTIC WEB (MDSW2004), 1. **Anais...** [S.l.: s.n.], 2004.

KOPYTOV, A. **SysBench**: a system performance benchmark. Acessado em Março/2014, https://launchpad.net/sysbench.

LASSILA, O. et al. **Resource Description Framework (RDF) Model and Syntax Specification**. Acessado em Fevereiro/2014, http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.44.6030.

LEELIPUSHPAM, G. J.; SHARMILA, J. Live VM migration techniques in cloud environment – A survey. In: CONFERENCE ON INFORMATION & COMMUNICATION TECHNOLOGIES (ICT). **Anais...** IEEE Computer Society, 2013. p.408–413.

LI, Y.; LI, W.; JIANG, C. A survey of virtual machine system: current technology and future trends. In: ELECTRONIC COMMERCE AND SECURITY (ISECS), 2010 THIRD INTERNATIONAL SYMPOSIUM ON. **Anais...** [S.l.: s.n.], 2010. p.332–336.

LI, Z. et al. A Live Migration Strategy for Virtual Machine Based on Performance Predicting. In: INTERNATIONAL CONFERENCE ON COMPUTER SCIENCE AND SERVICE SYSTEM,

2012., Washington, DC, USA. **Proceedings...** IEEE Computer Society, 2012. p.72–76. (CSSS '12).

LIBRELOTTO, G. R. et al. Uma Ontologia aplicada a um Ambiente Pervasivo Hospitalar. **8a Conferência da associação portuguesa de sistemas de informação**, [S.l.], 2008.

LIBVIRT. **The virtualization API**. Acessado em Março/2014, http://libvirt.org/.

LUBUNTU. **Lubuntu**: lightweight, fast, easier. Acessado em Março/2014, http://lubuntu.net/.

MARAN, V. et al. Uma definição ontológica de elementos de contexto relevantes na adaptação de documentos em ambientes hospitalares pervasivos. **Revista Brasileira de Computação Aplicada**, [S.l.], v.5, n.1, p.26–41, abr 2013.

MCGUINNESS, D. L.; HARMELEN, F. van. **OWL Web Ontology Language**. Acessado em Abril/2014, http://www.w3.org/TR/2004/REC-owl-features-20040210/.

MELL, P.; GRANCE, T. The NIST definition of cloud computing. **National Institute of Standards and Technology**, [S.l.], v.53, n.6, p.50, 2009.

NOY, N. F.; MCGUINNESS, D. L. et al. **Ontology development 101**: a guide to creating your first ontology. [S.l.]: Stanford knowledge systems laboratory technical report KSL-01-05 and Stanford medical informatics technical report SMI-2001-0880, 2001.

O'CONNOR, M.; DAS, A. SQWRL: a query language for owl. In: OWLED. **Anais...** [S.l.: s.n.], 2009. v.529.

OPENSSH. **FREE version of the SSH**. Acessado em Março/2014, http://www.openssh.com/.

PALLIS, G. Cloud Computing: the new frontier of internet computing. In: IEEE INTERNET COMPUTING. **Anais...** IEEE, 2010. v.14, p.70–73.

POLLOCK, J. T. **Semantic web for dummies**. [S.l.]: John Wiley & Sons, 2009.

SAGANA, C.; GEETHA, M.; SUGANTHE, R. Performance enhancement in live migration for cloud computing environments. In: INTERNATIONAL CONFERENCE ON INFORMATION COMMUNICATION AND EMBEDDED SYSTEMS (ICICES), 2013., Chennai. **Anais...** IEEE Computer Society, 2013. p.361–366.

SANDBERG, R. et al. Design and implementation of the Sun network filesystem. In: SUMMER USENIX CONFERENCE. **Proceedings...** [S.l.: s.n.], 1985. p.119–130.

SEABORNE, A.; PRUD'HOMMEAUX, E. **SPARQL Query Language for RDFb**. Acessado em Março/2014, http://www.w3.org/TR/rdf-sparql-query/.

SEGARAN, T. et al. **Programming the Semantic Web**. 1st.ed. [S.l.]: O'Reilly Media, Inc., 2009.

SQLITE. **SQLite About**. Acessado em Abril/2015, https://www.sqlite.org/about.html.

STRUNK, A.; DARGIE, W. Does Live Migration of Virtual Machines Cost Energy? In: INTERNATIONAL CONFERENCE ON ADVANCED INFORMATION NETWORKING AND APPLICATIONS, Barcelona. **Anais...** IEEE Computer Society, 2013. p.514–521.

STUDER, R.; BENJAMINS, V. R.; FENSEL, D. Knowledge Engineering: principles and methods. **Data Knowl. Eng.**, Amsterdam, The Netherlands, The Netherlands, v.25, n.1-2, p.161–197, Mar. 1998.

WEI, B.; LIN, C.; KONG, X. Energy optimized modeling for live migration in virtual data center. In: INTERNATIONAL CONFERENCE ON COMPUTER SCIENCE AND NETWORK TECHNOLOGY (ICCSNT) (VOLUME:4), Harbin. **Anais...** IEEE Computer Society, 2011. p.2311–2315. (ICCSNT '11).

WOOD, T. et al. Black-box and Gray-box Strategies for Virtual Machine Migration. In: USENIX CONFERENCE ON NETWORKED SYSTEMS DESIGN & IMPLEMENTATION, 4., Berkeley, CA, USA. **Proceedings...** USENIX Association, 2007. p.17–17. (NSDI'07).

YE, K. et al. VC-Migration: live migration of virtual clusters in the cloud. In: ACM/IEEE 13TH INTERNATIONAL CONFERENCE ON GRID COMPUTING, 2012., Washington, DC, USA. **Proceedings...** IEEE Computer Society, 2012. p.209–218. (GRID '12).

YOUSEFF, L.; BUTRICO, M.; DA SILVA, D. Toward a unified ontology of cloud computing. In: GRID COMPUTING ENVIRONMENTS WORKSHOP, 2008. GCE'08. **Anais...** [S.l.: s.n.], 2008. p.1–10.

ZHANG, W. et al. LVMCI: efficient and effective vm live migration selection scheme in virtualized data centers. In: INTERNATIONAL CONFERENCE ON PARALLEL AND DISTRIBU-

TED SYSTEMS (ICPADS), Singapore. **Anais...** IEEE Computer Society, 2012. p.368–375. (ICPADS '12).

APÊNDICES

APÊNDICE A – INSTALAÇÃO DO AMBIENTE VIRTUALIZADO

No primeiro momento é preciso verificar se a máquina onde o ambiente será configuração possui suporte a aceleração de hardware virtualizado disponibilizado pelo processador, em caso de negativo o ambiente pode ser usado para virtualização, no entanto haverá uma perda de desempenho das instancias de máquinas virtuais neste ambiente.

A.1 Verificando o suporte a aceleração

Para verificar este suporte é preciso instalar a ferramenta cpu-checker através do comando sudo apt-get install cpu-checker, posteriormente o comando kvm-ok espera-se retornar a seguinte mensagem:

```
INFO: /dev/kvm exists
KVM acceleration can be used
```

Se a mensagem retornada for igual a mensagem abaixo, então a máquina não apresenta suporte a aceleração.

```
INFO: Your CPU does not support KVM extensions
KVM acceleration can NOT be used
```

Em alguns casos o suporte existe mas é necessário um comando extra para habilitar a aceleração de hardware para o KVM. Estes comandos devem ser executados em modo root:

```
sudo modprobe kvm_intel
sudo /usr/sbin/kvm-ok
```

A.2 Instalação da Libvirt e outras bibliotecas

Assim, segue-se adiante com a instalação. A partir de então foi instalado o KVM através do comando *sudo apt-get install qemu-kvm libvirt-bin ubuntu-vm-builder bridge-utils*. Juntamente com este comando é instalado a API *libvirt*, *libvirtd* e o *virsh*. Abaixo é exemplificado algumas das bibliotecas instaladas pelo comando:

- *libvirt-bin*: fornece a libvirtd que é preciso para administrar instancias de KVM e QEMU através da libvirt;

- *qemu-kvm*: é o backend;
- *ubuntu-vm-builder*: poderosa ferramenta de linha de comando para a construção de máquinas virtuais;
- *bridge-utils*: fornece uma bridge entre a sua rede es a máquinas virtuais.

Em seguida é necessário adicionar o usuário nos novos grupos criados com o comando anterior, ao grupo kvm e ao grupo libvirtd, e em seguida o sistema operacional foi reiniciado para que o usuário passe a pertencer, de fato, aos grupos, os comando estão listados abaixo:

```
sudo adduser `id -un` kvm
sudo adduser `id -un` libvirtd
```

Enfim para testar se a instalação ocorreu satisfatoriamente, utiliza-se e comando

```
virsh -c qemu:///system list
```

Como saída espera-se que a coluna Id Name e State apareçam no terminal. No entanto não irá aparecer nenhuma máquina virtual pois ainda não foram criadas e instanciadas. Então para que seja possível a criação de máquinas virtuais foi utilizado duas ferramentas, uma em linha de comando e outra com interface gráfica.

A.3 Criação de máquinas virtuais

A ferramenta virt-install foi desenvolvida pela Red Hat para facilitar a criação de maquinas virtuais, para instalar esta ferramenta no ubuntu utiliza-se o comando: sudo apt-get install virtinst

Através desta ferramenta conecta-se ao QEMU e com vários parâmetros é configurada a máquina virtual. Na lista abaixo é apresentado a lista de parâmetros que foi utilizado para criar uma máquina virtual:

- -n: Nome da máquina virtual
- -r: Memória que será alocada para a instancia da máquina virtual
- –vcpus: Número de CPUs virtuais
- -path: Caminho onde será gerado o disco rígido

- size: Tamanho do disco rígido em GB
- -c: Define que a instalação será por CD-ROM ou uma imagem ISO e informa o seu caminho
- –os-type: Otimiza a configuração do convidado para um tipo de sistema operacional
- –accelerate: Otimiza a instalação para o QEMU
- –network: Configura a rede do convidado informando o tipo de conexão com o host e o nome do dispositivo de rede.
- –noautoconsole: Não tenta automaticamente se conectar ao console do convidado
- –hvm: Informa que deve ser realizada a virtualização completa
- –disk: Especifica mídia para usar como armazenamento para os convidados, com várias opções como caminho da imagem e caminho do disco.

No comando abaixo, através dos parâmetros listados acima, foi criado uma máquina virtual Ubuntu, com 1GB de RAM, 12 GB de disco rígido, através da uma ISO baixada do site oficial do Ubuntu:

```
sudo virt-install --connect qemu:///system -n vm11 -r 1024 --vcpus=2
--disk path=/var/lib/libvirt/images/vm11.img,size=12 -c
/var/lib/libvirt/images/ubuntu-12.04.3-desktop-amd64.iso --vnc
--noautoconsole --os-type linux --accelerate --network=bridge:virbr0
-hvm
```

A outra ferramenta utilizada foi a *virt-manager*. Diferentemente da *virt-install* ela possui interface gráfica facilitando ainda mais o gerenciamento das maquinas virtuais. Para instalar esta ferramenta foi utilizado o comando:

```
sudo apt-get install virt-manager
```

Após o termino da instalação o virt-manager pode ser acessado através da lista de programas do Ubuntu ou no terminal pelo comando sudo virt-manager. A Figura 4 apresenta a tela inicial do programa.

A.4 Configuração SSH

No sistema operacional Ubuntu já é disponibilizado um cliente SSH. Mas para dar acesso remoto a máquina é necessário que a mesma tenha um serviço de SSH rodando e ouvindo a porta 22. Então foi executado o comando abaixo, nas duas máquinas citadas no item 4.1:

```
sudo apt-get install openssh-server
```

Após este comando a máquina está disponível para acesso remoto através do SSH. No entanto para os testes foi necessário criar uma chave e utilizar autenticação por chave ao invés de utilizar senha. Isto para que se tenha mais segurança no acesso ao servidor e se torna mais fácil para fazer a autenticação na aplicação cliente. Não necessitando informar a senha a cada acesso remoto.

Então, é criado a chave pública e privada através dos comandos abaixo, utilizando a flag *–b* para criptografar as chaves em 4096 *bits*:

```
mkdir ~/.ssh
chmod 700 ~/.ssh
ssh-keygen -t rsa -b 4096
```

Após a criação das chaves é preciso copia-las para os servidores aos quais serão realizados os acessos através da aplicação. O comando utilizado abaixo copia os arquivos para o servidor informado no parâmetro <host>:

```
ssh-copy-id <username>@<host>
```

A partir do envio das chaves não será mais necessário informar a senha para acesso ao servidor que tiver com a chave. Desde que a máquina que está pedindo o acesso possua a chave privada e que o servidor possua a chave pública desta máquina.

A.5 Servidor NFS

O servidor NFS para compartilhamento de arquivo foi configurado em uma terceira máquina para armazenar a imagem das máquinas virtuais. As duas máquinas servidores de máquinas virtuais devem acessar o caminho das imagens que por padrão são armazenadas no

seguinte diretório: */var/lib/virt/images*. Então, para instalar o servidor NFS utiliza-se o seguinte comando:

```
sudo apt-get install nfs-kernel-server
```

A configuração do servidor é realizada editando o arquivo */etc/exports* onde são definidas os diretórios compartilhados e as regras de acesso ao servidor. Nesta máquina também foi instalado o ambiente de virtualização e compartilhado, então, o diretório images da *libvirt*. Então a linha adicionada no arquivo */etc/exports/* foi o que segue abaixo, consecutivamente o comando para inicializar o servidor NFS:

```
/var/lib/virt/images  *(ro,sync,no_root_squash)
sudo /etc/init.d/nfs-kernel-server start
```

Nas máquinas que deverão ”enxergar” o diretório uma linha é adicionado no arquivo de inicialização */etc/fstab* para que na inicialização do servidor o diretório seja montado ou simplesmente montar o diretório compartilhado. Foi montado o diretório com o seguinte comando, onde <host> é a identificação do servidor NFS, e consecutivamente o comando para adicionar na inicialização do sistema:

```
sudo mount <host>:/var/lib/virt/images /var/lib/virt/images
<host>:/var/lib/virt/images /var/lib/virt/images nfs rw,hard,intr 0 0
```