UNIVERSIDADE FEDERAL DE SANTA CATARINA
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA

Rafael Brundo Uriarte

UM ARCABOUÇO DE MONITORAMENTO E AUTO-PROTEÇÃO PARA NUVENS PRIVADAS

Florianópolis(SC)

2012

Rafael Brundo Uriarte

UM ARCABOUÇO DE MONITORAMENTO E AUTO-PROTEÇÃO PARA NUVENS PRIVADAS

Dissertação submetida ao Programa de Pós-Graduação em Ciência da Computação para a obtenção do Grau de Mestre em Ciência da Computação.
Orientador: Prof. Dr. Carlos Becker Westphall

Florianópolis(SC)

2012

Catalogação na fonte elaborada pela biblioteca da
Universidade Federal de Santa Catarina

A ficha catalográfica é confeccionada pela Biblioteca Central.

Tamanho: 7cm x 12 cm

Fonte: Times New Roman 9,5

Maiores informações em:
http://www.bu.ufsc.br/design/Catalogacao.html

Rafael Brundo Uriarte

UM ARCABOUÇO DE MONITORAMENTO E AUTO-PROTEÇÃO PARA NUVENS PRIVADAS

Esta Dissertação foi julgada aprovada para a obtenção do Título de "Mestre em Ciência da Computação", e aprovada em sua forma final pelo Programa de Pós-Graduação em Ciência da Computação.

Florianópolis(SC), 20 de agosto 2012.

Prof. Dr. Ronaldo dos Santos Mello
Coordenador

Prof. Dr. Carlos Becker Westphall
Orientador

Banca Examinadora:

Prof. Dr. Carlos Becker Westphall
Presidente

Prof. Dr. Edmundo Roberto Mauro Madeira

Prof. Dr. Mario Antonio Ribeiro Dantas

Profa. Dra. Carla Merkle Westphall

À minha irmã, que sempre esteve comigo.

AGRADECIMENTOS

Quando alguém ler esta dissertação talvez não perceberá, apesar de provavelmente possuir vários problemas e erros, o seu maior erro está já na primeira página. Esta dissertação não foi feita por uma pessoa, mas sim por todos que já passaram ou fazem parte da minha vida. Assim, agradeço a todos e espero que cada um saiba seu papel no processo de superação de mais esta fase.

Una volta che avrete imparato a Volare, camminerete sulla terra guardando il cielo perchè è là che siete stati ed è là che vorrete tornare.

Leonardo da Vinci

RESUMO

Um dos novos desafios do paradigma de computação em nuvem é a administração efetiva destes sistemas e recursos devido, a sua heterogeneidade, escalabilidade e a falta de ferramentas adequadas. Consumo de energia, desempenho, provisão de recursos e segurança são somente alguns fatores relevantes no gerenciamento. Neste âmbito, a computação autônoma visa facilitar e automatizar este gerenciamento (gerenciamento sem intervenção humana) através de quatro propriedades: auto-otimização, auto-cura, auto-configuração e auto-proteção. O uso de computação autônoma em computação em nuvem, principalmente focando em nuvens privadas, foi pouco explorado até o momento. Este trabalho procura dar um dos primeiros passos para portar os princípios de computação autônoma para nuvens privadas com a definição de uma arquitetura para o monitoramento deste tipo de nuvem, uma das bases da computação autônoma. Esta também propõe o uso simplificado de umas das propriedades, a auto-proteção que se beneficia da base de monitoramento desenvolvida. Para validar esta proposta foi desenvolvido um arcabouço de código aberto e gratuito denominado PANOPTES. O Panoptes usa o paradigma multi-agente para o monitoramento efetivo, distribuído e escalável dos recursos físicos e virtuais da nuvem e, assim, fornece a base para a tomada correta de decisões. A interação com o administrador de sistemas e a sincronia com os objetivos da organização ocorre através da definição de políticas de alto nível. Dentre outras, as vantagens deste arcabouço são a facilidade de estender e adaptar o arcabouço para as próprias necessidades e a compatibilidade com os padrões em vigor. No decorrer do trabalho, os paradigmas supracitados e os pilares deste foram documentados para facilitar a sua compreensão, além de justificar as escolhas de cada parte importante na arquitetura, relacionar os padrões mais relevantes no desenvolvimento e uso destes. Por fim, um caso de uso é apresentado para validação da proposta.

Palavras-chave: Computação em Nuvem, Nuvens Privadas, Computação Autônoma, Sistemas Multi-Agente

ABSTRACT

A major challenge of cloud computing is the effective management of cloud systems and resources due to its heterogeneity, scalability and lack of suitable tools. Energy consume, performance, provisioning and security are only some of the relevant factors for the management of these systems. In this area, autonomic computing aims to facilitate and automate the management (without human intervention) through four proprieties: self-configuration, self-healing, self-optimization and self-protection. The use of autonomic computing on cloud computing, mainly focusing in private clouds has not been deep explored and is still on its infancy. In this work, we give one of the first steps to bring the idea of autonomic computing to private clouds with the definition of a monitoring architecture, which is one of the base for this technology and also we propose the use of a simplistic auto-protection solutions using these definitions. To validate this proposal an open source framework named PANOPTES was developed. It uses the multi-agent paradigm for an effective, distributed and scalable monitoring of physical and virtual resources of a cloud, thus providing a solid support for the decision taking. The interaction with the cloud administrator and the synchronization with the aims of the organization is through the definition of high level policies. Among others, the features of the framework are that it is easily extensible, it is easy to adapt it to specific needs and it is compatible with standards. Through the work, the cited paradigms and the pillars for these paradigms are explained and, furthermore, the choices made on the architecture and framework are justified and we list the most used standards related to the development of these. Finally, an use case is deployed to validade the proposal and is described

Keywords: Cloud Computing, Private Cloud, Autonomic Computing, Multi-Agent System

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

SUMÁRIO

1 INTRODUÇÃO

“Nel mezzo del cammin di nostra vita mi ritrovai per una selva oscura che’ la diritta via era smarrita.”

– Dante Alighieri(1265–1321).

> "The information technology industry loves to prove the impossible possible. We obliterate barriers and set records with astonishing regularity. But now we face a problem springing from the very core of our success and too few of us are focused on solving it. More than any other I/T problem, this one if it remains unsolved will actually prevent us from moving to the next era of computing. The obstacle is complexity. Dealing with it is the single most important challenge facing the I/T industry."[1] (HORN, 2001)

Computação em nuvem vem deixando de ser considerada somente uma tendência para se tornar uma realidade que promete mudar o modo que a computação é vista e usada. Este novo paradigma procura liberar o usuário dos problemas relacionados com as camadas inferiores da sua arquitetura e concentrar-se somente no seu próprio objetivo, seja ele de desenvolvimento, expansão de recursos ou como usuário de software.

Uma característica relevante para computação é nuvem é a abstração da complexidade, porém esta traz consigo vários problemas de integração e gerenciamento. Isto acontece pois os recursos disponíveis e existentes, são heterogêneos e precisam ser integrados, gerenciados e fornecidos como se fossem homogêneos. Além das dificuldades técnicas, esta complexidade exige mão de obra altamente capacitada.

[1] A indústria da tecnologia da informação adora provar o impossível possível. Esquecemos barreiras e quebramos recordes com uma regularidade impressionante mas agora estamos enfrentando um problema oriundo do núcleo do nosso sucesso e poucos de nós estão focados na sua resolução. Mais do que outros problema de TI, este continua sem solução e vai dificultar a transição para a nova era de computação. O obstaculo é a complexidade. Lidar com isto é o desafio atual mais importante da indústria da TI. (Tradução Própria)

Além dos problemas e dificuldades citados existe a questão econômica, segundo um estudo realizado pela IBM (MURCH, 2004) para cada dólar gasto em infraestrutura dez outros são gastos no gerenciamento desta. Os paradigmas atuais exigem mão de obra qualificada (relativamente escassa no mercado) e de alto custo. O crescimento destas redes e o aumento da complexidade previstos na computação em nuvem requerem o aumento exponencial desta mão de obra (deve-se administrar também a integração dos recursos). Se desconsiderarmos esta alternativa e continuarmos com o paradigma atual (com a maioria do trabalho efetuado por seres humanos), os custos tendem a crescer descontroladamente e estes sistemas estariam cada vez mais suscetíveis a erros. Uma possível solução para se lidar com este problema provem da computação autônoma: O auto-gerenciamento.

Um sistema autônomo é um sistema computacional capaz de se autogerenciar a partir de um conjunto de objetivos definidos pelos administradores do sistema, i.e., é capaz de regular os próprios parâmetros funcionais para otimizar o sistema mantendo as prioridades e a segurança deste. A inspiração deste conceito advêm do sistema nervoso autônomo humano, sistema que regula e controla funções como respiração, circulação, controle de temperatura e digestão. Estas funções são reguladas de forma autônoma (ou semi) e se adaptam as necessidades do ambiente ou do próprio corpo, um exemplo seria de um indivíduo que entra em um lugar frio, seu sistema nervoso autônomo tenta impedir a queda de temperatura corporal e este se arrepia e/ou começa a tremer para gerar calor. Os sistemas autônomos, assim como seu sistema análogo, procura manter o sistema funcionando de modo otimizado e correto de acordo com políticas definidas pelos administradores mas com pouca ou nenhuma intervenção destes.

Os seus quatro aspectos mais importantes são: auto-otimização, auto-proteção, auto-configuração e auto-cura (IBM White Paper, 2005) (MCCANN; HUEBSCHER, 2004) (MURCH, 2004) (NAMI; BERTELS, 2007). Apesar de um número considerável de arquiteturas e soluções para computação autônoma terem sido propostas e definidas, pouco foi implementado/-testado, e este é um dos principais desafios apresentados em (HUEBSCHER; MCCANN, 2008). Outro considerável desafio na área reside na dificuldade (oriunda, principalmente, da falta de bases concretas da engenharia de softwares específicos) de desenvolver sistemas autônomos sólidos (MULLER; KIENLE; STEGE, 2009).

Para o desenvolvimento dos sistemas autônomos se faz necessário a coleta de informações sobre o ambiente e sobre o próprio sistema ao longo do tempo. Esta é uma propriedade que se destaca pois é a base para a análise e tomada de decisões destes sistemas. Porém, a quantidade de informação relevante para esta tomada de decisão é considerável e deve ser adquirida com

o mínimo de impacto possível na performance do sistema. O monitoramento serve de base para todas as outras quatro propriedades.

Um dos quatro pilares do auto-gerenciamento, a auto-proteção, tem como principal objetivo se configurar e adaptar para atingir e maximizar propriedades como segurança, privacidade e proteção de dados(MCCANN; HUEBSCHER, 2004). Esta característica tem especial relevância no contexto de computação em nuvem. Pesquisas como (Trend Micro Inc., 2011) (Colt Technology Services, 2011) (International Data Corporation, 2009) relatam que a falta de segurança é um dos principais fatores que impedem ou retardam a adoção de computação em nuvem pois, parte dos dados não está diretamente sobre a supervisão do usuário final. A figura 1 ilustra essa preocupação com segurança.

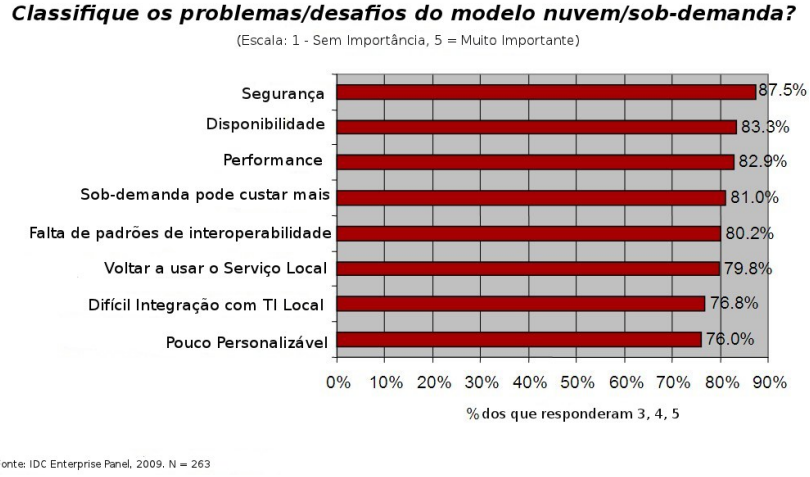

Figura 1: Pesquisa sobre os problemas e desafios para a adoção do paradigma de computação em nuvem (Traduzido de (International Data Corporation, 2009))

A grande maioria das soluções e estudos relacionados a computação em nuvem têm focado principalmente em soluções de nuvens públicas e pouca atenção tem sido dirigida a nuvens privadas. Estas possuem múltiplas utilidades, grandes corporações e governos podem ter um retorno econômico relevante flexibilizando e disponibilizando recursos para seus usuários e mantendo o controle na organização. Outro modo de se beneficiar de nuvens privadas é utilizar como um primeiro passo para a migração parcial (com nuvens híbridas) ou total (nuvens públicas) aproveitando o melhor dos dois

mundos.

Considerando estes problemas e limitações, este trabalho procura facilitar e simplificar a tarefa de gerenciamento, primeiramente, fornecendo e explicando alguns dos conceitos e ideias mais importante relacionados a estes paradigmas e, posteriormente, especificando uma arquitetura para atacar o problema com soluções específicas para nuvens privadas. A arquitetura definida ataca primeiramente o problema de monitoramento nestes sistemas e posteriormente uma definição simplificada da arquitetura de proteção. O sistema em questão foi desenhado e construído sob o paradigma de sistemas multi-agentes, um conceito reconhecidamente eficaz para minimizar a complexidade, controlar a heterogeneidade mantendo a escalabilidade dos seus componentes, sempre levando em consideração os conceitos definidos em 1.1.

Com o intuito de verificar a exequibilidade desta arquitetura foi implementado um arcabouço denominado Panoptes [2]. Este arcabouço foi disponibilizado em código aberto e gratuitamente e, apesar de não contar com a implementação de alguns padrões e melhores práticas aqui citadas, serve como primeiro passo em direção à automação de nuvens privadas.

1.1 OBJETIVOS

Esta dissertação procura avançar o estado da arte sobre um problema crescente que atinge e limita a escalabilidade da computação em nuvem privada: a custosa e cada vez mais complicada tarefa de gerenciamento deste sistema. A arquitetura aqui apresentada envolve o uso do conceito de computação autônoma. Apesar de não ser um conceito novo, poucos resultados práticos foram apresentados na literatura assim procura-se desenvolver um arcabouço capaz de facilitar o desenvolvimento de sistemas deste tipo para nuvens privadas levando em conta as suas particularidades como heterogeneidade e elasticidade.

Além disso, este trabalho tem como objetivo disponibilizar um arcabouço de fácil adaptação para as necessidades específicas de seus usuários e para validar a arquitetura supracitada.

Devido a restrições de escopo este trabalho foca principalmente no monitoramento dos recursos e na propriedade de auto-proteção (dos sistemas autônomos) e nos componente necessários para o funcionamento correto destas funções.

[2] O nome referencia Argus Panoptes que na mitologia grega era um gigante com cem olhos. Na lenda ele era um ótimo sentinela pois quando dormia sempre somente alguns olhos se fechavam.

1.1.1 Objetivo Específico

Com esse trabalho pretende-se:

(a) Apresentar os conceitos de computação autônoma, computação em nuvem e tecnologias relacionadas a estas;

(b) Resumidamente definir e especificar as principais diferenças de computação em nuvem para outros conceitos como grades computacionais;

(c) Especificar um arcabouço baseado em sistemas multi-agentes capaz de facilitar o desenvolvimento de sistemas autônomos para nuvens privadas, visando inicialmente a parte de monitoramento;

(d) Especificar uma arquitetura simplista para a propriedade de auto-proteção dos sistemas autônomos utilizando o sistema de monitoramento especificado.

(e) Desenvolver um protótipo destes sistemas;

(f) Testar a integração do arcabouço com os módulos de expansão, através do desenvolvimento de um módulo com funções específicas para permitir a fácil adição de mecanismos de proteção.

1.1.2 Organização do Trabalho

Esta dissertação está organizada da seguinte maneira:

- **Cap. 1** - Apresenta a introdução ao problema de pesquisa e o contexto da dissertação, colocando suas questões e objetivos, bem como, a definição da proposta e a organização do estudo;
- **Cap. 2** - Discorre sobre os principais conceitos, características e ferramentas relacionadas com o paradigma de computação em nuvem situando o leitor sobre a sua abrangência e situação atual.
- **Cap. 3** - Descreve as abordagens de computação autônoma e suas peculiaridades. Também, é apresentado brevemente os conceitos e usos de sistemas multi-agentes nesta área;
- **Cap. 4** - Explica os trabalhos correlatos a paradigmas e conceitos usados na nesta dissertação bem como padrões e definições que orientam o desenvolvimento nessas áreas. Por fim, são descritas as principais diferenças entre data centres tradicionais, nuvens públicas e nuvens privadas;

- **Cap. 5** - Este capítulo propõe a arquitetura e um arcabouço que utiliza dos conceitos definidos nos capítulos anteriores, trabalhos correlatos e abordagens próprias para levar computação autônoma para ambiente de nuvens privadas. Também descreve um estudo de caso e descreve brevemente a avaliação experimental realizada;

- **Cap. 6** - Apresenta as conclusões, contribuições decorrentes desta dissertação, limitações da pesquisa e planos para trabalhos futuros.

2 COMPUTAÇÃO EM NUVEM

"We are all in the gutter, but some of us are looking at the stars."
–Oscar Wilde (1854–1900).

2.1 TERMO

O termo *Cloud* ou nuvem provavelmente deriva dos diagramas de redes usados desde os anos 90 para designar a malha de circuitos das redes de telecomunicações. Primeiramente as empresas de telecomunicação ofereciam o serviço de circuitos ponto-a-ponto alugados. Com o desenvolvimento e avanço das tecnologias de comutação, ao invés do caro serviço de ponto-a-ponto, estas empresas começaram a disponibilizar serviços de VPN (*Virtual Private Network* ou Rede Privada Virtual) ou transmissão de dados, mantendo certa garantia de banda mas a um custo muito menor, fazendo a comutação entre seus circuitos redundantes e menos utilizados no momento. Assim estas empresas poderiam evitar congestionamento e, de certa forma, garantir a qualidade do serviço. Porém, como desvantagem nem as próprias fornecedoras do serviço saberiam o caminho exato que um pacote faria pela malha de circuitos até seu destino e, para generalizar e simplificar, o termo "nuvem"era utilizado para representar este caminho. Fazendo um paralelo com o seu significado na área de redes, o termo foi designado para esse novo modelo computacional, onde não se sabe (ou não interessa saber) exatamente a proveniência dos recursos computacionais utilizados.

2.2 HISTÓRICO

A história de computação em nuvem não tem um começo bem definido. Já em 1961 John MacCarthy sugeriu que poder computacional e até aplicações específicas poderiam ser entregues usando o modelo econômico de utilidade (o mesmo do fornecimento de água, luz e telefone). Desde então muitas tecnologias e paradigmas foram desenvolvidos mas, foi somente a

partir dos anos 90 que esta ideia começou a se desenvolver devido a popularização da internet e da banda larga.

A primeira definição acadêmica formal relacionada com as definições mais aceitas atualmente foi a publicada em (CHELLAPPA, 1997), que definiu computação em nuvem como:

> a computing paradigm where the boundaries of computing will be determined by economic rationale rather than technical limits[1]

Um dos primeiros passos importantes no âmbito comercial ocorreu em 1999, quando a empresa *SalesForce* (SALESFORCE, 2012) fundada por Marc Benioff, começou a oferecer aplicações comerciais através da internet. Estas soluções usavam o conceito de software "sob-demanda", ou seja, o que hoje conhecemos por SaaS (*Software as a Service* ou Software como Serviço). Com este serviço, a empresa teve ótimos resultados devido à flexibilidade e velocidade, abrindo as portas e atraindo atenção para este conceito. O próximo desenvolvimento marcante foi o *Amazon Web Services* (AMAZON, 2012), iniciado em 2002, que fornecia armazenamento, poder computacional e até mesmo o agenciamento de tarefas para seres humanos ou para serem resolvidas por inteligência artificial (seu funcionamento é similar ao de um mercado, se oferece uma tarefa com um pagamento definido e os desenvolvedores podem aceitá-las ou não). A disponibilização deste serviço ocorreu pois a própria empresa percebeu que menos de 10% da capacidade computacional do seu parque tecnológico era usada em 90% do tempo e o restante era somente usado para suprir picos de demanda esporádicos.

A própria *Amazon* lançou em 2006 o serviço EC2 (*Elastic Compute Cloud*), um serviço que permite aos indivíduos ou empresas alugar máquinas e rodar as suas próprias aplicações. Por este pioneirismo, a *Amazon* é considerada até hoje um ponto de referência e através de seus serviços conseguiu criar padrão não formal, mas considerado o padrão *de facto*.

Mais recentemente, grandes empresas como a *Google*, a *IBM* e várias universidades começaram a investir pesadamente em pesquisas sobre o tema. Este paradigma continua ainda atraindo grandes investimentos e possuí ofertas de diferentes serviços e soluções.

2.3 TECNOLOGIAS BASE

O paradigma, computação em nuvem pode ser visto também como um guarda-chuva que abriga várias outras tecnologias já consolidadas no mer-

[1] Um paradigma computacional, onde as fronteiras da computação serão determinadas por lógicas econômicas ao invés de limites técnicos (tradução própria).

cado e alia seus benefícios em direção ao fornecimento de recursos computacionais. Dentre as tecnologias que fazem parte podemos citar como exemplo: Virtualização, acordos de nível de serviço e computação utilitária. Nesta seção estas tecnologias serão conceituadas para o melhor entendimento de computação em nuvem.

2.3.1 Computação Utilitária

Computação utilitária é o modelo de negócios para o fornecimento de recursos computacionais como armazenamento ou processamento sob demanda, onde o pagamento do serviço é feito de acordo com a quantidade usada a preços preestabelecidos (WIKIPEDIA, 2011b). Em computação, muitos visionários acreditavam que a computação utilitária poderia virar realidade. Isto é, que os recursos computacionais poderiam ser explorados do mesmo modo que eletricidade, água e outros serviços que são fornecidos a população. Esta não é uma ideia recente, temos como exemplo Leonard Kleinrock, um dos cientistas chefes da ARPANET (uma versão inicial do que se tornaria a internet) que em 1969 fez a seguinte declaração:

> "As of now, computer networks are still in their infancy. But as they grow up and become more sophisticated, we will probably see the spread of computer utilities, which like present electric and telephone utilities, will service individual homes and offices around the country"(KLEINROCK, 2005) [2]

2.3.2 Acordo de Nível de Serviço

O acordo de nível de serviço, comumente chamado de SLA (do termo *Service Level Agreement*), é a parte de um contrato de serviços entre duas ou mais entidades no qual o nível da prestação do serviço é definido formalmente (WIKIPEDIA, 2011a). Nesta descrição formal podem ser incluídas várias informações, variando com a área e abrangência do contrato. Como exemplo podemos citar: o gerenciamento de problemas, garantias, medidas emergenciais, segurança, confidenciabilidade, tempo médio de reparo, tempo médio entre falhas, relatórios para monitoramento.

[2] As redes de computadores ainda estão na sua infância, mas, assim que crescerem e ficarem mais sofisticadas, provavelmente veremos a expansão da computação utilitária, que como energia elétrica e telefone, serão serviços fornecidos nas casas e escritórios pelo país. (Tradução própria)

O gerenciamento de SLA se dá em duas fases principais: a negociação do contrato, onde o cliente e o provedor acordam sobre garantias e preços; e a fase de monitoramento, na qual se verifica o cumprimento deste contrato. O acordo de nível de serviço se mostra essencial no apoio ao paradigma de computação em nuvem, pois como recursos são fornecidos por terceiros, um modo de controle e uma forma de garantia são exigidos.

2.3.3 Virtualização

Virtualização é um processo que abstrai logicamente o sistema operacional do hardware criando uma nova camada de software entre os mesmos, a qual fica responsável pela comunicação com o hardware e permite a execução simultânea de vários sistemas operacionais (chamados de máquinas virtuais ou MVs). Com isto, os recursos disponíveis são divididos entre estas máquinas virtuais. Esta camada é chamada de monitor de máquina virtual (*Virtual Machine Monitor* ou VMM) e pode, também, ser denominado "sistema operacional para sistemas operacionais" ou ainda "hipervisor". O uso da virtualização representa a ilusão de várias máquinas virtuais independentes, cada uma rodando uma instância de um sistema operacional virtualizado (SMITH; NAIR, 2005). A Figura 2 ilustra o conceito apresentado.

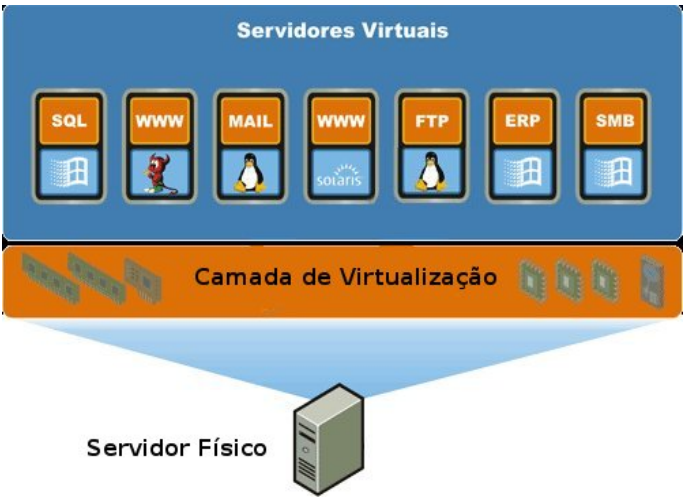

Figura 2: Camada de Virtualização

As MVs são estes sistemas operacionais que usam a simulação de uma

máquina física (camada VMM), isto é, uma réplica física da máquina real e os usuários têm a ilusão de que o sistema está disponível para seu uso exclusivo (SUGERMAN; VENKITACHALAM; LIM, 2001).

A virtualização vem sendo usada há mais de 40 anos. Na década de 70 a *IBM* já utilizava esta tecnologia em mainframes. O motivo da popularização do interesse pelo tema é devido ao grande poder de processamento e grande quantidade de memória disponível nos computadores atuais, assim esta técnica pode ser utilizada na grande maioria dos computadores presentes, tanto servidores quanto computadores pessoais. (LAUREANO, 2006).

2.3.4 Tipos de Virtualização

Atualmente, grande parte das ferramentas que permitem a virtualização possuem uma junção dos tipos de virtualização básicos. A seguir são apresentados os modelos clássicos para usar este conceito:

- Emulação de Hardware: É uma forma complexa de virtualização que reproduz o comportamento de um hardware (conjunto de instruções, estado de execução do processador, memória cache, ciclos de relógio), ou seja, as MVs se baseiam em hardwares simulados. Essa forma de virtualização é muito utilizada por desenvolvedores de hardware e *firmwares* pois não precisa de hardware real e os mesmos podem testar suas soluções. Um dos grandes pontos negativos é a baixa performance, podendo ser até 100 vezes mais lento que o hardware real(JONES, 2006).

- Virtualização Completa ou Total: Este modelo prevê a simulação de toda a estrutura de hardware de forma que o sistema hóspede trabalhe como se estivesse trabalhando diretamente no hardware. Nessa solução uma das vantagens é que o sistema operacional hospedeiro não precisa ser modificado. A simulação executada nessa técnica não é tão lenta quanto na executada na técnica de emulação, pois não é necessário representar os estados de execução do hardware, e sim, representar a memória principal, o conjunto de instruções do processador, interrupções, exceções e acesso aos dispositivos da máquina real.

- Paravirtualização: uma técnica de virtualização em que a máquina virtual não simula o hardware, mas apresenta uma API (*Application Programming Interface*) para as máquinas virtuais terem acesso ao hardware real. Essa técnica requer que o sistema operacional virtualizado seja explicitamente portado para permitir a sua execução. Também possui um desempenho melhor pois os *drivers* executados nas máquinas

virtuais paravirtualizadas são os *drivers* reais para os dispositivos físicos enquanto na virtualização completa estes *drivers* são emulados acima.

A tabela 1 contem alguns exemplos de aplicação e as respectivas técnicas usadas (a maioria destas ferramentas utiliza conceitos híbridos mas foi apresentado seu tipo base):

Tabela 1: Ferramentas de Virtualização

Projeto	**Tipos de Virtualização**
Bochs	Emulação de Hardware
QEMU	Emulação de Hardware
VMWARE	Virtualização Total
KVM	Virtualização Total
XEN	Paravirtualização

2.4 CLASSIFICAÇÕES

A computação em nuvem usa na maioria dos casos o modelo de computação utilitária definido na seção 2.3.1. Uma das definições importantes neste paradigma se refere aos recursos e serviços disponibilizados pelo provedor aos seus usuários. Nesta seção são descritas as principais categorias destes recursos. É importante notar que estas categorias podem ser utilizadas para o respectivo fornecimento sem utilizar computação em nuvem, isto é, são definições existentes que foram englobadas pela computação em nuvem. Além das definições abaixo, é apresentada na Figura 3 uma ilustração dos 3 principais modelos de serviço e os seus respectivos público-alvo:

- Software como Serviço (*Software as a Service* ou SaaS): É um modelo de distribuição de software em forma de um serviço ou prestação de serviço, onde um produtor de software desenvolve, opera, mantém e o disponibiliza para clientes remotos. Estes clientes usam e pagam por ele de acordo com as suas necessidades, isto é sob demanda. Usualmente este serviço é acessado através da internet e portanto pode ser acessado de qualquer lugar com acesso à mesma. Os usuários mais frequentes deste serviço são os clientes dos provedores, podemos exemplificar esta categoria com os serviços de e-mail na internet;

- Plataforma como Serviço (*Platform as a Service* ou PaaS): É o fornecimento de toda uma plataforma para desenvolvimento de software (principalmente desenvolvimento voltado para web), o que abstrai dos desenvolvedores os requisitos de hardware e possivelmente de outras camadas de softwares necessárias como por exemplo o banco de dados, o servidor web e o suporte a linguagem de programação agilizando e reduzindo a complexidade do desenvolvimento. É relevante considerar que o negócio dos provedores de PaaS se concentra em uma melhor experiência dos desenvolvedores, criando soluções para facilitar e acelerar o desenvolvimento. Neste caso os clientes do serviço são desenvolvedores de aplicações;

- Infraestrutura como Serviço (*Infrastructure as a Service* ou IaaS): É a entrega de processamento, armazenamento, rede e outros recursos computacionais fundamentais, onde o cliente pode usar software próprio. O cliente não controla a infraestrutura da nuvem mas retem o controle sobre o sistema operacional, armazenamento, uso de softwares e controle dos componentes da rede(Cloud Security Alliance, 2009). Basicamente o provedor entrega uma ou várias máquinas como infraestrutura para o cliente (normalmente virtuais para melhor uso dos recursos e compartilhamento destes). Uma das principais tecnologias usadas nesse modelo é a Virtualização. O público alvo da IaaS é principalmente administradores de rede, engenheiros de rede e administradores de setores de tenologia.

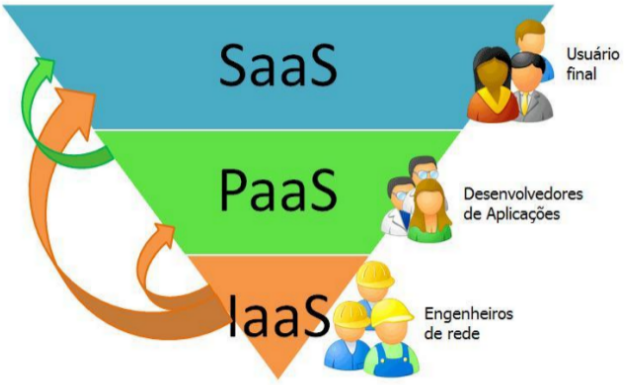

Figura 3: Modelos de serviço e seu respectivo público.

Além da categorização dos serviços oferecidos, também há uma categorização relacionada à abrangência e acesso à nuvem. A classificação geralmente citada é a de nuvem pública, nuvem privada e nuvem híbrida . Neste trabalho são descritas estas três principais e uma que vêm ganhando popularidade nos últimos tempos, a nuvem comunitária:

- Nuvem Privada (*Private Cloud*): O serviço da nuvem é acessado somente por uma organização, podendo ser administrada pela própria empresa ou por terceiros. Segundo (SOTOMAYOR et al., 2009), o objetivo principal de uma nuvem privada não é vender capacidade pela Internet através de interfaces acessíveis ao público em geral, mas sim dar aos usuários locais uma infraestrutura ágil e flexível para suportar cargas de trabalho de serviços dentro de seu próprio domínio administrativo. Este modelo teoricamente é o que promove menores riscos, em detrimento da sua natureza privada.

- Nuvem Comunitária (*Community Cloud*): Este modelo é baseado no compartilhamento de uma nuvem por diversas organizações (universidades ou indústrias), sendo esta suportada por uma comunidade específica que partilha de interesses semelhantes, tais como requisitos de segurança, missão, área de atuação. Também o mesmo se assemelha a nuvem privada em relação às politicas de acesso e segundo (MELL; GRANCE, 2009) este tipo de modelo de implantação pode existir localmente ou remotamente e pode ser administrado por alguma empresa da comunidade ou por terceiros.

- Nuvem Pública (*Public Cloud*): A infraestrutura da nuvem é disponibilizada ao público geral ou um grande grupo industrial (MELL; GRANCE, 2009) e é fornecida por uma organização que vende ou simplesmente disponibiliza o serviço. Este conceito proporciona às organizações economia de escala, já que compartilha recursos mas, por outro lado, limita a customização e aumenta a preocupação com a segurança dos dados, SLAs e políticas de acesso, uma vez que os dados podem estar distribuídos em lugares desconhecidos e são de difícil recuperação.

- Nuvem Híbrida (*Hybrid Cloud*): Uma nuvem privada, no entanto, pode dar suporte à uma nuvem híbrida, através da complementação da capacidade da infraestrutura local com a capacidade computacional de uma nuvem pública (SOTOMAYOR et al., 2009), i.e., além dos recursos exclusivos de uma organização, podem ser requeridos recursos complementares a um provedor de computação em nuvem. Normalmente a requisição de recursos extras é usada sob-demanda para manter a

qualidade do serviço. A nuvem privada/híbrida também pode permitir acesso remoto através de interfaces, como por exemplo a de *Web Services* que a *Amazon EC2* utiliza (SOTOMAYOR et al., 2009).

Na Figura 4 definições supracitadas são ilustradas com a divisão entre os recursos externos e internos.

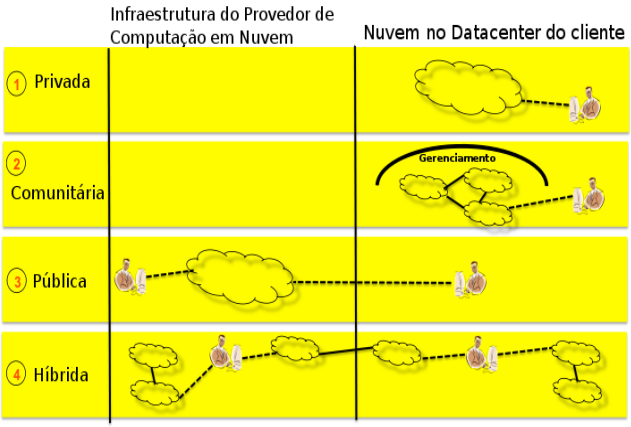

Figura 4: Modelos de Implementação, traduzido de (Cloud Security Alliance, 2009)

2.5 FERRAMENTAS

Com a popularidade da Computação em nuvem, muitas ferramentas e soluções têm surgido com o intuito de fornecer alternativas para a implantação de soluções de computação em nuvem, nesta seção serão apresentadas as principais ferramentas livres e de código aberto para o fornecimento de IaaS.

2.5.1 Eucalyptus

Eucalyptus (EUCALYPTUS, 2012) sigla para *Elastic Utility Computing Architecture Linking Your Programs To Useful Systems* ou, numa tradução livre, Arquitetura de Computação Utilitária e Elástica para Vincular seus Programas a Sistemas Úteis, é o resultado de um projeto de pesquisa do de-

partamento de computação da Universidade de Santa Bárbara, na Califórnia, Estados Unidos, em computação em nuvem, mais especificamente o Eucalyptus é um arcabouço para computação em nuvem que propõe uma arquitetura modular que facilita o entendimento e extensão para o uso na academia.

O arcabouço Eucalyptus foi desenvolvido para fornecer nuvens privadas ou para servir como parte de uma nuvem híbrida. Este implementa o conceito de infraestrutura como serviço e, assim, aborda questões do paradigma de computação em nuvem como instanciamento de MVs, ferramentas administrativas para computação em nuvem, redes virtuais e definições e execuções de SLAs. Basicamente esta ferramenta permite que seus usuários rodem e controlem uma coleção independente de máquinas virtuais compatíveis com a interface EC2/S3, as interfaces mais aceitas no mercado e que são consideradas padrões de *facto*.

Na sua arquitetura, o Eucalyptus é dividido em cinco camadas de alto nível e cada uma possui sua própria interface de *Web Service*, conforme apresentado na Figura 5, segue uma breve descrição de cada uma destas:

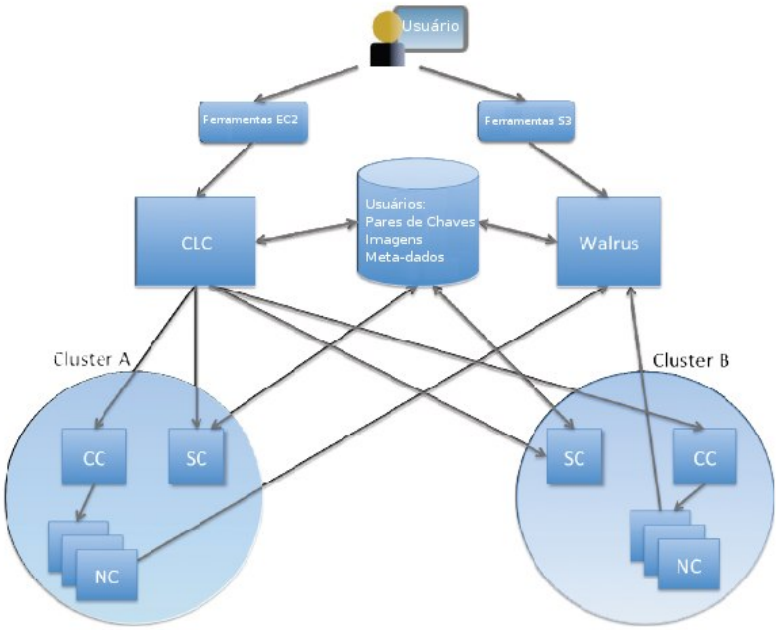

Figura 5: Arquitetura do Eucalyptus, traduzido de (EUCALYPTUS USER GUIDE, 2010)

- Controlador de Nó (*Node Controller*): O Controlador de nó é executado em todos os nós designados para hospedar MVs. É o responsável por controlar a execução, inspeção e término das instâncias de MVs. Responde diretamente ao controlador de cluster.

- Controlador de Cluster (*Cluster Controller*): Geralmente é executado na máquina principal de um cluster. Todos os nós submissos a este controlador precisam estar no mesmo domínio de *broadcast* (*ethernet*). O Controlador de Cluster faz também o agendamento e coleta informações da execução das MVs num controlador de nós específico e gerencia instâncias de rede virtual.

- Controlador de Armazenamento (*Storage Controller*): Serviço de armazenamento baseado no método *put/get* que implementa a interface S3 (*Simple Storage Service*) da *Amazon*, provendo um mecanismo para acesso e armazenamento de imagens de MVs e dados de usuários.

- Controlador de Nuvem (*Cloud Controller*): Ponto de entrada na nuvem para usuários e administradores. Este módulo indaga os gerenciadores de nodos sobre informações acerca de recursos, faz decisões de agendamento de alto nível e as implementa através de requisições aos controladores de cluster.

- Walrus: Permite que dispositivos de blocos sejam anexados às instâncias de máquinas virtuais como se fossem volumes de dados (partições), permitindo assim, a persistência das informações e modificações efetuadas durante a instância de MV. Este serviço é compatível e similar ao EBS (*Elastic Block Storage*) da empresa *Amazon*.

2.5.2 OpenNebula

O OpenNebula (OPENNEBULA, 2012) é um conjunto de ferramentas de código aberto para a criação de nuvens. Esta ferramenta pode ser usada para implementar nuvens privadas, híbridas, comunitárias e até nuvens públicas. Assim como o Eucalyptus, o OpenNebula é usado para fornecer IaaS, facilita e flexibiliza o gerenciamento da infraestrutura física transformando-a em virtual. A arquitetura do OpenNebula pode ser dividida em três macro camadas conforme ilustra a Figura 6:

Dentre as suas vantagens aqui são listadas algumas das mais importantes:

- Possui o Código Aberto - O que permite a melhor compreensão e a

possibilidade de modificar a ferramenta de acordo com a necessidade do usuário.

- Adaptável - Por ser modular possibilita a fácil alteração e adaptação para a ferramenta.

- Testado - Uma ferramenta utilizada por grandes empresas e universidades, financiado por países e empresas. Alguns dos casos de uso demostram o uso em grande escala, fundamentos compatíveis com o paradigma de computação em nuvem.

- Interoperável - Por seguir grande parte dos padrões disponíveis, o OpenNebula procura ser interoperável e evitar a incompatibilidade entre vendedores ou ferramentas.

Para melhor compreensão a seguir apresenta-se a uma outra visão da arquitetura com as funções de algumas partes do sistema, a Figura 7 também descreve esta arquitetura:

- Ferramentas (*Tools*) - São as ferramentas de gerenciamento que são desenvolvidas usando a interface provida pelo Núcleo do OpenNebula.

- Núcleo (*Core*) - Consiste nos componentes responsáveis pelo controle e monitoração de máquinas virtuais, armazenamento e nós. O núcleo usa a camada de drivers para fornecer os recursos necessários. Os principais componentes desta camada são:

- Gerenciador de Pedidos (*Request Manager*) - Lida com as requisições dos clientes.

- Gerenciador de Máquinas Virtuais (*Virtual Machine Manager*) - Módulo responsável por monitorar e gerenciar as máquinas virtuais.

- Gerenciador de Transferências (*Transfer Manager*) - Módulo usado para gerenciar a transferência de imagens de máquinas virtuais.

- Gerenciador de Redes Virtuais (*Virtual Network Manager*) - Cria e gerência uma rede virtual baseada na configuração do sistema.

- Gerenciador de Nós (*Host Manager*) - Módulo para gerenciar e monitorar os recursos físicos da nuvem.

- Banco de Dados (*Database*) - Armazenamento para a persistência de estruturas de dados como dentro do Opennebula.

- *Drivers* - OpenNebula possui um conjunto de módulos adaptados para interagir com *middlewares* específicos (ex. Hipervisores, mecanismos de transferência de arquivos e serviços de informação). Estes adaptadores são chamados *Drivers*.

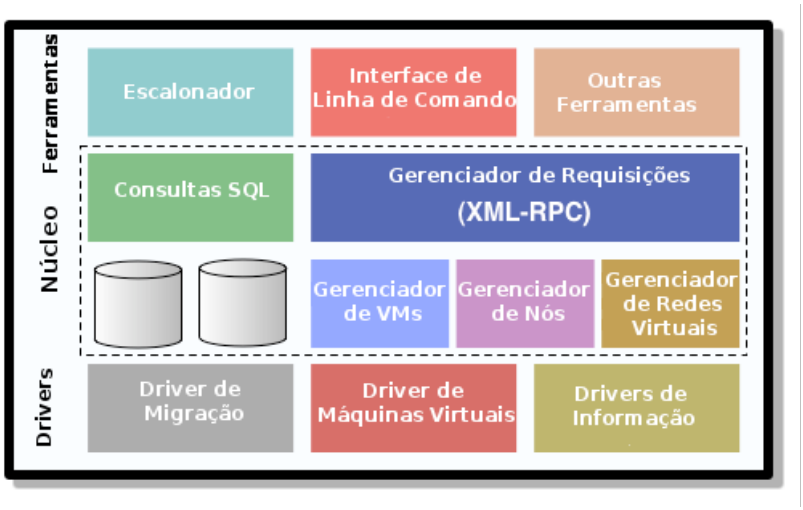

Figura 6: Arquitetura do OpenNebula, traduzida e adaptada de (LLORENTE, 2010)

Além das características supracitas o OpenNebula permite ao usuário acesso através da linha de comando, pela interface *SunStone* e por interfaces padrões como o OCCI e a EC2 (descrito na seção 4.2. A interface SunStone é um dos primeiros esforços dessas ferramentas para simplificar o acesso aos serviços providos e tem como principal objetivo usuários finais. No âmbito do controle de usuários é possível controlar por grupos, usuários por listas de controle de acesso.

2.6 FERRAMENTAS DE GERENCIAMENTO PARA OS USUÁRIOS DA NUVEM

O gerenciamento e o controle das instâncias de máquinas virtuais criadas pelos usuários requer ferramentas que auxiliam estes nesta tarefa além das próprias interfaces fornecidas pela ferramenta IaaS. Nesta seção, apresentamos alguma ferramentas que cumprem este papel e podem ir além, permi-

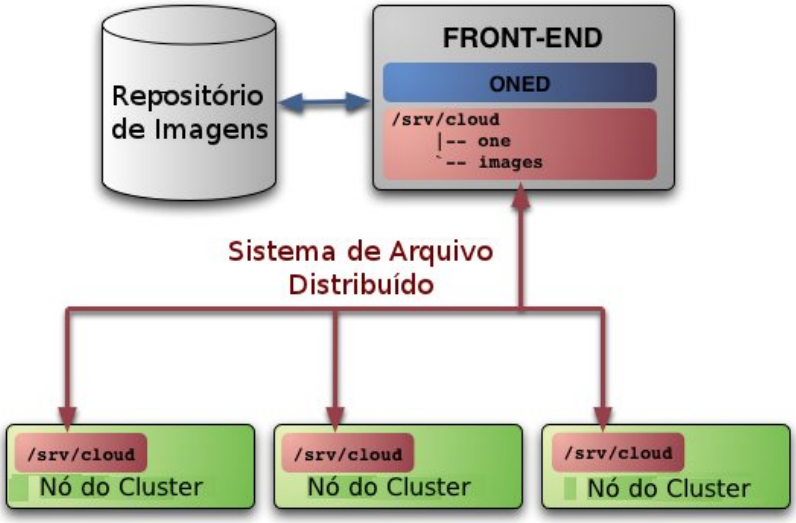

Figura 7: Outra visão da Arquitetura do OpenNebula, traduzida e adptada de (LLORENTE, 2010)

tindo o gerenciamento de nuvens diferentes.

2.6.1 HybridFox

HybridFox é uma extensão de código aberto para o navegador Firefox e, por conseguinte, possui como desvantagem a dependência deste navegador. Esta extensão provê uma interface amigável para interagir com *Web Services* compatíveis com o S3 e o EC2 da Amazon. A figura 8 ilustra essa interface. As suas principais características são:

- Interface Amigável;
- Fácil Instalação;
- Informar zonas disponíveis;
- Gerenciamento de máquinas virtuais (iniciar, listar, parar, reiniciar,etc);
- Gerenciamento de Imagens;e
- Gerenciamento de endereços Ips.

Além destas características existe a possibilidade de gerenciar diferentes nuvens usando esta ferramenta, desde que estas sejam compatíveis com as interfaces supra citadas.

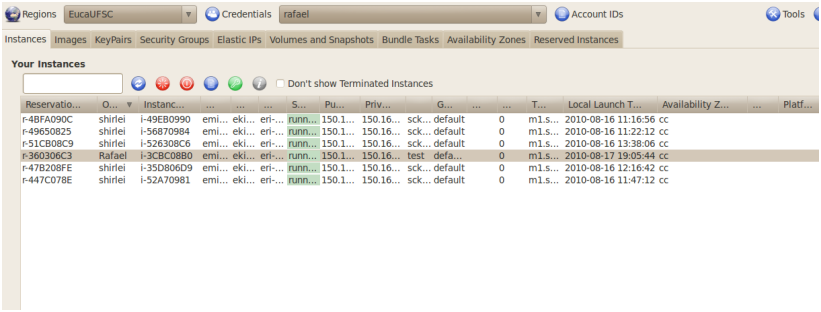

Figura 8: Tela da Ferramenta HybridFox

2.6.2 Euca2Tools

Euca2tools é um conjunto de ferramentas de linha de comando inspirado na ferramenta distribuída pela Amazon, usado para interagir com *Web Services* que usam uma API (*Application Programming Interface*) compatível com o EC2 e S3 da Amazon. Estas ferramentas podem ser usadas com o Eucalyptus, Amazon e todos os serviços compatíveis com estes padrões. Para o desenvolvimento destas ferramentas a linguagem Python foi usada junto com as bibliotecas Boto e M2Crypto. Esta ferramenta apresenta alguma dificuldade na sua utilização para usuários sem conhecimento técnico pois é necessário setar variáveis de ambiente para o seu funcionamento correto.

Seguem algumas características do Euca2tools:

- Informar zonas disponíveis;
- Gerenciamento de máquinas virtuais (iniciar, listar, parar, reiniciar,etc);
- Gerenciamento de Imagens;e
- Gerenciamento de endereços Ips.

2.7 SUMÁRIO DO CAPÍTULO

Computação em nuvem é um novo paradigma que utiliza muito dos conceitos e tecnologias existentes. Neste capítulo foram apresentados os principais conceitos das tecnologias relacionadas a computação em nuvem, em especial nuvens privadas, uma visão geral com os conceitos importantes dentro deste paradigma e sua situação atual. Entre estas, o SLA é uma ferramenta que garante o que foi contratado do lado jurídico e deve-se dar atenção especial pois por estar ainda em desenvolvimento, muitos padrões e funcionalidades não foram implementadas. Outra tecnologia usada na grande maioria dos casos é a virtualização pois permite o fácil gerenciamento de recursos não homogêneos e prove a flexibilidade para compartilhar os recursos e atender múltiplos clientes finais.

Esta junção de tantas ideias permite grandes benefícios mas deve-se considerar alguns problemas que podem ser ocasionados como a falta de detalhes na visão do sistema, o aumento da complexidade da solução e a dificuldade de encontrar falhas e erros. Ainda, a segurança em computação em nuvem é um campo muito estudado pela falta de soluções atuais que retardam a adoção deste.

Por fim, também foram apresentadas várias tecnologias importantes para o desenvolvimento do trabalho que ajudam a complementar o paradigma de computação em nuvem. Ferramentas como o OpenNebula e o Eucalyptus ajudam a difundir a computação em nuvem e permitir que a comunidade acadêmica e a indústria se beneficiem da computação em nuvem, não somente como usuário final, que aluga o serviço mas também como provedor (seja interno ou externo).

3 SISTEMAS AUTÔNOMOS

"Toute réussite déguise une abdication."

– Simone de Beauvoir (1908–1986).

Sistemas autônomos envolvem várias áreas da computação, de inteligência artificial até redes de computadores. Apesar de não ser um conceito novo, com o desenvolvimento e crescimento da complexidade dos sistemas distribuídos e redes de computadores, além do desenvolvimento de novas tecnologias, alavancam um interesse cada vez maior nesse tipo de solução.

Estes sistemas procuram transformar a ideia de existirem sistemas auto-gerenciados em realidade e, provavelmente, apareceu com a concepção dos próprios sistemas. Alguns projetos foram importantes e serviram de base para o desenvolvimento da ideia de computação autônoma. Já em 1997 a Agência de Projetos de Pesquisa Avançada de Defesa dos Estados Unidos (*Defense Advanced Research Projects Agency* ou DARPA), famosa pelo desenvolvimento inicial da internet, lançou um projeto militar de comunicação decentralizada para agentes móveis que se adaptava a mudanças do ambiente (alterando frequência, uso de banda e topologia) chamado *Situational Awareness System* (Sistema Consciente da Situação). Outros projetos da DARPA e de outros grandes centros vieram a seguir porém, o termo "sistemas autônomos" foi cunhado por (HORN, 2001) em um manifesto que assertava que um dos principais desafios da computação seria a complexidade dos sistemas e a dificuldade de gerenciá-los.

Neste capítulo serão apresentados os principais conceitos e características desse conceito.

3.1 PRINCIPAIS CARACTERÍSTICAS

Além disso, neste manisfesto os representantes da IBM sugeriram que quatro propriedades seriam essenciais para os sistemas autônomos serem auto-gerenciáveis: auto-configuração (*self-configuring*), auto-cura (*self-healing*), auto-proteção (*self-protecting*) e auto-otimização (*self-optimizing*).

Figura 9: Características dos Níveis de Autonomia, traduzido de (IBM White Paper, 2002)

Estes mecanismos e propriedades serão detalhadas nesta seção, bem como a classificação dos níveis de autonomia de um sistema.

3.1.1 Níveis Evolutivos

Na implementação de sistemas autônomos existe uma escala que define o nível de autonomia de um sistema partindo de um nível totalmente manual para o nível autônomo. A figura 9 representa esta evolução. Mais detalhes sobre cada nível são descritos a seguir:

- Nível Básico - Este nível é o ponto de partida para sistemas autônomos. Uma boa parte das empresas atualmente estão neste nível que parte do sistema é gerenciada independentemente pelos profissionais de TI. Estes instalam, configuram, monitoram e, eventualmente resolvem problemas deste sistemas sem maiores auxílios do próprio sistema.

- Nível Gerenciado - Neste nível algumas facilidades foram implementadas. Normalmente tecnologias de gerenciamento são usadas para coletar informações de várias partes dos sistemas em uma menor quanti-

dade de interfaces, reduzindo assim o tempo necessário para resumir e entender o que está ocorrendo no sistema.

- Nível Preditivo - Para alcançar este nível se faz necessário a introdução de novas tecnologias que possam prover a relação dos dados entre as várias partes dos sistemas, reconhecer padrões, verificar possíveis otimizações e sugerir melhoras. Estas sugestões podem ser aceitas ou não pelos administradores e apesar de um papel diminuto, os administradores são fundamentais para o bom funcionamento do sistema.

- Nível Adaptativo - Com o amadurecimento destas tecnologias e a melhora do poder preditivo os sistemas podem evoluir para o estágio adaptativo, onde podem corrigir e tomar ações automaticamente, isto é, sem intervenção humana. Estas ações são baseadas nas informações coletadas, no conhecimento do contexto do sistema e guiadas por SLAs.

- Nível Autônomo - Finalmente, o sistema autônomo é um sistema que age baseado em políticas de alto nível, nas quais os administradores definem objetivos. Os sistemas possuem uma fonte de melhores práticas e a experiência adquirida em níveis anteriores.

3.1.2 Propriedades de Auto-Gerenciamento

Adicionalmente. a efetividade das propriedades supracitadas depende de mecanismos e características que apoiem e forneçam informações relevantes para o sistema. A figura 10 sintetiza a relação destas características, as quais são explicadas nesta seção.

3.1.2.1 Auto-Configuração

A instalação, configuração de sistemas complexos é um processo caro e passível de erro mesmo se efetuado por especialistas. A auto-configuração é a característica que permite que um sistema se ajuste automaticamente às mudanças percebidas no sistema. Um exemplo seria com a introdução de um módulo no sistema atual, além da configuração deste novo modulo, pode ser necessário configurar os módulos existentes para interagir com o novo e, em um sistema autônomo, este processo seria contemplado pela propriedade de auto-configuração.

As adaptações se dão de acordo com políticas de alto nível - podem ser representadas, por exemplo, por Objetivos de Nível de Serviço (*Service*

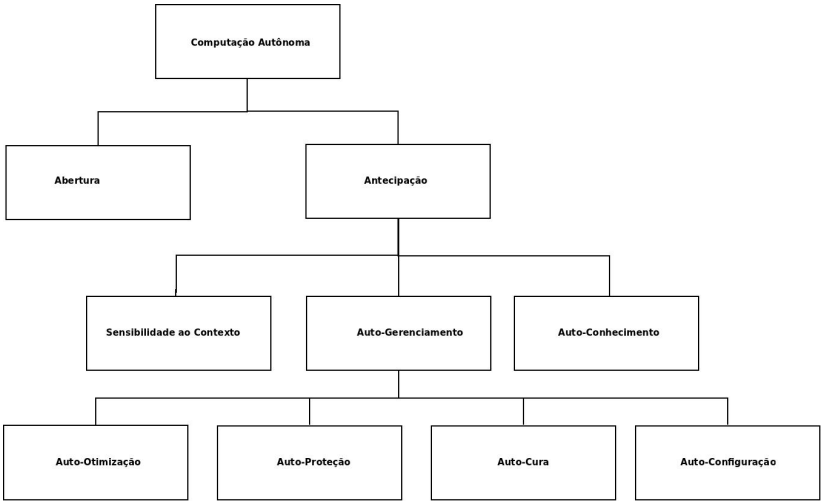

Figura 10: Relação das Propriedades e Características dos Sistemas Autônomos (adaptada e traduzida de (LIN; MACARTHUR; LEANEY, 2005)

Level Objectives ou SLOs) - que especificam o objetivo do sistema mas não como chegar a este objetivo.

3.1.2.2 Auto-Cura

Auto-cura é a propriedade do sistema que assegura sua recuperação efetiva e automática, quando falhas são detectadas. Entretanto, ao contrário de técnicas de tolerância a falhas tradicionais, auto-cura requer não só o mascaramento da falha, mas também a identificação do problema e seu reparo imediato, sem interrupção do serviço e com o mínimo de intervenção externa.

3.1.2.3 Auto-Proteção

Esta propriedade confere ao sistema a capacidade de se defender de ataques, tanto acidentais quanto maliciosos. Para a sua completude o sistema necessita meios (base de conhecimento, heurísticas, etc) para identificar estes ataques e possíveis soluções. Advertências iniciais podem ser usadas para prever e antecipar falhas no sistema. No âmbito de ataques maliciosos um exemplo seria alguém tentando invadir o sistema utilizando um servidor de

SSH (*Secure Shell* ou Terminal Seguro) com um ataque de dicionário para conseguir a senha de um usuário, já em ataques acidentais, um usuário poderia deletar um arquivo importante para o sistema sem intenção de prejudicar mas ainda assim deve ser prevenido.

3.1.2.4 Auto-Otimização

Auto-otimização consiste na capacidade do sistema de ajustar automaticamente suas políticas de utilização de recursos a fim de maximizar a alocação e uso dos mesmos, satisfazendo às demandas dos usuários. Esta propriedade procura constantemente otimizar as configurações para obter melhores resultados, medindo estes com os objetivos do sistema definidos por uma política de alto nível.

3.1.2.5 Outras Propriedades

Além das propriedades básicas, vários autores citam outras importantes características que ajudam a definir sistemas autônomos. É importante ressaltar que estas propriedades não são unanimidade mas ainda assim devem ser consideradas na construção de um sistema autônomo.

- Auto-Conhecimento (*Self-Awareness*) - A propriedade auto-conhecimento prevê a habilidade do sistema conhecer a si mesmo e seus componentes, saber do seu estado e disponibilidade de recursos. Uma das principais utilidades dessa propriedade é mensurar e auxiliar no planejamento de ações. Para o efetivo auto-conhecimento é necessário um sistema de monitoramento que fornece estas informações.

- Sensibilidade ao Contexto (*Context-Awareness*)- Um sistema autônomo deve ter uma visão de aspectos importantes do seu ambiente e área de atuação para prever e reagir a este.

- Antecipação (*Anticipatory*) - Como o próprio nome sugere, antecipação é a propriedade que procura antecipar e prever cenários e, assim, agir para evitar possíveis problemas ou tentar atingir um cenário que vá de acordo com a política de alto nível em vigor.

- Abertura (*Openess*) - A abertura é uma propriedade que sugere que os sistemas autônomos sejam portáveis para múltiplas plataformas, funcionem em ambiente heterogêneos e procurem usar padrões abertos na sua construção(NAMI; BERTELS, 2007).

3.1.2.6 Modelo MAPE-K

MAPE-K (*Monitor, Analyze, Plan and Execute* ou Monitorar, Analisar, Planejar e Executar) é o modelo de referência da computação autônoma apresentado em (KEPHART; CHESS, 2003) para o gerenciamento de elementos autônomos. Este modelo especifica um gerente autônomo para gerenciar recursos. Estes recurso podem ser qualquer software ou hardware dentro do sistema. Os dois principais elementos são o elemento gerenciado e o gerente autônomo. A Figura 11 ilustra este modelo.

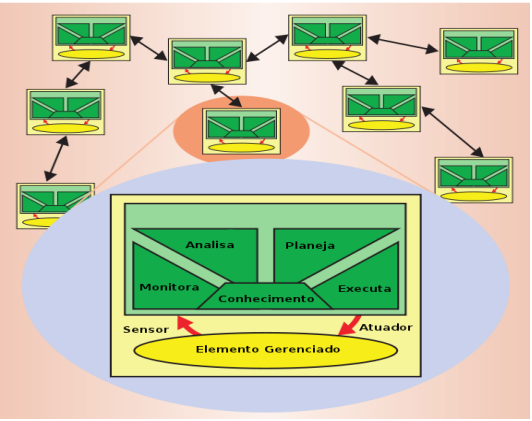

Figura 11: Modelo Mape-K (KEPHART; CHESS, 2003)

- Monitoramento - O monitoramento coleta dados do elemento gerenciado e os relacionamentos deste com outros elementos (que estejam dentro da visão de mundo do elemento gerenciado). Os dados coletados podem ser filtrados e até agregados entre si ou a informações armazenadas gerando informações e/ou conhecimento que permitam a identificação de problemas que demandem mudanças significativas para o sistema.

  Os tipos de monitoramento e os sensores a serem usados são, geralmente, específicos aos elementos gerenciados já que não existe uma interface padrão que abranja todos os elementos passíveis de monitoramento. Como exemplo de dados monitorados podemos citar um sistema autônomo para controlar um serviço de impressão, onde métricas como número de páginas impressas, nível de tinta dos cartuchos e disponibilidade de impressoras são importantes. Já métricas genéricas

de servidores podem ser uso da CPU, uso de memória RAM, tempo de resposta de um elemento da rede, etc.

- Análise - A etapa de análise foca nos dados adquiridos na fase de monitoração e procura achar padrões que sejam interessantes para o cumprimento das políticas definidas. Podemos exemplificar com a carga de processamento de um servidor. A informação que o servidor está com 90% de carga no momento pode estar dentro do previsto na política, mas analisando o histórico pode ser que seja necessário ativar um novo servidor com a mesma função para distribuir a carga.

  Uma vez que detectada uma anomalia ou alguma política não sendo totalmente satisfeita, o módulo de análise cria uma requisição para o módulo de planejamento. Estas requisições devem conter as mudanças a serem efetuadas. Este mecanismos permitem ao gerente autônomo aprender sobre o ambiente, bem como prever cenários futuros(IBM White Paper, 2005).

- Planejamento - Esta fase atende as requisições do módulo de análise e cria ou seleciona procedimentos para a realização das alterações nos elementos gerenciados. Este plano é representado por um grupo de mudanças desejadas e este grupo é passado para a próxima fase, a execução(IBM White Paper, 2005).

- Execução - A função de execução provê um mecanismo para agendar e fazer as mudanças necessárias no sistema (IBM White Paper, 2005). Estas mudanças foram definidas na fase de planejamento

- Conhecimento - A representação do conhecimento pode ser feita de diversas formas, neste trabalho serão apresentadas brevemente algumas das formas mais conhecidas e usadas para este fim.

3.2 SISTEMAS MULTI-AGENTES

O tema de sistemas multi-agentes está relacionado com diferentes áreas e o seu conceito base varia de acordo com estas áreas. Levando isso em consideração, neste trabalho usaremos o seguinte conceito:

Como o nome sugere, um sistema multi-agente é um sistema composto de múltiplos agentes inteligentes que interagem entre si. Em geral, estes agentes agem de acordo com políticas definidas pelos seus usuários (administradores, usuários finais, etc) e normalmente eles cooperam, negociam e coordenam de acordo com os objetivos. Este paradigma pode ser usado para

resolver problemas que não são facilmente resolvidos por sistemas centralizados ou agentes individuais. Para isto, os sistemas multi-agentes proporcionam uma estrutura modular que ajuda a dividir essa complexidade em partes (SYCARA, 1998). Seguem algumas propriedades que são importantes em sistemas multi-agentes:

- Autonomia: Os agentes devem seguir diretivas de alto nível mas sem interferência direta do usuário. Também, em caso de falha de outros sistemas o agente deve tomar decisões autônomas mas sempre levando em conta os objetivos do sistema.

- Reatividade: É a propriedade que permite que o agente reaja sobre a influencia de mudanças no ambiente ou novas informações.

- Comunicação e Cooperação: Em muitas situação é necessário que os agentes interajam e troquem informações para coletar dados que possam proporcionar uma base sólida para a tomada de decisão.

- Raciocínio: Tendo a autonomia como uma das importantes propriedades, é necessário ao menos um certo grau de inteligência para a tomada de decisão. A fonte desta tomada de decisão é uma base de conhecimento, o estado atual e a habilidade de tomar decisões de acordo com as políticas de alto nível.

Na questão técnica alguns fatores são importantes para a colaboração e sucesso do sistema:

- Evitar um controle central e global do sistema;

- Os dados e informações devem ser descentralizados;

- A computação deve ser assíncrona.

3.3 SUMÁRIO DO CAPÍTULO

A empresa IBM previu em 2001 que um dos maiores desafios da computação nas próximas décadas seria lidar com a complexidade crescente dos sistemas computacionais. Para enfrentar este problema começaram a desenvolver um paradigma baseado no sistema autônomo humano. Este não se limita somente a rede de computadores e computação em nuvem mas é muito utilizado em outras áreas como controle de viniculturas ou gerenciamento de robôs de salvamento. A sua principal característica é o auto-gerenciamento,

apoiado por suas quatro propriedades base: auto-cura, auto-proteção, auto-configuração e auto-otimização.

O gerente autônomo possui quatro funções: monitoramento, análise, planejamento e execução. Muito similar, é o paradigma de sistemas multi-agentes que é considerado o melhor meio de se atingir sistemas autônomos (TESAURO et al., 2004), (DIAO et al., 2003) e (De Wolf; HOLVOET, 2003).

Assim, neste capítulo foram explorados os conceitos relacionados à computação autônoma e sistemas multi-agentes que servirão de base para a aplicação deste paradigma na computação em nuvem.

4 TRABALHOS RELACIONADOS E PADRÕES

"If I have seen farther than others, it is because I was standing on the shoulder of giants."

– Isaac Newton (1643–1727).

4.1 TRABALHOS RELACIONADOS

O escopo dessa dissertação reuni vários conceitos e paradigmas que de alguma forma se relacionam com o objetivo principal do gerenciamento autônomo de nuvens privadas. Assim este capítulo será dividido em subáreas com os trabalhos relacionados as partes do trabalho desenvolvido e terminará com uma visão sobre o que já foi desenvolvido para área da dissertação.

4.1.1 Monitoramento

Monitoramento é a ação de coletar informações relacionadas com as características e o status dos recursos que são de interesse. A área de monitoramento é um campo já consolidado que possui muitos trabalhos e abordagens diferentes. Estes diferentes trabalhos contemplam grande parte das necessidades gerais dos *data centres* e paradigmas vigentes. Para usá-los na computação em nuvem várias ideias e abordagens foram usadas.

A abordagem clássica cliente/servidor de monitoramento e gerenciamento se mostrou problemática e ineficaz por muitos motivos: a complexidade das redes, e os nós que estão se tornando cada vez mais heterogêneos e complexos; as redes com estes nós crescem vertiginosamente mas a escalabilidade destes sistemas é limitada; tarefas específicas como gerenciamento de serviços, prevenção de ataques e auto-cura não partilham o mesmo modelo (cliente/servidor) (RAY et al., 2008). Nesta nova era de redes orientadas a serviço, o paradigma tradicional de clientes/servidores é pouco compatível.

Assim, pesquisas têm focado em paradigmas decentralizados(GLITHO; MAGEDANZ, 2002) mas, apesar da grande quantidade de trabalhos, poucos

resultados práticos são relevantes. A maioria ainda sofre com problemas de falta de flexibilidade com tecnologias, a falta de auto-gerenciamento e principalmente a falta do foco em segurança. Nesta seção são analisados trabalhos com este foco.

No modelo Publicar/Assinar (*Publish/Subscribe*), as entidade que desejam enviar mensagens "publicam"o evento em um intermediário na forma de eventos, enquanto as entidades que desejam receber certos tipos de eventos assinam estes. O publicador não sabe quem será o receptor das mensagens e pode ou não saber se estes existem ou não. Do mesmo modo, componentes que recebem as mensagens podem não saber quem assinou para cada tópico. É comum neste paradigma que uma entidade possa ser publicador e assinante ao mesmo tempo. Para existir este modelo baseado em eventos é necessário usar uma entidade separada para este fim, isto é, um servidor que desassocia a conexão entre os agentes e provê os mecanismos necessários para que a mensagem chegue aos assinantes. Alguns estudos sobre a performance sugerem que este é um mecanismo de comunicação escalável e eficiente para grandes sistemas distribuídos (SEGALL et al., 2000) (CARZANIGA; WOLF, 2002).

No trabalho (KUTARE et al., 2010) é apresentado um sistema de monitoramento analítico para o gerenciamento de *data centres* de grande porte. Um das principais preocupações apresentadas no trabalho é a escalabilidade pois atualmente, até 25% dos dados armazenados por empresas são do monitoramento. O trabalho procura definir uma arquitetura para sistemas de todos os tamanhos e camadas de abstração usando o conceito de zonas, ideia útil para o monitoramento de nuvens pois em geral, as ferramentes de IaaS usam clusters como pontos de separação. Outra característica importante na arquitetura é análise local dos dados obtidos, pois apesar de usar parte do processamento local, existe uma redução na quantidade de dados transferidos e uma ideia da importância dos dados capturados, permitindo a tomada de ações de acordo. Foram efetuados testes em três diferentes cenários e estes mostraram que o uso do processamento e memória ocasionados pelo sistema de monitoramento e os resultados foram menores que 2% do total usado, resultados convergentes com um dos principais objetivos do trabalho, manter a escalabilidade e influenciar pouco na capacidade do sistema.

Em um trabalho anterior, o autor colaborou com o desenvolvimento de uma solução para o monitoramento de nuvens privadas denominado PCMONS (CHAVES; URIARTE; WESTPHALL, 2011). Esta solução é baseada em um arquitetura apresentada na figura 12 que foi validada com um sistema de monitoramento supracitado. A arquitetura é dividida em três camadas:

- Camada de Visualização - Esta camada é responsável por apresentar os dados monitorados aos usuários, normalmente administradores do sistema ou gerentes do negócio. Geralmente os dados desta camada

são apresentados como resumos ou em gráfico. Esta camada pode ser adaptada de acordo com o usuário final ou as necessidades dos usuários da nuvem;

- Camada de Integração - Como as nuvens são compostas de recursos heterogêneos e com diferentes interfaces é preciso uma camada de integração para a coleta de dados e fornecê-los de modo uniforme à camada de apresentação, este processo é feito nesta camada;
- Camada de Infraestrutura - Camada que engloba todos os recursos da nuvem, sejam eles hardware or software. Como exemplos temos: recursos de rede, servidores, sistemas operacionais, hipervisores, etc.

O PCMONS instrumenta, centraliza e simplifica a tarefa de monitoramento (Brundo Uriarte et al., 2011) para o administrador de rede e tenta preencher esta lacuna na área, percebida pelos seus autores. É importante notar que a principal função deste é fornecer e formatar dados para a melhor compreensão do sistema pelos administradores e assim simplificar a administração da nuvem, porém, o PCMONS usa uma arquitetura centralizada e não têm como meta suportar computação autônoma o que torna seu uso difícil e inadequado nesta área.

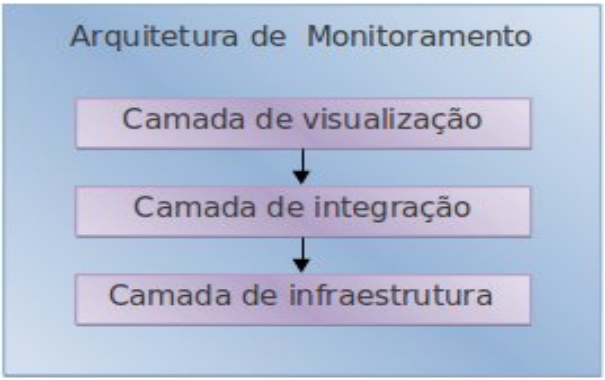

Figura 12: Arquitetura para Monitoramento de Nuvens Privadas

4.1.2 Computação em Nuvem

No trabalho (CAO; LI; XIA, 2010) é apresentada uma arquitetura para prover e requisitar serviços de uma nuvem com a qualidade de serviço (*Qua-*

lity of Service ou QoS) garantida.

Devido às similaridades (mesmo que as diferenças sejam grandes) entre grades e computação em nuvem, vale citar trabalhos com objetivos similares na área de grades que possam ser uteis no desenvolvimento do trabalho na área de computação em nuvem. Em (LIU et al., 2005) é apresentada uma arquitetura autônoma para tratar o dinamismo e heterogeneidade em grades para permitir que os serviços sejam adaptados de acordo com políticas de alto nível.

Foram consideradas áreas mais específicas mas ainda assim relevantes como base para o desenvolvimento desta dissertação. O problema de gerenciamento de recursos em computação em nuvem foi tratado no trabalho (WUHIB; STADLER; SPREITZER, 2010). Foi especificado um mediador (*middleware*) para a alocação de recursos usando um protocolo de fofoca (em inglês *gossip protocol*, protocolo baseado nas redes sociais que transmite informações para agentes aleatórios) para atingir: justiça na alocação dos recursos, adaptação e escalabilidade. Para este fim uma função de utilidade é formalizada considerando CPU e memória. Além disso, o artigo provê uma solução heurística para o problema, completo e testa através de simulação apresentando resultados satisfatórios.

Já no campo da semântica, o trabalho (EJARQUE; SIRVENT; BADIA, 2010) apresenta um sistema distribuído para alocação de recursos chamado SERA (*Semantically Enhanced Resource Allocation*) ou Alocador de Recurso Semanticamente Melhorado). O artigo usa o paradigma multi-agente para monitorar e negociar a alocação de recursos. O cenário usado é baseado em diferentes provedores de computação em nuvem (oferecendo IaaS) e a relação com clientes. As vantagens desta abordagem são a escalabilidade desta solução e a independência de tecnologias e hardware que podem existir entre os nós, pois cada agente trabalha de modo local e lida com as particularidades usando a concepção de BDI (*Belief-Desire-Intention* ou Crenças-Desejos-Intenções) para chegar a um consenso entre estes clientes e os provedores.

4.1.3 Computação Autônoma

Mesmo que esta área já tenha certo tempo de pesquisa existem poucos trabalhos com ideias inovadores para o gerenciamento autônomo de redes de computadores. Segue uma análise geral sobre as principais soluções propostas. Vale observar que, mesmo com certa flexibilidade, a aplicação destas tecnologias requer a adaptação das ferramentas para o cenário de nuvem.

Recovery-Oriented Computing (PATTERSON; BROWN, 2008): iniciativa da UC Berkeley Stanford que permite a modelagem de sistemas com

alta confiabilidade para os serviços da Internet, fornecendo suporte à recuperação de falhas inesperadas.

O projeto Oceano (IBM, 2005) foi proposto pela IBM e permite que se modele e desenvolva um protótipo de estruturas gerenciáveis e escaláveis para sistemas computacionais de larga escala voltado principalmente para a Web. Como exemplo, em caso de um grande evento a ser transmitido pela internet, vários servidores podem ser redirecionados para suprir a demanda. Alguns destes conceitos estão presentes na computação em nuvem.

Autonomia (DONG et al., 2003) é um sistema multi-agente desenvolvido pela universidade do Arizona que propõe atingir as propriedades de computação autônoma através da migração e auto-disposição dos agentes. Sua arquitetura prevê o uso nas redes legadas (sem suporte a computação autônoma) e é dividida em dois módulos: A interface de gerenciamento de componentes, responsável por prover uma interface para definir as políticas do sistema (comportamento) e os componentes gerenciados em tempo de execução, que por sua vez, gerenciam as operações usando o modelo MAPE-K.

Já o projeto ACCORD (LIU; PARASHAR, 2006) define um arquitetura autônoma como um conjunto de unidades de software independentes com interfaces específicas e dependentes de contexto. Este contexto e as regras (internas e externas) são usadas para orientar o comportamento adaptativo dos elementos e a sua composição.

O projeto ANA (BOUABENE et al., 2010) afirma que a internet pouco evoluiu desde o inicio e possui uma infraestrutura ultrapassada que em pouco tempo não será mais o suficiente para manter uma qualidade aceitável de serviço. Assim, este projeto procura explorar o paradigma de computação autônoma para desenvolver uma nova arquitetura para redes que mantenha o máximo grau de flexibilidade (suportando os serviços existentes) e seja escalável. Este projeto planeja suportar adaptação dinâmica e reorganização da rede de acordo com o contexto em tempo de execução, econômico e necessidade social dos usuários.

Estes projetos procuram prover ferramentas para o desenvolvimento de sistemas autônomos porém são generalistas, uma boa parte de código fechado/comerciais e também não consideram as particularidades do paradigma de computação em nuvem.

4.2 PADRÕES

Os ambientes computacionais envolvem diversos sistemas diferentes trabalhando em conjunto em diferentes organizações, plataformas e localizações geográficas. O que permite a constante evolução, interoperabilidade e

evita o aprisionamento por fornecedor (a impossibilidade de trocar de fornecedor por questões técnicas).

4.2.1 Computação em Nuvem

Segundo (MELL; GRANCE, 2009) importantes requisitos para que uma implementação seja identificada como computação em nuvem são: Segurança, Portabilidade e Interoperabilidade. Estes requisitos essenciais precisam de padrões maduros e de alta qualidade para serem levados em conta, mas geralmente estes processos precisam de anos para serem completados. Como a popularização de computação em nuvem é um fenômeno novo muitos esforços estão sendo despendidos com padronizações mas, em sua maioria, ainda estão nos estágios inciais e sem grande aceitação. Neste documento foram analisadas duas importantes iniciativas em direção a esta padronização:

- Open Cloud Computing Interface (OCCI) - O grupo de trabalho *Open Grid Forum* foi um dos primeiros grupos a se preocupar com a padronização de computação em nuvem. Este grupo, auto dominado líder em padronização para nuvens lançou uma especificação de um protocolo e API para interface de gerenciamento dos recursos e computação em nuvem. O protocolo OCCI é um protocolo de Transferência de Estado Representacional (REST), que foge dos padrões e práticas usadas no protocolo HTTP somente quando necessário. Esta interação ocorre pelos métodos (HTTP) como GET, POST, PUT, etc. Até a escrita deste documento, os trabalhos mais importantes publicados pela OCCI são (METSCH; EDMONDS, 2010), (EDMONDS; NYR, 2010) and (Thijs Metsch, 2009). Os dois primeiros são especificações do núcleo e a extensão deste para o gerenciamento e administração de IaaS. O último descreve as necessidades de administradores de computação em nuvem através de casos de uso. Um ponto relevante é a falta de um aprofundamento na questão de monitoramento, mencionado brevemente nestes documentos com intuito de descrever a sua importância, mas sem mostrar como.

- Open Cloud Standards Incubator - Esta iniciativa, do grupo *Distribute Mananagement Task Force* (DMTF) (Open Cloud Standards Incubator, 2010) foca em padronizar as interações entre múltiplas nuvens, criando uma arquitetura semântica e detalhando a sua implementação. O Gerenciamento interoperável é também uma prioridade e, para este fim, foi lançado um documento intitulado "*Use Cases and Interactions for Managing Clouds*"(Casos de Uso e Interação para o Gerenciamento

de Nuvens), que descreve esta arquitetura semântica. Mais especificamente, a parte de monitoramento é mencionada como uma necessidade mas sem detalhes adicionais.

Antes de escolher ferramentas para um nuvem privada é importante verificar a implementação de padrões usados nestas ferramentas. Na falta de padrões existe a possibilidade de acabar-se preso a um vendedor específico (*vendor lock in*) ou dependente de ferramentas particulares.

Em ferramentas como o OpenNebula, um esforço para a padronização foi iniciada, como por exemplo a implementação da API da interface OCCI baseada no último rascunho (*draft*).

Algumas ferramentas como o OpenNebula, mostram esforços em direção à padronização, implementando alguns padrões mais importantes (ainda que incompletos) como por exemplo a API da OCCI, baseado no último rascunho da especificação desta. Outra ferramenta citada neste trabalho, o Eucalyptus, não incluia nenhum padrão aberto, somente o padrão *de facto* usado pela empresa Amazon, o EC2.

Muitas organizações estão trabalhando em padrões da computação em nuvem e assim, tentando promover a tecnologia. Os esforços vão de testes de segurança (*benchmarks*), com enfâse na interoperabilidade até o desenvolvimento de interfaces para a interação com o usuário.

Os esforços não são limitados em desenvolvimentos de padrões para gerenciamento ou desenvolvimento de software mas contemplam também o usuário final. Uma adaptação da norma ISO 27002 (na área de segurança) para computação em nuvem está sendo efetuada mas ainda sem resultados concretos(BORENSTEIN; BLAKE, 2011).

4.2.2 Agentes

O paradigma de agente já é um paradigma amadurecido e muito discutido. Assim, foi criada uma agência chamada *Foundation for Intelligent Physical Agents* ou Fundação para Agentes Ativos Inteligentes para formalizar o resultado destas pesquisas. O seu objetivo é criar padrões para facilitar a integração de agentes heterogêneos e suas plataformas. Os principais resultados foram desenvolvidos nas seguintes especificações:

- Arquitetura Abstrata - A arquitetura abstrata FIPA (*Foundation for Intelligent Physical Agents* ou Fundação para Agentes Ativos Inteligentes) define os elementos desta arquitetura e a relação destes, provendo um guia para instanciação e interoperabilidade entre diferentes implementações.

- Especificação do Gerenciamento de Agentes - Nesta área, a FIPA desenvolveu um arcabouço normativo onde os agentes existem e operam. Neste documento são definidas a criação, registro, localização, comunicação e migração dos agentes. Em geral, neste arcabouço existe um serviço de nomes (páginas amarelas e páginas brancas) e um serviço de transporte de mensagens.

- Transporte de Mensagens de Agentes - Esta especificação trata do transporte de mensagens entre agentes e também de diferentes plataformas. São providos detalhes de como deve ser o envelope da mensagem e o comportamento básico deste.

- Comunicação de Agentes - A troca de mensagens entre agentes deve estar de acordo com a linguagem desenvolvida pela FIPA, a *Agent Communication Language (ACL)* ou Linguagem de Comunicação de Agentes. Esta linguagem segue uma estrutura gramatical específica baseada na teoria do discurso. Apesar disso, existe a flexibilidade de definição pelo usuário de mais parâmetros de acordo com a sua necessidade.

4.2.3 Computação Autônoma

A padronização da computação autônoma é um esforço relativamente recente iniciado pela difusora do termo, a IBM e outra gigante do setor de tecnologia, a empresa Fujitsu. Ao longo dos anos outras instituições aderiram ao esforço mas pouco foi desenvolvido até o momento. Segundo (TEWARI, 2006) os principais resultados destes esforços, até o momento, são a escolha das principais tecnologias a serem usadas. Segue uma lista destas:

- Esquemas XML (*Extensible Markup Language Schema*) - É um modo de documentar ou expressar algo em um modo legível por máquinas. É a base da troca de mensagens em computação autônoma.

- Linguagem de Descrição de Serviços Web (*Web Services Description Language ou WSDL*) - Linguagem criada pelo orgão W3C que usa o esquema XML para envelopar as mensagens e transmite via interfaces WEB.

- Modelo de Informação Comum (*Common Information Model ou CIM*) - É o modelo desenvolvido pela DMTF que usa o esquema XML para definir o formato das imagens de interação entre os gerentes autônomos sobre os elementos gerenciados.

Existem, também, padrões para definir a parte da base de conhecimento mas não são cobertos nesta dissertação.

4.3 DIFERENÇAS ENTRE NUVENS PÚBLICAS, NUVENS PRIVADAS E *DATACENTERS* TRADICIONAIS

Na comparação entre nuvens privadas e os data centres tradicionais, existem muitas similaridades entre os ambientes mas ainda assim algumas características justificam a adaptação das ferramentas existentes e criação de novas específicas para o contexto de computação em nuvem. Dentre estas características, elasticidade e rápida disposição de novos recursos, além da complexidade que uma nova camada (virtualização) traz com isto.

Existem também diferenças estruturais e a necessidade de abstrair a complexidade ou detalhes da arquitetura para os usuários finais. O trabalho (Cloud Security Alliance, 2009) elenca as principais características que diferenciam nuvens de outros paradigmas e, dentre estes, data centres tradicionais:

- Serviço sob-demanda - Os clientes podem, unilateralmente, requisitar a provisão de novos serviços/recursos automaticamente sem intervenção humana no lado do provedor (neste caso o provedor pode ser a própria organização e o cliente o usuário da organização).

- Velocidade de Acesso (Banda Larga) - O fornecimento de serviços necessita de uma velocidade de acesso mínima para o seu bom funcionamento e acesso. Em linhas gerais, nuvens privadas são menos afetadas pois a qualidade da conexão dentro da própria empresa é vista como prioridade, além de ser facilitada pelo número reduzido de pontos de acesso e pelas tecnologias usadas dentro da organização (ex.: LAN).

- Armazém de recursos - Armazéns de recursos são formados com recursos de uma ou mais localidades visando o uso otimizado e elasticidade. É importante ressaltar que até mesmo em nuvens privadas podem existir usuários com perfis diferentes mas que os recursos devem ser vistos como genéricos e independentes de localização.

- Elasticidade - O uso de recursos deve ser rapidamente escalável e elástico. Para os clientes estes recursos devem ser virtualmente ilimitados e serem usados de acordo com a sua necessidade.

- Mensuração do uso dos Serviços- A nuvem deve controlar e otimizar o uso de recursos utilizando métricas de acordo com o tipo de serviço

sendo fornecido. Tanto o provedor quanto o consumidor podem monitorar e controlar a utilização dos recursos.

4.4 SUMÁRIO DO CAPÍTULO

Este capítulo apresenta um resumo da literatura relacionada com este trabalho, é relevante notar que apesar da grande quantidade de trabalhos em cada área (agentes, computação autônoma, computação em nuvem, grids, clusters), pouco foi feito na integração dos conceitos apresentados. Com o objetivo de integrar e aproveitar as vantagens destas tecnologias para melhor gerenciar computação em nuvem, este trabalho foi efetuado.

5 PROPOSTA

"Un tas de pierres cesse d'être un tas de pierres, des qu'un seul homme le contemple avec, en lui, l'image d'une cathédrale"

– Antoine de Saint-Exupery (1900–1944).

5.1 ARQUITETURA PROPOSTA

Em uma análise profunda da literatura foi percebida uma falta de uma base sólida para o desenvolvimento de soluções compatíveis com os requisitos de nuvens privadas. Assim, neste trabalho foi proposta uma arquitetura e consequentemente um arcabouço para o desenvolvimento de soluções que forneçam um monitoramento de uma nuvem privada que permita a implementação de diferentes módulos para os primeiros passos em segurança na direção de uma nuvem autônoma.

Ao longo do estudo algumas características se mostraram importantes para atingir os objetivos do trabalho. Dentre estas, estão alguns requisitos que orientaram o desenvolvimento deste arcabouço:

- Arquitetura - A estrutura organizacional deve respeitar uma hierarquia de melhores decisões mas, em caso de falhas ou decisões simples, deve agir de modo autônomo (reforçando as propriedades dos agentes autônomos);
- Política - A política de alto nível definida pelos responsáveis deve estar disponível para os agentes independentemente do status dos outros componentes da rede. Assim permitindo a decisão autônoma dos agentes;
- Visão dos Agentes - Os agentes devem ter a percepção local dos componentes no nível corrente e dos níveis abaixo na hierarquia;
- Decisão- Os agente devem entrar em acordo sobre a ação ou plano a ser executado, levando em conta a hierarquia e escopo da visão de cada um;

- Arcabouço - O arcabouço deve ser modular, de fácil compreensão, simples de estender e afetar o mínimo possível no desempenho da nuvem privada.

Em linhas gerais, dentre as definições necessárias que posteriormente serão detalhadas temos: banco de dados local e geral, tomada de decisão, comunicação, disposição dos agentes.

5.2 DETALHES DO ARCABOUÇO

Definir um arcabouço flexível, de pouco impacto negativo no sistema e suportar sistemas que foram desenvolvidos sem levar em conta a possibilidade de gerenciamento autônomo é um grande desafio e requer a especificação em vários níveis. Considerando isto, o arcabouço foi dividido em duas grandes partes: Monitoramento e Segurança. O monitoramento é o responsável por fornecer os dados requiridos pelos módulos de segurança. Este mesmo sistema foi pensado para servir de base também para as outras propriedades dos sistemas autônomos: auto-configuração, auto-cura e auto-otimização.

Por existirem grandes diferenças entre os modelos de entrega dos serviços em uma nuvem, limitou-se o escopo inicial deste trabalho para o modelo de Infraestrutura como serviço, o qual é flexível e provê maior controle. Devido a esta decisão, existe a necessidade de considerar tanto baixo nível (monitoramento dos sistemas operacionais) quanto níveis mais altos, como MV.

Além do modelo de serviço IaaS, também se optou pelo modelo de implantação de nuvem privada, pois este, em geral, é um dos meios mais rápidos de empresas e universidades se beneficiarem de computação em nuvem mantendo os dados e software sobre o seu próprio controle. Os modelos de serviço e de implantação são apresentados na Seção 2.4. Outra razão para a escolha é a falta de soluções específicas para esta área.

5.2.1 Monitoramento

A ideia de criar um sistema de monitoramento específico para este tipo de paradigma veio da participação na criação do sistema de monitoramento PCMONS (CHAVES; URIARTE; WESTPHALL, 2011), que desenvolve um sistema de monitoramento para nuvens privadas. Apesar de ser um dos poucos trabalhos desenvolvidos na área este sistema não foi desenvolvido com o intuito de suportar computação autônoma e redes com muitos nós.

Com isso foi decidido redefinir a arquitetura para suportar sistemas

multi-agentes, com ênfase na escalabilidade mas usando como base a arquitetura (com as modificações necessárias) definida em (CHAVES; URIARTE; WESTPHALL, 2011). Na figura 13 é detalhada esta arquitetura com as modificações prevista para cumprir com estes objetivos.

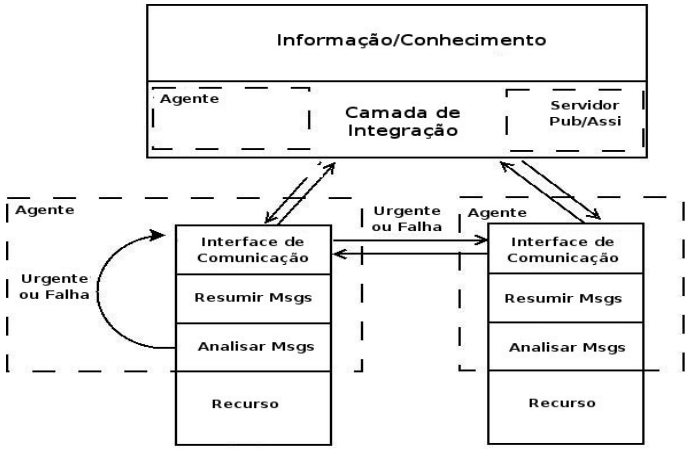

Figura 13: Arquitetura de Monitoramento Para Nuvens Privadas em Sistemas Autônomos

Se compararmos a arquitetura de monitoramento na qual a ferramenta PCMONS é baseada (vide 4.1) e a figura 13 vemos notáveis diferenças. O monitoramento efetivo na nova arquitetura funciona através de agentes, que além de adquirir as informações requeridas também pré-processam e resumem esta informação antes de enviar para a camada superior. Além do mais, esta arquitetura permite que em caso de falha de comunicação com as camadas superiores ou em caso de mensagens urgentes seja possível a comunicação entre agentes (aqui não limitados ao tipo de agente, i.e., os agentes de monitoramento podem transmitir informações para outros agentes como agentes de segurança), assim evitando atrasos em demasia provenientes deste pré-processamento.

A camada de integração é também gerenciada por agentes que possuem uma visão mais ampla e usam o mesmo processo de resumo dos agentes usados nos recursos. Os agentes dos recursos se comunicam via um servidor que fornece um canal de comunicação que usa o paradigma publicar/assinar e por fim prepara os dados e os coloca disponíveis no canal de comunicação e em uma base dados para a nuvem.

A distribuição dos agentes se dá tanto nos recursos físicos quanto vir-

tuais. Estes agentes de monitoramento devem verificar especificamente este nó e se comunicar com os outros agentes do sistema. Na figura 14 é apresentado um modelo para o agente de monitoramento base para a aquisição das informações em um nó. Este modelo pode conseguir informações de diferentes origens (arquivos de log, usando comandos no sistema, diretamente no sistema de arquivos, etc). Estas informações passam por um processo de pre-análise, resumo (para evitar tráfego desnecessário) e são enviados pelo canal de comunicação para os seus relativos destinos.

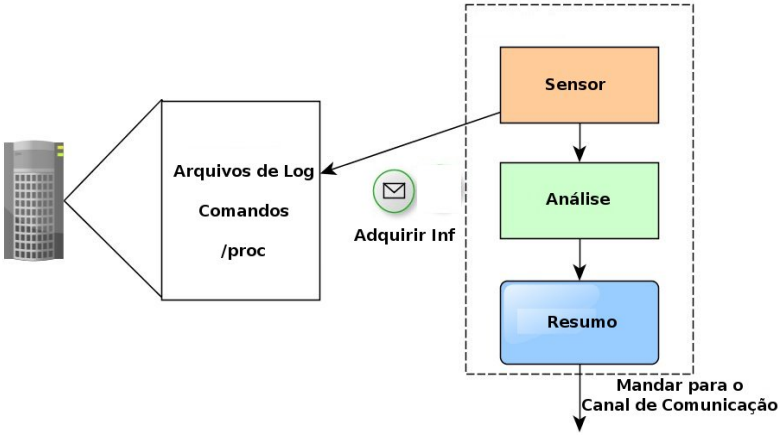

Figura 14: Captura de Informações e Pré-Processamento em Nós da Nuvem

A definição de quais recursos e informações são necessárias ocorre tanto pelo núcleo principal quanto pelos módulos acoplados pois estes últimos podem precisar de informações não suportadas pelo núcleo. De acordo com o nível hierárquico e capacidade computacional, as informações são processadas e analisadas para evitar a transferência de dados pouco úteis e manter o uso de recursos baixo. Ainda, se os agentes de segurança estiverem registrados para receber informações específicas, isto pode ocorrer no mesmo nível sem ter que chegar ao topo da hierarquia e retornar um comando dos agentes com visões mais abrangentes.

5.2.1.1 Canal de Comunicação

O canal de comunicação neste arcabouço é responsável pela organização da comunicação entre os agentes. Para este fim foi escolhido o modelo Publicar/Assinar, descrito na seção de trabalhos relacionados, pois permite a redução do tráfego de rede. Segundo esse objetivo, é necessário que, em caso de grandes volumes de dados a serem enviados, se permita uma conexão direta entre os agentes para evitar a transmissão para um terceiro.

A importância deste não deve ser subestimada porque é um dos principais responsáveis por permitir a escalabilidade do sistema. Para este fim, foi usado o conceito de publicar/assinar mas, adaptado às nuvens. Isto é, usa-se a hierarquia que divide em nós de uma nuvem privada em *clusters*, criando assim um canal de comunicação para os nós em cada *cluster* que, por sua vez, se comunicam com os canais de comunicação de outros *clusters*. Para minimizar a quantidade de dados transmitida é relevante uma análise no canal de comunicação da relevância da informação para outros canais de comunicação e o resumo das informações.

A figura 15 demostra a arquitetura do canal de comunicação na arquitetura de uma nuvem. Nesta, existem dois *clusters* com múltiplos nós e um gerenciador, cada *cluster* possui o seu próprio canal de comunicação que é responsável pela distribuição das mensagens publicadas para todos os agentes locais, inscritos que estejam no nível indicado pelo editor (os níveis dos agentes são indicados na próxima seção). No caso da figura apresentada, um agente enviou uma mensagem especificamente para os agentes gerenciadores de *clusters* e o canal de comunicação transmite esta mensagem localmente e, também, repassa para o canal de comunicação no segundo *cluster* que por sua vez passa a mensagem para o seu gerenciador local.

5.2.1.2 Tipos de Agentes

Os agentes deste arcabouço são adaptados de acordo com a camadas hierárquicas da arquitetura *de facto* usada em nuvens privadas. Seguem os tipos definidos neste arcabouço:

- Agente de Nós: Este agente é o responsável direto por capturar as informações nos recursos da nuvem. Estes recursos não são limitados a servidores, podem ser recursos de rede, recursos virtuais, etc. Estes devem tomar decisões autônomas somente em casos excepcionais como em falhas de comunicação ou em situações as quais são definidas como de emergência pelos módulos desenvolvidos.

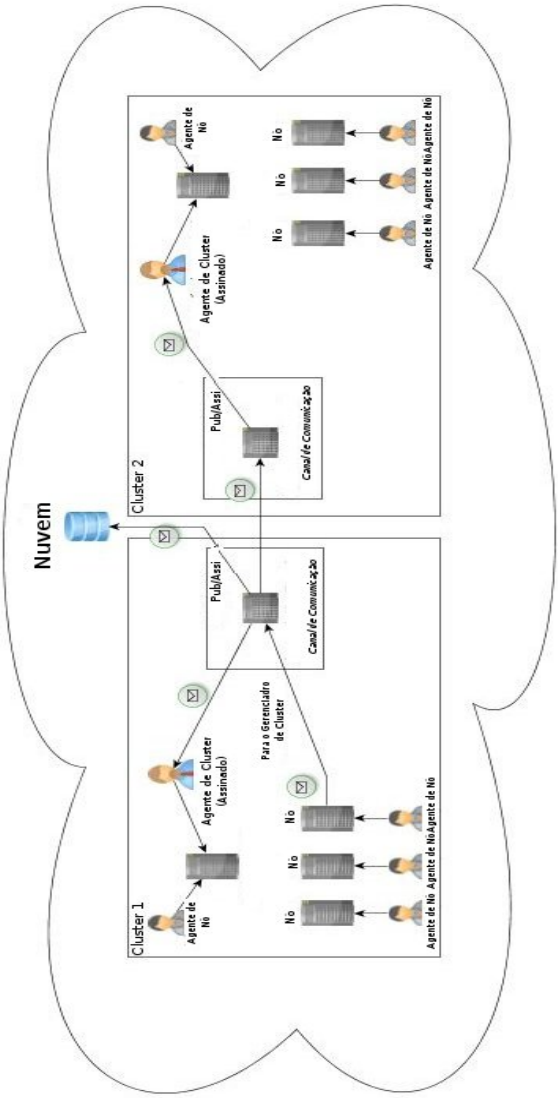

Figura 15: Canal de Comunicação na Nuvem

- Agente de *Cluster*: A principal função deste tipo de agente é agregar as informações dos agentes de recursos locais e, com uma visão mais abrangente, tomar e delegar decisões de acordo com a política do sistema.

- Agente de Nuvem: Este componente recebe as informações importantes (resumidas) de todos os *clusters* da nuvem privada e é capaz de se beneficiar da visão geral que possui dos componentes (virtuais e físicos) para tomar decisões mais complexas, não imediatas a primeira vista.

5.2.2 Segurança

O tema segurança em computação é um tema genérico que contempla muitas áreas. A computação em nuvem possui ambiente altamente dinâmicos e heterogêneos com novos tipos de ameaças surgindo diariamente, fato que requer um sistema flexível capaz de se adaptar ou ser facilmente estendido. Ainda neste paradigma, para sistemas de larga escala como computação em nuvem, segurança é um dos temas centrais porque é uma das maiores preocupações que impedem os usuários de adotar a tecnologia. Por isso na arquitetura criada, a propriedade de auto-proteção foi priorizada diante as outras propriedades (auto-cura, auto-configuração e auto-otimização). No trabalho de (CUSTOMER..., 2011) é feita uma análise do estado atual da segurança em computação em nuvem principalmente no ponto de visto do cliente. Como os dados, em geral, estão além do *firewall* do cliente, este não possuí total controle sobre as práticas de segurança. Dentre as principais preocupações temos confidenciabilidade, integridade, disponibilidade e vulnerabilidade a ataques. Considerando estas preocupações, o Panoptes procura dar suporte para evitar ataques em ambientes de computação em nuvem.

Esta parte do arcabouço que lida com a segurança é focada em módulos. Os usuários do arcabouço definem módulos que requerem informações e dados do sistema de monitoramento e utilizam destes para mensurar o nível de perigo de um determinado evento. A definição da reação sobre determinado nível de perigo, calculado pela função de utilidade que responde com uma porcentagem, é determinada pelos próprios módulos. Esta determinação ocorre após o uso de lógica nebulosa para converter a porcentagem em um dos valores definidos.

Para lidar com problema na arquitetura proposta serão designados agentes para os principais recursos das nuvem privadas e nos principais níveis hierárquicos (são detalhados na seção de implementação). Este agentes podem agir a partir de quatro fontes de informação diversas:

- Base Central - A primeira alternativa é solicitar a uma base de dados disponível para toda a nuvem, esta base, além das informações recentes é o modo padrão (modo escolhido pelo agente se nada anormal estiver ocorrendo) pois provê a possibilidade de uma análise mais profunda pois conta com histórico e dados que vão além das informações básicas.

- Ordem dos Agentes Superiores na Hierarquia - Como os agentes em camadas superiores possuem acesso mais abrangente a informações, estes podem detectar ataques com maior probabilidade de acerto. Assim, comandos oriundos de níveis superiores da hierarquia devem ser seguidos.

- Entre Agentes - Em caso de problemas ou de dificuldades nas duas primeiras fontes de ações, é possível a troca de informações e visões entre agentes.

- Autônomo - A última fonte destas informações é uma base de dados local resumida que supre o agente com informações básicas para que, em caso de falhas das duas primeiras fontes de informações, o agente continue funcional e tome as suas próprias decisões. Esta, também serve para orientar outros agentes.

A figura 16 mostra um diagrama de atividades que descreve o procedimento genérico dos agentes de segurança no recebimento de informações de outros agentes ou do canal de comunicação. A entrada de mensagens se dá por outro agente ou pelo canal de comunicação, de qualquer modo, existe a procura pelos padrões especificados no módulo de segurança e se encontrada qualquer informação correspondente a este padrão, esta é enviada para a fase pré-análise. Se for detectado que esta informação é urgente o agente pode efetuar uma ação para conter o problema, senão passa ao procedimento padrão, i.e., juntar com o histórico e tentar encontrar algum outro perigo e agir de acordo.

5.2.3 Implementação

Para a validação da solução, verificar se é factível e ter uma análise geral das funcionalidade e benefícios da arquitetura especificada foi decido criar um protótipo da arquitetura definida. Por ser um protótipo, não foram usados todos os conceitos e padrões citados mas somente as partes mais importantes do sistema que poderão, futuramente, ser aperfeiçodas e estendidas.

No desenvolvimento, a linguagem de programação Python foi preferida devido a familiaridade do autor com esta linguagem, a sua simplicidade

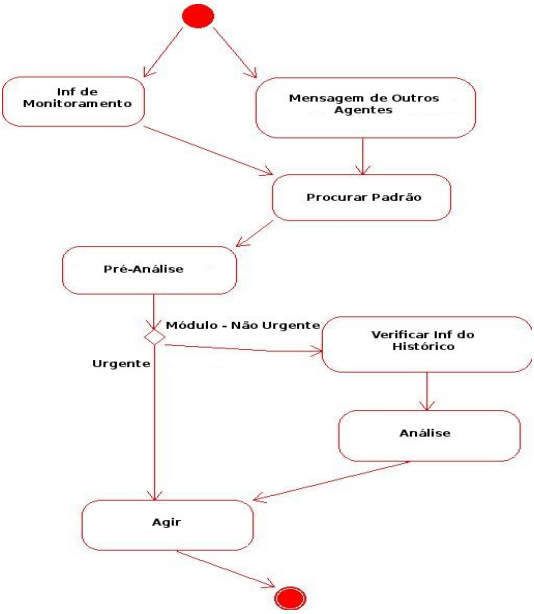

Figura 16: Diagrama de Atividades Descrevendo a Visão Geral dos Agentes de Segurança

e a possibilidade de reusar partes da ferramenta PCMONS(CHAVES; URIARTE; WESTPHALL, 2011), além da fácil leitura do código, o que motiva e facilita a sua extensão. Mais especificamente, do PCMONS, foram reusadas somente partes do código relativas a injeção de dados e programas nas máquinas virtuais(Brundo Uriarte et al., 2011). Segue a descrição das partes relevantes do trabalho divididas por área.

5.2.3.1 Base de Dados

Conforme a arquitetura, se faz necessário o uso de duas bases de dados: uma local para a persistência local dos dados que alimenta os agentes nos recursos para que em caso de falha de comunicação sirva de base para tomada de decisão e uma segunda base distribuída e comum para todos os agentes para agregação os dados e, assim, ter uma visão mais abrangente do sistema que permita o acesso também a informações históricas do sistema.

Para este fim, foi escolhido o paradigma Chave/Valor (*Key/Value Paradigm*). Este paradigma procura simplificar e retirar a complexidade do SGBD (Sistema de gerenciamento de banco de dados) e assim aproximar a base de dados aos problemas e o entendimento dos programados e gerenciados destes, em detrimento de compatibilidade, garantias de integridade e a generalização. Também, como não existe controle no SGBD, a aplicação fica responsável pela integridade dos dados. Porém, podemos citar várias vantagens, normalmente presentes nas implementações atuais:

Escalabilidade Vertical (scale up), significa adicionar recursos em um único nó do sistema (mais memória ou um disco rígido mais rápido). Ou seja fazemos um upgrade no hardware do nosso servidor para atender a demanda crescente de processamento e armazenamento.

Escalabilidade Horizontal (scale out), significa adicionar mais nós ao sistema, tais como adicionar um novo servidor e um sistema de software que permita a distribuição do trabalho entre múltiplas máquinas.

- Escalabilidade horizontal - Isto é, a possibilidade de adicionar mais nós ao sistema e que permita a distribuição do trabalho entre múltiplas máquinas. Esta forma de escalabilidade economicamente viável em grande implementações e perfeitamente compatível com computação em nuvem;
- Não existe a necessidade de definir esquemas;
- Orientada a Atributos;
- Performance.

A ideia principal é usar uma tabela de espalhamento (*hash table*) na qual existe uma chave única que aponta para um dado ou item em particular. Um dos principais problemas é a alteração parcial de dados, forçando a substituição toda do dado ou uso de operações especiais para este fim.

Para a função de persistência local dos dados decidimos usar a biblioteca para a linguagem Python chamada y_serial (Y_SERIAL, 2012). Esta biblioteca tem o objetivo de simplificar a persistência de dados e objetos. Ela é baseada no paradigma Chave/Valor, para abstrair a complexidade mas, quase que ironicamente, na sua implementação usa base SQLite para salvar os dados.

Assim, existe a possibilidade de adicionar notas (*tags*) para os dados e procurar por estas notas. Neste caso, o valor é um objeto Python que pode ser qualquer tipo de objeto, desde arquivos (imagens, binários, etc) a texto. No segundo plano, esta biblioteca serializa estes objetos e comprime, otimizando o uso sem grandes perdas de performance.

No caso da base central foi escolhida a solução chamada Redis (*REmote DIctionary Server* ou Servidor Remoto de Dicionário). Redis é um projeto de código aberto, que suporta o paradigma de Chave/Valor, salva os dados na memória com persistência opcional. Foi desenvolvida e é mantida pelos colaboradores Salvatore Sanfilippo and Pieter Noordhuis patrocinados pela empresa VMWare (VMWARE, 2012). A linguagem usada foi ANSI C mas suporta as principais linguagem conhecidas incluindo Python.

O projeto Redis segue um caminho alternativo no desenvolvimento do paradigma chave/valor, esta permite tipos mais complexos de dados e algumas operações atômicas para estes tipos de dados. A vantagem da abordagem do Redis, mantendo os dados em memória, é a grande velocidade de acesso e escrita e redução da complexidade interna, já que a representação dos dados em memória é muito mais simples. Dentre as possíveis desvantagens, não se pode salvar dados maior que a memória volátil e se não persistido pode ser perdido. Além disso, o Redis permite o uso de replicação e principalmente, a escalabilidade horizontal, característica que vai de encontro com a capacidade e benefícios das nuvens privadas.

5.2.4 Comunicação

Para a implementação do protótipo do canal de comunicação definido na arquitetura foi usado o protocolo XML-RPC, protocolo simples de chamada de procedimento remoto que utiliza o padrão de codificação XML e utilizando o protocolo HTTP como um mecanismo de transporte. Assim, para a criação do canal de comunicação foi desenvolvido um servidor que provê

acesso a métodos com funções para a comunicação usando o principio publicar/assinar. Seguem exemplos das assinaturas destes métodos considerados na implementação:

```
#Adiciona agente no sistema
def add_agent(self, agent):
...

#Enviar mensagem para um tópico
def send_topic(self, message, *topic):
...

#Assina um agente a um tópico.
def subscribe(self, topic, agent):
...

#Manda as mensagens não locais para os outros servidores
def send_SubPubServer(self, msg, agent, *topic):
...

#Remove a assinatura de um agente em um tópico específico
def remove_from_topic(self, topic, agent):
...
```

Existem outros métodos internos (como o de resumo de mensagem) mas não foram apresentados aqui para a simplificação. Além de um servidor por *cluster* existem ainda um servidor XML-RPC em cada agente para permitir a recepção de mensagens de outros agentes ou do servidor.

Para diminuir a probabilidade de falhas, foi considerado o conceito de redundância. A figura 17 explica este processo através de um diagrama de atividades que representa os estados dos canais de comunicação em relação ao sistema. Na inicialização o sistema define um agente para cada *cluster* como o canal de comunicação, neste protótipo o gerenciador da nuvem escolhe aleatoriamente um agente que possa ser acessado por todos os outros (conexão de rede). Do mesmo modo é escolhido um servidor para a replicação que é usado em caso de faltas. Após o processo de eleição é necessário definir nos agentes (iniciados pelo controlador da nuvem) estes servidor, em caso de falha no servidor principal, o servidor secundário é ativado para substituí-lo e um novo agente é escolhido para ser o servidor de replicação. Por último, este novo servidor é definido nos agentes.

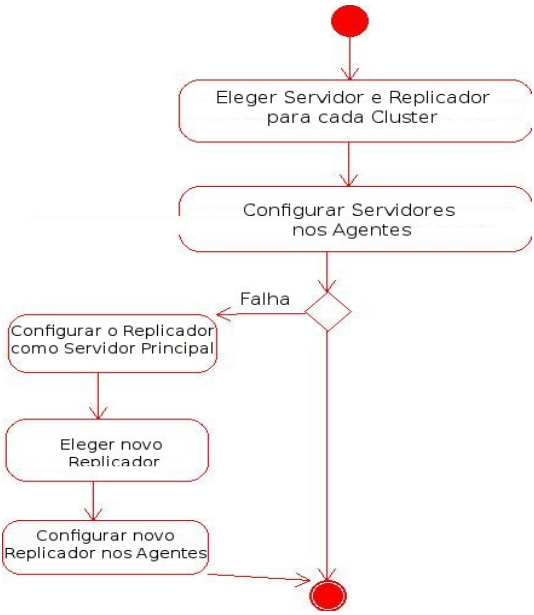

Figura 17: Diagrama de Atividades do Canal de Comunicação entre agentes

5.2.5 Decisão

A ideia inicial deste trabalho era criar uma linguagem padrão e esquema para especificar a tomada de decisão dos agentes e de um modo padronizado para ser usado por cada módulo mas, devido a limitação do escopo e de existirem vários trabalhos (RAO; GEORGEFF, 1995) (LIAO, 2005) (SUGENO; YASUKAWA, 1993) que o fazem, foi decidido usar as linguagens mais aceitas atualmente e restringir para uma linguagem específica no futuro.

Quando um módulo recebe a resposta de uma ação suspeita do sistema de monitoramento ele processa essa informação por uma função de utilidade, definida pelo próprio módulo e pode responder ao sistema de duas maneiras. A primeira é por uma porcentagem de risco ou diretamente com um valor de lógica difusa com os valores a seguir:

1. Baixo
2. Moderado
3. Substancial
4. Severo
5. Crítico

Estes valores são salvos no histórico e associados a um ator (que pode ser um ip, um login ou uma origem) que auxilia o sistema a tomar decisões futuras. Isto também evita gasto da memória local para os agentes. O agente salva somente data e hora origem, e nível de segurança. Se disponível outras informações (mais completas) são salvas no banco de dados geral. A função de utilidade deve ser implementada como uma subclasse da classe *Utility-Function* onde possui um método para receber um aviso de ação efetuada e responder com um dos valores supracitados.

O risco associado a cada ação efetuada por agentes fora do sistema devem gerar uma resposta de acordo com o nível do risco e a taxa definida na política de segurança.

Ainda, levando em consideração a autonomia local dos agentes em caso de falha na comunicação ou problemas no servidor de comunicação os agentes podem possuir uma base de regras no estilo Evento-Condição-Ação para tomar as próprias decisões. Resumidamente, este tipo de base espera um evento que procura condições que foram satisfeitas com este novo evento em uma base, se encontrar esta condição, uma ação especificada junto com a condição é tomada.

5.2.6 O Agente

O arcabouço prevê dois principais tipos de agentes, separados para manter a simplicidade e para permitir a fácil adaptação/substituições destes de acordo com a necessidade dos usuários do arcabouço. Outra razão para separar-los é mantê-los autônomos e independentes. Estes papéis são: monitoramento e segurança.

Os agentes apesar de possuírem diferentes papéis respeitam a mesma hierarquia, o que influi na visão que os agentes têm do sistema.

5.2.7 Disposição dos Agentes

No arcabouço os agentes devem ser posicionados de acordo com a plataforma escolhida. Grande parte das ferramentas disponíveis para nuvens privadas seguem o modelo descrito na seção de computação em nuvem 1.1.2. Esta arquitetura prevê quatro principais componentes: nós, controlador de nuvem, controlador de *cluster* e armazenamento.

Considerando também os objetivos deste trabalho, o sistema de agentes usa a mesma hierarquia, i.e., existem agentes para nós (podendo ser estendidos para armazenamento e componentes de rede) que possuem somente a visão local da máquina, agentes de *clusters* que possuem uma visão mais ampla capaz de coletar dados de vários nós neste *cluster* e dar respostas mais acuradas. Por fim os agentes de nuvem possuem informações mais concisas mas com a visão geral do sistema que permite ações mais complexas. A necessidade de criar a distinção entre os dois últimos tipos de agentes vem de uma tentativa de manter o sistema escalável apesar do seu tamanho.

A figura 18 ilustra o sistema e a disposição supracitada. Visando a simplificação da figura, foi adicionado somente um agente por recurso mas estes podem ser de diferentes tipos. No caso do Panoptes em cada recurso temos os agentes de segurança e agentes de monitoramento.

5.2.8 Monitoramento

A implementação dos agentes de monitoramento concentrou-se principalmente no papel dos agentes de nós, onde a coleta dos dados brutos ocorre. Dentro das suas funções o agente implementa o monitoramento de arquivos de *logs* ou o uso de comandos. No caso dos *logs* os agentes primeiramente verificam se os arquivos já foram monitorados anteriormente buscando fazer somente a análise diferencial. Esta verificação ocorre com a comparação das

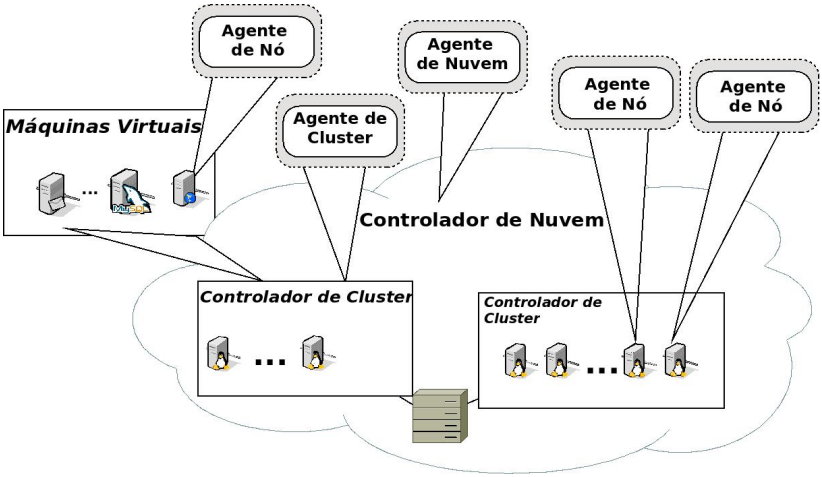

Figura 18: Hierarquia e Disposição dos Agentes no Arcabouço Desenvolvido

duas primeiras linhas (salvas na base de dados local) e duas linhas aleatórias do arquivo. Se o arquivo for o mesmo a análise inicia do ponto que foi terminada na última execução do agente. Após a análise do arquivo os agentes esperam por sinais do sistema de aquivo (configurados através da biblioteca chamada *PyInotify*) de modificação e verificam somente as novas inserções. Outro modo de monitoramento é com tempo pré-definido que verifica as diferenças ocorridas desde a última verificação.

Para análise no Panoptes são usadas expressões regulares para busca de padrões nestes *logs* ou saídas de comandos. A informação é disponibilizada para os agentes responsáveis. Também, pode-se especificar uma outra expressão regular para resumir a mensagem extraindo somente a parte importante desta. As informações são instanciadas na classe *Msg* e salvas na base de dados local e da nuvem (além de avisar os agentes se necessário de acordo com a seção 5.2.4). Como continuação, o resumo da informação ocorre com a compressão disponibilizada por bibliotecas da linguagem Python. O sistema permite também a extração e o uso do registro data/hora comum nos arquivos de *log*. Quando encontrado o padrão especificado uma mensagem é enviada para o controlador de nuvem se especificado nos módulos do nível de nuvens ou no controlador de *cluster*, se diferente.

5.2.9 Política de Alto Nível

Para o gerenciamento da política de alto nível, foi criada uma interface simples solicitando ao administrador o nível de segurança requerido pelo sistema onde o arcabouço Panoptes será usado. O resultado desta política permite definir a porcentagem (a entrada deve ser um número de 1-100) de segurança necessária no sistema. Futuramente, para um sistema mais complexo a tradução destas políticas para baixo nível poderá ser criado com abordagens como ontologias. Deve-se levar em consideração que quanto maior o nível de segurança requerido na política de alto nível, maiores serão as limitações e possíveis falsos alarmes já que com o mínimo sinal de perigo o sistema agirá para evitá-lo. Esta possibilidade pode acabar tornando a experiência dos usuários desagradável e o sistema ineficiente, assim ajustar o nível de segurança requerido é essencial.

Com esta dissertação o sistema passa aos módulos este valor e estes, por sua vez, ficam responsáveis em avisar o sistema em caso de violação desta variável de segurança. A decisão de deixar aos módulos foi tomada considerando a flexibilidade do sistema permitindo que cada módulo use diferentes variáveis e métricas na função de utilidade para dar uma resposta mais precisa.

O desenvolvimento desta interface foi efetuado com a linguagem HTML e Python CGI especificamente para a Web. Salvando-as na base de dados para a nuvem. A figura 19 mostra a página de configuração do Panoptes.

5.2.10 Configuração dos Módulos

A configuração dos módulos procura ser concisa mas mantendo a flexibilidade. Nas configurações previstas temos dois tipos de monitoramento, através do uso de comandos a serem efetuados em terminais ou arquivos pré-definidos. Se a linha com o arquivo (*File*) for deixada em branco o agente usará a linha de configuração do comando (*Command*).

A busca do padrão nos tipos de mensagens de um arquivo de log ou da saída de um comando é ditada por expressões regulares, se o padrão é encontrado a mensagem é processada de acordo com as configurações definidas nos módulos e explicadas a seguir. Um dado relevante neste contexto é o nível do agente a recolher as informações (*AgentLevel*). Pode-se também especificar a frequência da verificação do arquivo (se deixado em branco este verifica a cada modificação) e definir também a frequência da execução de um comando.

Visando a redução do número da troca de mensagens entre os agen-

Figura 19: Tela de configuração das políticas de alto nível

tes, existe a variável do tempo de resumo (*SummarizeTime*), isto é, permite que dentro deste intervalo as mensagens encontradas serão salvas somente localmente (não se aplica em mensagens com status urgentes) e depois resumidas e enviadas aos responsáveis. Já a variável "informações requeridas"(*InfRequired*) define qual parte da mensagem é relevante para a função de utilidade, prevenindo o envio de dados desnecessários. Esta última é também definida por expressões regulares.

Por fim, a variável *Actions* ou ações, defini o local onde as ações em caso de eventos (após a tomada de decisão) serão efetuadas.

```
[RegularExpressionCommand]
File = Endereço para o Arquivo
Command = Comando de Monitoramento
RegularExpression = Padrão a ser encontrado
InfRequired = Padrão para coletar as informações
#necessárias na linha do log ou da resposta
FrequencyFileCheck = Frequência da Verificação do arquivo
FrequencyCommand = Frequência da Verificação com o Comando
```

```
SummarizeTime = Tempo máximo para resumir os dados

AgentLevel = Nível dos agentes para capturar as informações

Actions = Onde os comandos definidos serão efetuados
#(qual tipo de máquina ou IP)
```

As mensagens especificadas através do modelo acima são enviadas aos agentes de segurança que devem conter um função de utilidade seguindo o modelo a seguir.

```
def utility_function(msg_list):
        percentage = 0
        return percentage

def action(level):
        if level == 0:
                pass
        elif level == 1:
                pass
        elif level == 2:
                pass
        elif level == 3:
                pass
        elif level == 4:
                pass
```

Este modelo, com o primeiro método, recebe as mensagens e as processa do seu próprio modo (escrito na linguagem python) e responde com uma porcentagem de risco. Esta porcentagem é devolvida ao sistema que compara com o nível de segurança requerido (definido nas políticas de alto nível) e este envia o nível para o agente de segurança que retorna uma instrução de reação ao sistema. Nesta instrução são contemplados o nível e o comando a ser executado.

5.2.11 Visão Geral da Implementação

O protótipo foi desenvolvido levando em consideração a arquitetura definida para o Panoptes. A figura 20 mostra a arquitetura do Panoptes e al-

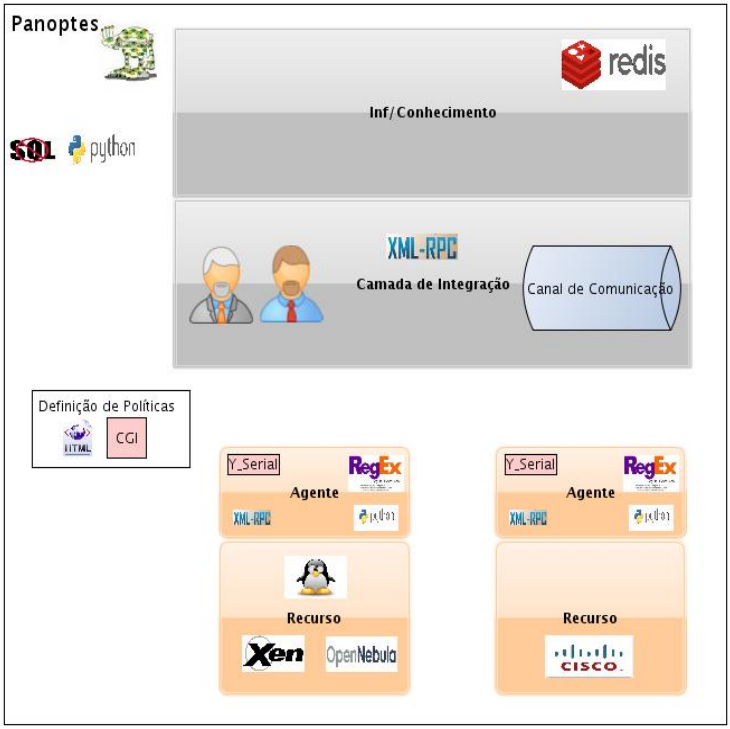

Figura 20: Arquitetura com as tecnologias usadas na implementação do arcabouço Panoptes.

gumas das tecnologias usadas nas camadas especificadas. Apresenta também exemplos de recursos nos quais os agentes agem, podendo ser uma MV iniciada pelo OpenNebula, um roteador da empresa Cisco ou outros. Alguns conceitos como *NoSQL* e a linguagem Python foram usadas em todo o sistema.

Ao longo da dissertação cada nó do diagrama de fluxo é detalhado, mas procuramos apresentar uma visão funcionamento do sistema como um todo. Assim, figura 21 demostra o funcionamento do sistema e passa uma visão estruturada das principais funções do arcabouço. As primeiras três ações, i.e., a coleta de dados importantes na nuvem, são efetuada pelos agentes de monitoramento (que inclui agentes em diferentes níveis da hierarquia). A parte de envio de informação e persistência dos dados é feita pelo canal de comunicação (servidor Pub/Ass). As outras ações apresentadas são de res-

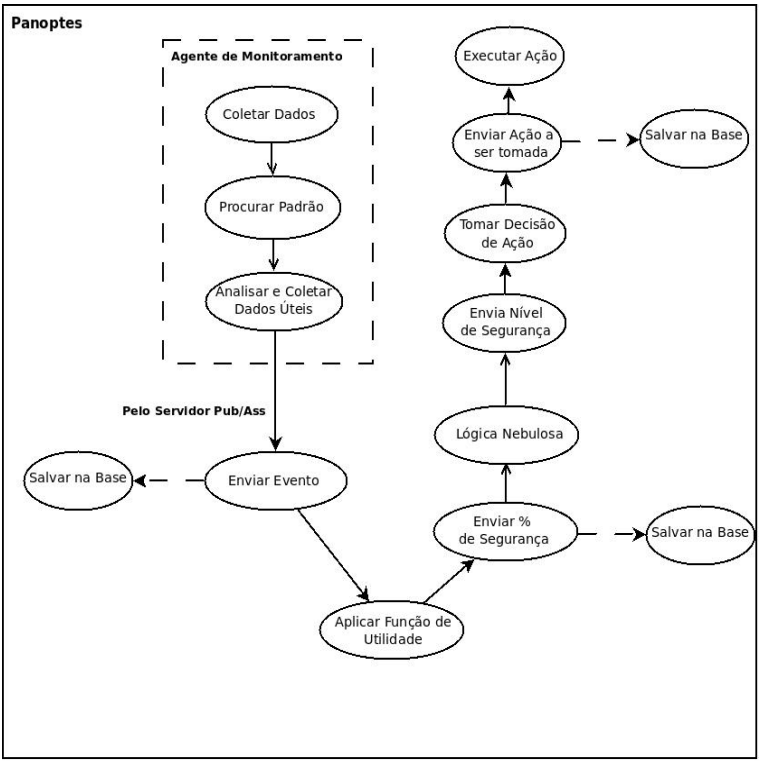

Figura 21: Funcionamento Geral do Arcabouço Panoptes.

ponsabilidade dos agentes de segurança, sempre orientados pelas políticas de alto nível e também pelas configurações dos módulos sendo usados.

5.3 ESTUDO DE CASO E AVALIAÇÃO EXPERIMENTAL

Como primeira fase do desenvolvimento do estudo de caso e avaliação experimental foi implementado um ambiente de nuvens privadas no laboratório de redes e gerência do departamento de informática e estatística da Universidade Federal de Santa Catarina. Inicialmente, um ambiente de IaaS baseado na ferramenta Eucalyptus foi instalado. Apesar de ser uma ferramenta relativamente madura, vários problemas na instalação foram encontrados e contornados. Mais sobre a experiência com esta ferramenta foi descrita e está publicada no trabalho (CHAVES; URIARTE; WESTPHALL, 2008).

Um dos pontos relevantes na utilização e desenvolvimento de ferramentas para computação em nuvem é a interoperabilidade e, como continuação do trabalho de pesquisa, foi instalado a ferramenta OpenNebula para verificar a compatibilidade e diferenças entre estas duas ferramentas. Com o resultados de testes e avaliações, OpenNebula foi escolhida como base desta dissertação por ser totalmente de código aberto (enquanto a ferramenta Eucalyptus possuí uma versão comercial), gratuita, ser amplamente testada e usada e modular. Um dos objetivos deste trabalho é ser facilmente adaptável a outras ferramentas como o Eucalyptus mas esta adaptação foge do escopo deste. A tabela 2 apresenta um resumo sobre o hardware e software usados para o desenvolvimento do ambiente de testes.

Hardware	Software	Papel no Sis.
Proc.: AMD Phenom 9650 Quad@2.3GHz Mem. RAM: 4GB DDR2 667MHz Disco Rígido: 750GB Interface Serial	OpenSUSE 11.2 OpenNebula 3.2 KVM Nagios 3.0.6	Control.
Proc.: Intel Core 2 Quad@2.3GHz Mem. RAM: 3GB DDR2 333 MHz Disco Rígido: 320 GB Serial ATA	OpenSUSE 11.2 OpenNebula 3.2 KVM	Nó
Proc.: Intel Core 2 Quad@2.3GHz Mem. RAM: 3GB DDR2 333 MHz Disco Rígido: 320 GB Serial ATA	OpenSUSE 11.2 OpenNebula 3.2 XEN	Nó

Tabela 2: Descrição do hardware e software usados no ambiente de teste.

Em relação as ferramentas e softwares escolhidos, a escolha do sistema operacional OpenSUSE se deu pela facilidade de instalação e configuração, especialmente do hipervisor Xen e da configuração de servidores, bastante simplificadas pelo uso da ferramenta de instalação e configuração YaST.

A escolha do hipervisor de modo geral está atrelada ao software para computação em nuvem escolhido. No caso, no OpenNebula há suporte para três dos hipervisores mais importantes disponíveis: Xen, KVM e VMWare. O hipervisor ou monitor de máquina virtual é que permite que seja executado o sistema operacional hóspede, criando as interfaces virtuais para as interfaces reais do sistema hospedeiro. No caso do hipervisor KVM (*Kernel-based Virtual Machine*), o hardware utilizado deve possuir suporte a virtualização pelo processador, característica previamente encontrada apenas em computadores de grande porte, como mainframes, até o lançamento das extensões

IVT (*Intel Virtualization Technology*) pela empresa Intel e da AMD-V (*AMD Virtualization*), pela empresa AMD. Recentemente, a maioria dos computadores inclui este suporte. No hardware utilizado para testes, todos possuem suporte à virtualização, portanto a escolha inicial foi o hipervisor KVM. Devido a testes de compatibilidade e verificação da ferramenta OpenNebula, em um nó, o hipervisor KVM foi substituído pelo hipervisor Xen para verificar as possíveis dificuldades encontradas neste procedimento. Após a configuração da ferramenta suportar o estudo de caso, o seu uso ocorreu normalmente (mas foi necessário manter as imagens das máquinas virtuais sem otimizações e configurações específicas para manter a compatibilidade com os dois hipervisores).

Para verificar a validade do trabalho medindo o impacto do sistema descrito no sistema em geral foi desenvolvido um cenário que simula o uso real de nuvens privadas (em pequena escala). Segue a descrição deste estudo de caso.

5.3.1 Caso de Uso - Universidade TI

Como caso de uso foi definido o caso de uma universidade no setor de tecnologia que, para alguns cursos e matérias, fornece máquinas virtuais para testes e uso. Um problema comum para os alunos nesta área é a falta de uma máquina de teste, sem alterar o computador de produção que requer instalação e configuração de programas e sistemas. Outra questão é a aprendizagem através de exemplos e mostrando as consequências em máquinas reais. Assim, para este caso de uso, são disponibilizadas máquinas virtuais para este fim e com a possibilidade de adição de ferramentas com este objetivo.

Como exemplo, pode-se citar um curso de segurança que explica e desenvolve técnicas de defesas para ataques específicos. Usando MV é possível testar e aplicá-las sem danificar os sistemas de produção. Para a simulação deste ambiente foi usada uma imagem de MV padrão com o sistema operacional Ubuntu instalado e o servidor SSH para acesso a máquina.

5.3.1.1 Módulo Desenvolvido

Este módulo foi desenvolvido, tendo em vista ataques, uma interface de acesso para os alunos nas máquinas usadas, o protocolo SSH. No caso, foi especificado um módulo que visa conter múltiplas tentativas de acesso (mal sucedidas) a máquinas da nuvem oriundas do mesmo IP. Para este fim, foi especificado o seguinte módulo:

```
[RegularExpressionCommand]
#If you set a command and a file will verify both
File = /var/log/auth
Command =
RegularExpression = (.*sshd.*) (.*failure.*|.*failed.*)
InfRequired = (?:[0-9]{1,3}\.){3}[0-9]{1,3}

#if no time required for the information use T=n,
#the same ip of origin of the machine IO=n

#Frequency in seconds
#'Every' can be used to check every modification of file
FrequencyFileCheck = Every
FrequencyCommand =

#Max summarize time in seconds
SummarizeTime =
#Node Cluster Cloud Vm All
AgentLevel = Node VM
#Node Cluster Cloud Vm All IP
Actions = 10.180.48.3
```

Nesta especificação estão sendo definidos o tipo de mensagem que deve ser encontrada nos arquivos de *log*, isto é, mensagens que contenham a palavra SSHD (*Daemon of Security Shell* ou Servidor de Terminal Seguro) e que ao mesmo tempo possuam a palavra *failure* (falha) ou *failed* (falhou) que caracterizam tentativas frustradas de acesso a uma máquina, no caso só foi definida a procura em MVs e nós da nuvem. Se encontrado, recolhemos o ip mostrado na mensagem, além do IP o sistema manda automaticamente o horário da mensagem, o horário que a informação foi coletada e a máquina de origem.

Quando encontrado o padrão especificado, uma mensagem é enviada para o controlador de *cluster* que verifica o histórico dentro da janela definida na configuração do módulo. Este acesso é feito na base de dados da nuvem e a mensagem enviada. No caso, somente é efetuado um controle do número de tentativas de acessos frustrados em uma janela de tempo definida na função de utilidade, bem como o número de mensagens e o risco que este número representa. É importante ressaltar que preparações adicionais na base

podem ser efetuadas, como identificar o tipo de usuário (se é um usuário que frequentemente esquece da senha, por exemplo) e usar para definir a porcentagem de risco, esta área será aperfeiçoada com a definição e criação da base de conhecimento.

```
def utility_function(msg):
#time of 5 seconds and max attempts 6
iniT = msg[0] #time of msg
sourceIp = msg[1] #ip attacker
timeWindow = 5  #In seconds

maxNumber = 6

if msg_list is not None:
msg_list.append(msg)
else:
msg_list = [msg]

if msg_list.__len__() > maxNumber:
return 100
else:
return msg_list.__len__() * (100/maxNumber)
```

Com a resposta devolvida sobre o risco da mensagem monitorada o sistema de monitoramento envia uma mensagem contendo a porcentagem calculada para o controlador de nuvem que calcula o nível de risco (de 0 a 4), no momento esse cálculo foi feito pela seguinte escala:

- 100 até 90 - Nível 0
- 90 até 80 - Nível 1
- 80 até 65 - Nível 2
- 65 até 50 - Nível 3
- 50 até 0 - Nível 4

Recursos avançados como lógica nebulosa baseados em bases de conhecimento podem ser usados para a tarefa. Finalmente, com este resultado o sistema a partir de uma base de ações e as efetua no sistema. No caso, o usuário pode ser bloqueado por tempo limitado, de acordo com o risco.

```
def action(level):
if level == 0:
#for the future, register in the database
#as a user that  might forget the password
#sometimes but with warning
pass
elif level == 1:
#for the future, register in the database
#as a user that  might forget the password
#sometimes but with warning
pass
elif level == 2:
#for the future, register in the database
#as a user that  might forget the password
#sometimes but with warning
pass
elif level == 3:
block_SourceAddress(timeMinutes=1)
pass
elif level == 4:
block_SourceAddress(timeMinutes=5)
pass
```

5.3.2 Análise dos Resultados

Os resultados do módulo supracitado foram de acordo com o esperado. Uma dos fatores chave do sistema de monitoramento da nuvem é a integração de informações de várias camadas (de software e hardware) dos componentes de uma nuvem privada e usar desta integração para tomar melhores decisões.

No caso de uso, um atacante para não ativar os sistemas de proteção de uma máquina (normalmente implementados para considerar só o sistema local) tenta diferentes senhas em máquinas diferentes de uma nuvem. Assim, o sistema definido consegue detectar ataques em diferentes máquinas e usar essa informação para bloqueá-lo evitando tráfego desnecessário e que o mesmo tenha êxito na invasão destas.

Os testes efetuados foram testes de força bruta, isto é, tentativa e erro de senhas em máquinas diversas ou em somente uma mas com a vantagem que o bloqueio é efetuado em todos os tipos especificados ou em máquina definida na seção *Actions*. Com isso, o sistema conseguiu cumprir o espe-

cificado no módulo. Ainda que não o ideal, o estudo serviu para mostrar a viabilidade e simplicidade no desenvolvimento de módulos.

Considerando a implementação e os testes efetuados alguns pontos são considerados cruciais. O impacto direto no sistema na inicialização dos agentes é aceitável, levando em conta que o tempo médio de inicialização de uma máquina virtual (incluindo o tempo para a liberação do endereço IP) é acima de 16 segundos (com uma média de 21 segundos), enquanto o tempo adicionado para transferir o agente e inicia-lo adiciona menos de 1 segundo no tempo total, número totalmente aceitável em um ambiente de produção.

O impacto na performance do sistema e o número de mensagens requerem uma análise mais profunda e o desenvolvimento de um modelo estocástico para simular o número de eventos oriundos do sistema em questão. Estas duas variáveis são diretamente proporcionais as configurações definidas nos módulos, podendo ser otimizadas com por exemplo, intervalos de comunicação maiores ou resumos usando as funções do Panoptes.

Estão previstos para o futuro testes reais de escalabilidade com uma nuvem híbrida e análises mais completas sobre a implementação em ambientes de produção.

5.4 SUMÁRIO DO CAPÍTULO

Este capítulo apresenta a proposta de uma arquitetura para trazer, parcialmente, a computação autônoma para nuvens privadas. Para a validação desta foi detalhado um arcabouço e justificadas as escolhas das principais partes desse.

Finalmente, foi descrito um caso de uso com a implementação de uma nuvem privada e deste arcabouço, apresentando uma breve análise dos resultados que vão de acordo com o esperado.

6 CONSIDERAÇÕES FINAIS

"Auch aus Steinen, die einem in den Weg gelegt werden, kann man Schönes bauen."

– Johann Wolfgang von Goethe,(1749–1832).

A computação em nuvem está ganhando cada vez mais espaço com a sua flexibilidade e capacidade virtualmente infinita de suprir demandas. Também, o custo atrativo e a abstração dos detalhes e problemas desnecessários para o seu usuário final são pontos chave nesta tecnologia, porém do lado dos fornecedores existe o aumento da complexidade com a integração dos recursos (normalmente heterogêneos) e a, possibilidade de adição de novas camadas (por exemplo a camada de virtualização).

Neste contexto, esta dissertação primeiramente apresenta os conceitos base da tecnologia alvo, a computação em nuvem, discorrendo sobre as suas peculiaridades, problemas, modelos, vantagens e desvantagens. Compreendendo a natureza sua, um paradigma que pode solucionar alguma das desvantagens de computação em nuvem é descrito em detalhes: Computação autônoma. A partir desta contextualização foi feita uma análise do estado da arte sobre soluções para os problemas apresentados.

Na literatura muitos estudos e soluções apresentam ideias para o desenvolvimento de sistemas autônomos mas são, geralmente, pouco flexíveis e sem contar com as particularidades de cada paradigma. Para computação em nuvem, como um paradigma relativamente recente, poucas soluções foram apresentadas, o problema se agrava se falarmos de nuvens privadas pois grande parte das pesquisas têm se voltado para nuvens públicas.

Com isso foi proposta uma arquitetura que procura fornecer uma base para a criação de sistemas autônomos na área de computação em nuvem. A especificação do sistema de monitoramento que serve como a principal fonte de informações para a tomada de decisões das propriedades da computação autônoma. As principais preocupações na criação desta arquitetura foram a robustez e escalabilidade devido ao tamanho destes sistemas e ao uso do paradigma de sistemas multi-agentes, que contribui e vai de encontro com as melhores práticas para sistemas autônomos.

Ainda, a arquitetura desenvolvida necessita de diretrizes de que tipo de informação coletar e como processá-las, logo, notando que uma das principais necessidades da computação em nuvem é a segurança, foi projetada uma arquitetura simplificada prevendo a propriedade de auto-proteção. Para suportar esta propriedade, se faz necessário a definição de agentes de auto-proteção guiados por políticas de alto nível, que interagem entre si. Do mesmo modo foi definido uma hierarquia de acordo com a visão do mundo dos agentes mas sempre considerando a hierarquia usada nas nuvens privadas.

Com o objetivo de validar as ideias supracitadas foi feita uma análise dos padrões a serem seguidos nas áreas (quando não existentes foram descritos os padrões *de facto*, ou seja, os mais usados) e um arcabouço foi desenvolvido. Este arcabouço conta com dois tipos de agentes, de monitoramento e de proteção além de contar com um canal de comunicação baseado no paradigma Publicar/Assinar e uma interface *web* que permite a definição de políticas de alto-nível, neste caso o nível de segurança aceitável no sistema.

Resumidamente, o seu arcabouço funciona da seguinte maneira: O administrador do sistema ou responsável pela organização defini o nível de segurança requerido nesta e seleciona ou desenvolve os módulos dentro das áreas a serem cobertas pelo sistema. Assim, os agentes de monitoramento coletam as informações e repassam as mensagens relevantes aos agentes de auto-proteção interessados. Estes por sua vez analisam as informações através de uma função de utilidade e, se necessário, tomam as devidas ações para evitar uma ameaça.

A solução desenvolvida ainda está nos seus estágios iniciais e não contempla muitas áreas importantes. Nem todos os padrões definidos foram implementados devido a sua complexidade e a limitação de recursos. Outro ponto relevante é a complexidade e a falta de linguagens formais para a definição das configurações (o que monitorar) e a definição da função de utilidade. Como um dos principais objetivos era manter a flexibilidade do arcabouço foi necessário passar a função de utilidade diretamente aos módulos mas, esta característica traz consigo o problema que requer conhecimentos técnicos de programação e análise acurada das causas. O filtro da coleta de informações prevê também somente expressões regulares, populares em muitas áreas porém não intuitivas para usuários com pouco conhecimento técnico.

Como validação do próprio arcabouço testes de escalabilidade não foram realizados devido a limitação de recursos físicos, as ferramentas de simulação de computação em nuvem foram estudadas mas ainda são limitadas, principalmente com as nuvens públicas e não suportam linguagens diferentes usadas neste documento. Com isso a validação do arcabouço foi focada na sua funcionalidade com o desenvolvimento de módulos e a sua utilização.

Um fato importante é que diferentes ferramentas e soluções foram tes-

tadas procurando compreender os principais problemas enfrentados por empresas e universidades no uso de nuvens privadas. Na nuvem criada testamos duas ferramentas, vários hipervisores e a suas combinações e não somente a utilização desta para validação da ferramenta. Em resumo, este trabalho é fruto da experiência pioneira do laboratório LRG no cenário nacional.

Dentre as maiores dificuldades no desenvolvimento e definição do arcabouço estavam a descentralização pois esta requer diversos mecanismos para manter o sistema funcionando, a criação de algoritmos para redundância e em caso de isolamento a autonomia para a tomada de decisão baseado na visão de mundo de um dos agentes. Um exemplo de aplicação no cenário de nuvens privadas com a rápida reação com informações oriundas de recursos diversos que podem direcionar o sistema para a melhor decisão, esta característica foi demostrada nos módulos desenvolvidos na seção da validação experimental 5.3.

Certamente este trabalho não traz uma solução final para a autonomia do gerenciamento de computação em nuvem mas serve como um passo adiante no fornecimento de uma base maleável para a solução do problema. No caso, esta base é principalmente a coleta de dados e a preparação da informação para sistemas autônomos, bem como outros sistemas que possam se aproveitar destas informações. Como prova de conceito, os agentes de segurança implementam parte do conceito de auto-proteção usando as informações preparadas pelo sistema de monitoramento mostrando o valor adjunto destas.

6.1 PRINCIPAIS CONTRIBUIÇÕES

- Divulgação - Computação em nuvem tem sido citado como uma da principais soluções para muitos problemas da computação. Este trabalho ajuda a contextualizar e consolidar um escopo para o paradigma, documentando e descrevendo muitas das atuais soluções. Neste contexto, é importante lembrar que no cenário nacional o laboratório LRG foi um dos pioneiros a desenvolver e trabalhar com computação em nuvem;

- Computação Autônoma para computação em nuvem privada - Abrir o caminho para trazer este importante conceito para nuvens privadas é um das principais contribuições deste trabalho. Analisando o futuro a curto e médio prazo existe uma forte demanda por soluções que simplifiquem a tarefa de administração dos paradigmas da computação e principalmente de computação em nuvem.

- Arquitetura - A computação em nuvem e a computação autônoma são conceitos que englobam diferentes áreas e conceitos. Na definição da arquitetura foi necessário levar em conta diferentes alternativas e modos de compor as tecnologias necessárias para formar uma solução apropriada para o problema. Assim, este trabalho apresenta uma solução com as escolhas justificadas e, não só o faz em plano teórico mas valida através de um arcabouço justificando as suas escolhas.

- Desenvolvimento de Software de Código Aberto - Com o objetivo de fomentar o desenvolvimento de mais soluções para computação em nuvem, de acordo com a política vigente do governo e como uma forma de devolver para a sociedade o investimento feito neste documento este trabalho foi desenvolvido em código aberto e gratuito.

- Disponibilização do resultado do trabalho - O Arcabouço desenvolvido estará integramente disponível para pesquisas e implementação. Como este foi desenvolvido para ser modular e extensível, ele permite a adaptação para diferentes necessidades e cenários.

6.2 TRABALHOS FUTUROS

Como já dito, este trabalho é somente o primeiro passo em busca do auto-gerenciamento em computação em nuvem e posteriormente de outros paradigmas da computação. Assim, muito ainda precisa ser feito em diferentes áreas. Seguem alguns dos possíveis trabalhos baseados seguindo a arquitetura e o arcabouço definido nesta dissertação:

- Implementação de Padrões - Como trabalho futuro se planeja a implementação destes padrões para o desenvolvimento e difusão deste arcabouço.

- Monitoramento Passivo - Uma característica algumas vezes requeridas para o monitoramento efetivo é o monitoramento passívo. Esta característica é muito usada principalmente por razões de segurança, onde um nó ou sistema específico não pode ser acessado e este envia as informações para os agentes do sistema de monitoramento. Se não enviadas um sistema de verificação é ativado e em caso de problemas uma ação pode ser tomada.

- Propriedades de Computação Autônoma - As outras propriedades dos sistemas autônomos, auto-cura, auto-otimização e auto-configuração

podem ser suportadas pelo arcabouço com a especificação e o desenvolvimento de agentes para isto. Lembrando que a arquitetura foi desenvolvida para ser modular e de fácil integração o que facilita a adição destas propriedades.Também pode-se ampliar o suporte a propriedade de auto-proteção com uma arquitetura especializada, que considere alternativas e possivelmente reuse paradigmas já existentes.

- Adição de módulos - A reação e detecção de necessidade de mudanças no sistema alvo devem ser especificados através dos módulos desenvolvidos pelos seus usuários e abranger o máximo de casos possíveis. Novas ameaças e desafios surgem todos os dias, por isso é impossível criar uma solução que contenha e preveja todos os tipos de estados dos sistemas e suas reações, e assim foi escolhido o sistema com módulos para cada tipo de ataque mas estes precisam ser criados manualmente.

- Desenvolvimento de Heurísticas - Para informações relevantes não previstas nos módulos, heurísticas para definir ações para conter novos tipos de ataques poderiam ser implementadas.

- Estender para outros tipos de Nuvem - A arquitetura pode ser estendida para nuvens híbridas e posteriormente para nuvens públicas.

- Representação do Conhecimento - Este trabalho focou principalmente no monitoramento e apenas salvou a informação em modo primitivo. Um trabalho importante na área é a organização da informação e transformação desta em conhecimento preparando para o melhor uso dos módulos. Um alternativa de solução seria o uso de ontologia e a preparação dos dados para o uso de técnicas com mineração de dados.

REFERÊNCIAS BIBLIOGRÁFICAS

AMAZON. *Amazon Mechanical Turk*. 2012. <http://www.mturk.com>.

BORENSTEIN, N.; BLAKE, J. Cloud Computing Standards: Where's the Beef? *Ieee Internet Computing*, v. 15, n. 3, p. 74–78, 2011.

BOUABENE, G. et al. *The autonomic network architecture (ANA)*. 2010. 4–14 p. <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper-.htm?arnumber=5371088>.

Brundo Uriarte, R. et al. Projeto e Implantação de um Arcabouço para o Monitoramento de Nuvens Privadas. In: *LATIN AMERICAN CENTER OF STUDIES IN COMPUTER SCIENCE (CLEI)*. [S.l.: s.n.], 2011. p. 1–16.

CAO, B.-q.; LI, B.; XIA, Q.-m. A Service-Oriented Qos-Assured and Multi-Agent Cloud Computing Architecture. *Architecture*, Springer, p. 644–649, 2010. <http://www.springerlink.com/index/05H477577VK3H728.pdf>.

CARZANIGA, A.; WOLF, A. L. A Benchmark Suite for Distributed Publish / Subscribe Systems. *System*, 2002.

CHAVES, S. A.; URIARTE, R. B.; WESTPHALL, C. B. Towards an Architecture for Monitoring Private Clouds. *IEEE Communications Magazine*, IEEE, v. 49, n. December, p. 130–137, 2011. ISSN 01636804.

CHAVES, S. A. D.; URIARTE, R. B.; WESTPHALL, C. B. Implementando e Monitorando uma Nuvem Privada. 2008.

CHELLAPPA, R. Intermediaries in Cloud-Computing: A New Computing Paradigm. In: *INFORMS*. [S.l.: s.n.], 1997.

Cloud Security Alliance. *Security Guidance for Critical Areas of Focus in Cloud Computing*. 2009. 1–76 p. <http://www.cloudsecurityalliance.org-/guidance/csaguide.pdf>.

Colt Technology Services. *European CIO Cloud Survey*. 2011. <http://www.colt.net/cio-research/>.

CUSTOMER Security Concerns in Cloud Computing. In: ICN 2011, The Tenth International Conference on Networks. [S.l.: s.n.], 2011. p. 7–11. ISBN 978-1-61208-113-7.

De Wolf, T.; HOLVOET, T. Towards autonomic computing: agent-based modelling, dynamical systems analysis, and decentralised control. In: *IEEE International Conference on Industrial Informatics 2003 INDIN 2003 Proceedings*. Ieee, 2003. p. 470–479. ISBN 0780382005. <http:/-/ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1300381>.

DIAO, Y. et al. Managing Web server performance with AutoTune agents. *IBM Systems Journal*, IBM Corp., v. 42, n. 1, p. 136–149, 2003. ISSN 00188670. <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper-.htm?arnumber=5386833>.

DONG, X. et al. Autonomia: an autonomic computing environment. *Computing*, v. 9984357, p. 61–68, 2003. ISSN 10972641.

EDMONDS, A.; NYR, R. *Open Cloud Computing Interface - Core*. 2010. <http://ogf.org/documents/GFD.183.pdf>.

EJARQUE, J.; SIRVENT, R.; BADIA, R. M. A Multi-agent Approach for Semantic Resource Allocation. *2010 IEEE Second International Conference on Cloud Computing Technology and Science*, Ieee, p. 335–342, nov. 2010. <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper-.htm?arnumber=5708468>.

EUCALYPTUS. *Eucalyptus*. 2012. <http://open.eucalyptus.com/>.

GLITHO, R.; MAGEDANZ, T. Applicability of mobile agents to telecommunications. *Network, IEEE*, v. 16, n. 3, p. 6, may-june 2002. ISSN 0890-8044.

HORN, P. *Autonomic Computing: IBM's Perspective on the State of Information Technology*. [S.l.]: IBM, 2001.

HUEBSCHER, M.; MCCANN, J. A survey of autonomic computing-degrees, models, and applications. *ACM Computing Surveys*, ACM, v. 40, n. 3, p. 1–28, 2008. ISSN 03600300.

IBM. *The Océano project*. 2005. <http://researchweb.watson.ibm.com-/oceanoproject/>.

IBM White Paper. *IBM Global Services and Autonomic Computing*. 2002. <http://www-3.ibm.com/autonomic/pdfs/wp-igs-autonomic.pdf>.

IBM White Paper. An architectural blueprint for autonomic computing. *Quality*, Citeseer, v. 36, n. June, p. 34, 2005.

International Data Corporation. *IT Cloud Services Survey: Top Benefits and Challenges*. 2009. <http://blogs.idc.com/ie/?p=730>.

JONES, M. T. Virtual Linux: An Overview of virtualization methods, architectures, and implementations. *Online Dec*, IBM, 2006.

KEPHART, J. O.; CHESS, D. M. *The vision of autonomic computing*. [S.l.]: IEEE, 2003. 41–50 p.

KLEINROCK, L. A Vision for the Internet. *Science*, Springer Berlin, v. 2, n. 1, p. 4–5, 2005.

KUTARE, M. et al. Monalytics: Online Monitoring and Analytics for Managing Large Scale Data Centers. p. 141–150, 2010.

LAUREANO, M. *Máquinas Virtuais e Emuladores*. [S.l.]: Novatec Editora, 2006. 1–184 p. ISBN 8575220985.

LIAO, S.-h. Expert system methodologies and applications?a decade review from 1995 to 2004. *Expert Systems with Applications*, Elsevier, v. 28, n. 1, p. 93–103, 2005. ISSN 09574174. <http://linkinghub.elsevier.com/retrieve-/pii/S0957417404000934>.

LIN, P.; MACARTHUR, A.; LEANEY, J. Defining autonomic computing: a software engineering perspective. In: *Software Engineering Conference, 2005. Proceedings. 2005 Australian*. [S.l.]: IEEE, 2005. p. 88–97.

LIU, H. et al. An autonomic service architecture for self-managing grid applications. *The 6th IEEE/ACM International Workshop on Grid Computing, 2005.*, Ieee, p. 8 pp., 2005.

LIU, H. L. H.; PARASHAR, M. *Accord: a programming framework for autonomic applications*. [S.l.]: IEEE, 2006. 341–352 p.

LLORENTE, I. M. *OpenNebula Cloud Case Studies*. 2010. <http:/-/opennebula.org>.

MCCANN, J.; HUEBSCHER, M. Evaluation issues in autonomic computing. *Grid and Cooperative Computing-GCC 2004Workshops*, Springer, p. 597–608, 2004. <http://www.springerlink.com/index-/2q95n1w323nk28nc.pdf>.

MELL, P.; GRANCE, T. The NIST Definition of Cloud Computing. *National Institute of Standards and Technology*, NIST, v. 53, n. 6, p. 50, 2009. ISSN 10476210. <http://csrc.nist.gov/groups/SNS/cloud-computing/cloud-def-v15.doc>.

METSCH, T.; EDMONDS, A. *Open Cloud Computing Interface - Infrastructure*. 2010.

MULLER, H.; KIENLE, H.; STEGE, U. Autonomic Computing Now You See It, Now You Don't. *Software Engineering*, Springer, v. 5413, n. Software Engineering, p. 32–54, 2009. ISSN 08929912. <http://www.springerlink.com/index/0k008550k285gq9v.pdf>.

MURCH, R. *Autonomic computing*. Online, Safari Tech Books, 2004. <http://www.lavoisier.fr/livre/notice.asp?id=OR2WRKAKO3AOWU>.

NAMI, M.; BERTELS, K. A survey of autonomic computing systems. *Expert Systems*, IEEE Computer Society, 2007. <http://www.computer.org-/portal/web/csdl% -/doi/10.1109/CONIELECOMP.2007.48>.

Open Cloud Standards Incubator. *Architecture for Managing Clouds*. 2010. 1–57 p. <http://dmtf.org/standards/cloud>.

OPENNEBULA. *OpenNebula*. 2012. <http://opennebula.org>.

PATTERSON, D.; BROWN, A. *Recovery-Oriented Computing*. 2008.

RAO, A. S.; GEORGEFF, M. P. BDI Agents: From Theory to Practice. In: LESSER, V. (Ed.). *System*. San Francisco, 1995. p. 312–319. <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.37.7970>.

RAY, P. et al. Distributed Autonomic Management : An Approach and Experiment towards Managing Service-Centric Networks. 2008.

SALESFORCE. *SalesForce*. 2012. <http://www.salesforce.com>.

SEGALL, B. et al. Content Based Routing with Elvin4. In: *Proc AUUG 00*. [S.l.]: Citeseer, 2000. v. 61, n. 7.

SMITH, J. E.; NAIR, R. The architecture of virtual machines. *Computer*, IEEE, v. 38, n. 5, p. 32–38, 2005. ISSN 00189162. <http://ieeexplore.ieee-.org/lpdocs/epic03/wrapper.htm?arnumber=1430629>.

SOTOMAYOR, B. et al. Virtual Infrastructure Management in Private and Hybrid Clouds. *IEEE Internet Computing*, IEEE Educational Activities Department, v. 13, n. 5, p. 14–22, 2009. ISSN 10897801. <http://ieeexplore-.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5233608>.

SUGENO, M.; YASUKAWA, T. A fuzzy-logic-based approach to qualitative modeling. *IEEE Transactions on Fuzzy Systems*, v. 1, n. 1, p. 7, 1993. ISSN 10636706. <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper-.htm?arnumber=390281>.

SUGERMAN, J.; VENKITACHALAM, G.; LIM, B.-H. Virtualizing I/O Devices on VMware Workstation's Hosted Virtual Machine Monitor. In: *USENIX Annual Technical Conference*. USENIX Association, 2001. v. 7, n. 2, p. 1–14. ISBN 188044609X. ISSN 15407993. <http://portal.acm.org/citation.cfm?id=715774>.

SYCARA, K. P. Multiagent Systems. p. 79–93, 1998.

TESAURO, G. et al. A Multi-Agent Systems Approach to Autonomic Computing. *Organization*, IEEE Computer Society, p. 464–471, 2004. <http://portal.acm.org/citation.cfm?id=1018780>.

TEWARI, V. Standards for Autonomic Computing. *Intel Technology Journal*, v. 10, n. 04, 2006. ISSN 1535864X. <http://www.intel.com/technology/itj-/2006/v10i4/3-standards/1-abstract.htm>.

Thijs Metsch, S. M. Open Cloud Computing Interface - Use cases and requirements for a Cloud API. 2009.

Trend Micro Inc. *Cloud Security Survey Global Executive Summary*. 2011. <http://us.trendmicro.com/imperia/md/content/us/trendwatch/cloud/survey-.pdf>.

VMWARE. *VMWare*. 2012. <http://www.vmware.com>.

WIKIPEDIA. *WIKIPEDIA, A Enciclopedia Livre. Acordo de Nivel de Servico*. 2011. <http://pt.wikipedia.org/wiki/AcordodeNiveldeServico>.

WIKIPEDIA. *WIKIPEDIA, The Free Encyclopedia. Utility Computing*. 2011. <http://en.wikipedia.org/wiki/Utilitycomputing>.

WUHIB, F.; STADLER, R.; SPREITZER, M. Gossip-based resource management for cloud environments. *2010 International Conference on Network and Service Management*, Ieee, p. 1–8, out. 2010. <http:/-/ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5691347>.

Y_SERIAL. *Y_Serial*. 2012. <http://yserial.sourceforge.net/>.