CÉSAR ANGONESE

BALANCEAMENTO DE CARGA DE TRABALHO EM COMPUTAÇÃO EM NUVEM BASEADO EM REDES MAGNÉTICAS VIRTUAIS

Dissertação apresentada ao Programa de Pós-Graduação em Informática da Pontifícia Universidade Católica do Paraná como requisito parcial para obtenção do título de Mestre em Informática.

CURITIBA

2012

CÉSAR ANGONESE

BALANCEAMENTO DE CARGA DE TRABALHO EM COMPUTAÇÃO EM NUVEM BASEADO EM REDES MAGNÉTICAS VIRTUAIS

Dissertação apresentada ao Programa de Pós-Graduação em Informática da Pontifícia Universidade Católica do Paraná como requisito parcial para obtenção do título de Mestre em Informática.

Área de Concentração: Sistemas Distribuídos

Orientador: Prof. Dr. Alcides Calsavara
Co-orientador: Prof. Dr. Luiz Augusto de Paula Lima Jr

CURITIBA
2012

Angonese, César
Balanceamento de carga de trabalho em computação em nuvem baseado em redes magnéticas virtuais. Curitiba, 2012. 80p.

Dissertação – Pontifícia Universidade Católica do Paraná. Programa de Pós-Graduação em Informática.

1. Computação em Nuvem 2. Balanceamento de Carga 3. Redes Magnéticas Virtuais. I.Pontifícia Universidade Católica do Paraná. Centro de Ciências Exatas e de Tecnologia. Programa de Pós-Graduação em Informática II-t

Aos meus pais Idalino (in memória) e Hilda Angonese, a minha esposa Neusa e filhos Cesinha e Giordanna.

Agradecimentos

Primeiramente agradecer a Deus pelo dom da vida, aos meus familiares, principalmente minha esposa Neusa, pelo incentivo, apoio e carinho a mim dedicados durante esta pesquisa e paciência nos períodos em que estive ausente; aos meus filhos Cesinha e Giordanna que souberam e aceitaram a ausência no período em que estive me dedicando aos estudos.

Agradecimento especial a minha irmã Rosângela e sua Filha Ana Paula, muitos dias estive morando em sua casa durante o período de estudos.

Aos colegas, professores e funcionários do programa de Pós-Graduação em Informática da PUCPR que me acompanharam e apoiaram.

Agradecimento muito especial ao meu orientador Prof. Dr. Alcides Calsavara pela sabedoria, esforço, dedicação e paciência durante o período em que me orientou.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas

DNS	Domain Name System
RMV	Redes Magnéticas Virtuais
SaaS	Software As a Service
IaaS	Infrastructure As a Service
PaaS	Platform As a Service
QoS	Quality of Service
IP	Internet Protocol
SO	Operating System
CRM	Customer Relationship Management
API	Application Programming Interface
EC2	Amazon Elastic Compute Cloud
SLA	Service Level Agreement
HDFS	Hadoop Distributed File System
Amazon SQL	Amazon Simple Queue Service
Amazon S3	Amazon Simple Storage Service
AMI	Amazon Machine Image
SDK	Software Development Kit
VM	Virtual Machine
NIST	National Institute of Standards and Technology
CPU	Central Processing Unit
CEO	Chief Executive Officer
Mbs	Megabits

Resumo

Recentemente, um novo paradigma de computação emergiu e mudou a forma de trabalho e hospedagem das aplicações: a computação em nuvem (do inglês *Cloud Computing*). Neste modelo, um serviço é hospedado por uma máquina virtual que é alocada em alguma máquina física do *datacenter*. Porém, uma máquina virtual é um recurso finito, o que pode comprometer o atendimento da demanda vinda por meio de requisições de usuários. Uma solução para isso é a replicação do serviço, tal que cada réplica fique alocada em uma máquina virtual distinta. Para que o conjunto de réplicas atinja o propósito de atender à demanda, é necessário haver balanceamento de carga entre essas réplicas. Existem muitos algoritmos de balanceamento de carga de trabalho e, de fato, vários são empregados em sistemas de gerenciamento de *datacenters*. A presente dissertação realiza um estudo dos atuais algoritmos de balanceamento de carga de trabalho, verificando-se que, nestes algoritmos, existe um único ponto de falha podendo tornar-se um gargalo para o desempenho do sistema. Esta dissertação apresenta como solução uma abordagem totalmente distribuída baseada em Redes Magnéticas Virtuais [CAL10], no contexto de computação em nuvem. Como resultado de análise comparativa através de experimentos, o algoritmo de Redes Magnéticas Virtuais apresenta desempenho um pouco inferior em relação aos demais algoritmos, principalmente devido ao maior custo com a troca de mensagens. Entretanto, os benefícios de ser totalmente distribuído, incluindo escalabilidade e tolerância a falhas, podem justificar seu uso em computação em nuvem.

Palavras-Chave: Balanceamento de Carga; Computação em Nuvem; Redes Magnéticas Virtuais.

Abstract

Recently, a new computing paradigm emerged and changed the work way and application hosting: the Cloud Computing. In this model, a service is hosted by a virtual machine that is placed in some *datacenter* physical machine. But, a virtual machine is a finite resource, what can compromise the attending of the demand coming from the users´ requests. A solution for this is the service replication, so that each replica is allocated in a distinct virtual machine. In order to the replica set gets the goal to answer the request, it is necessary to have a load balancing among these replicas. There are a lot of work load balancing algorithms and, in fact, several are applied to *datacenter* management systems. The present dissertation makes a study of the current work load balancing algorithms, checking that, in these algorithms there is a unique point of failure that may become a bottleneck for the system performance. This current dissertation presents as a solution, a totally distributed approach based on Virtual Magnetic Nets [CAL10] in the cloud computing context. As a comparative analysis result through experiments, the Virtual Magnetic Network algorithm presents an inferior performance in relation to the other algorithms, mainly due to the higher cost with the message exchanging. However, the benefit of being totally distributed, including scalability and failure tolerance, can justify its use in Cloud Computing.

Keywords: Load Balancing, Cloud Computing, Virtual Magnetic Network.

Capítulo 1

1 Introdução

Recentemente, um novo paradigma de computação emergiu e mudou a forma de trabalho e hospedagem das aplicações: a *computação em nuvem* (do inglês, *cloud computing*). Computação em nuvem é um modelo computacional que permite ao usuário dispor de recursos teoricamente infinitos, sem necessidade de investimento em infraestrutura própria, pagando apenas pelos recursos consumidos. A implementação desse modelo dá-se, normalmente, através de um conjunto de servidores interconectados dando a impressão ao usuário que ele está usando uma única máquina [TAN07], denominado *datacenter*, e correspondente sistema de gerenciamento. Já era vislumbrado por John McCarthy em 1960, ele dizia que as facilidades de computação seriam fornecidas para o público em geral como uma utilidade [PAR66]. Entretanto, foi o CEO, da Google Eric Schmidt, em 2006, que usou o termo computação em nuvem para descrever um modelo de negócio que fornece serviços através da Internet. Deste momento em diante, a computação em nuvem ganhou popularidade, com investimentos e pesquisa de grandes empresas como Google, Microsoft, IBM, Amazon, órgãos ligados a governos como a NASA e grande parte da comunidade científica. Uma discussão detalhada sobre computação em nuvem pode ser encontrada em [GRE09].

Uma forma importante de uso de computação em nuvem é o provimento de infraestrutura para hospedagem de serviços. Nesse modelo de computação, um serviço é hospedado por uma máquina virtual que é alocada em alguma máquina física do *datacenter*. Porém, uma máquina virtual e o correspondente servidor que a hospeda possuem recursos finitos, o que pode comprometer o atendimento da demanda vinda por meio de requisições de usuários. Uma solução para isso é a replicação do serviço, tal que cada réplica fique alocada em uma máquina virtual distinta e que o conjunto de máquinas virtuais que hospedam essas réplicas fiquem alocadas em servidores distintos.

A Figura 1.1 representa uma plataforma que fornece computação em nuvem, composto por cinco servidores com configuração homogênea interconectados, no qual se observa que:

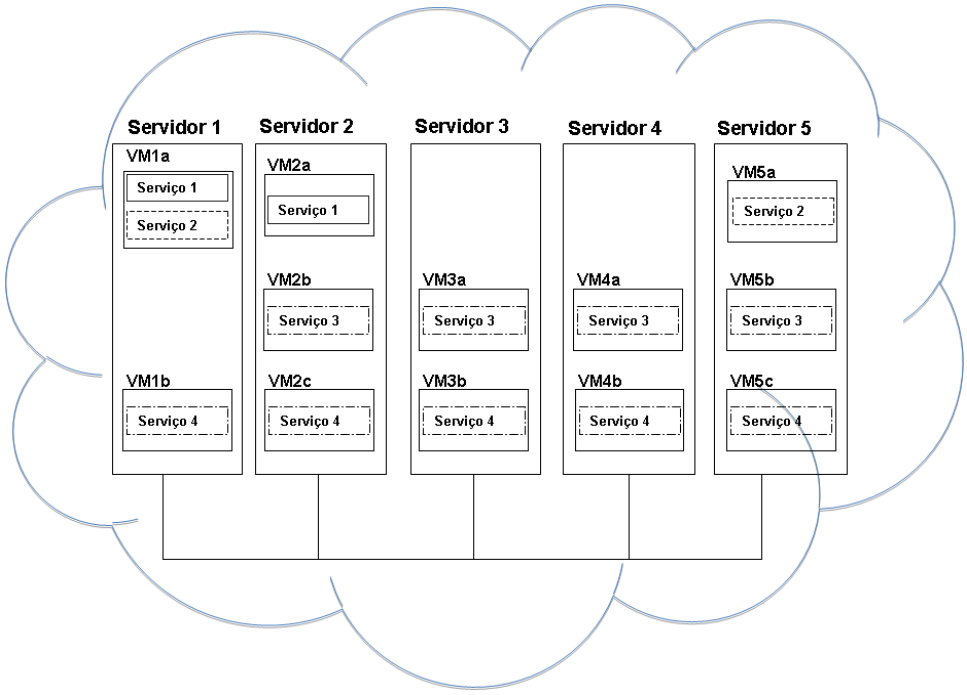

Figura 1.1 Arquitetura de uma plataforma que fornece computação em nuvem.

- O Serviço 1 é implementado por um conjunto de duas réplicas: uma hospedada na VM1a do Servidor 1 e outra na VM2a do Servidor 2;

- O Serviço 2 é implementado por um conjunto de duas réplicas: uma hospedada na VM1a do Servidor 1 e outra na VM5a do Servidor 5;

- O Serviço 3 é implementado por um conjunto de quatro réplicas: uma hospedada na VM2b do Servidor 2, uma na VM3a do Servidor 3, uma na VM4a do Servidor 4 e outra na VM5b do Servidor 5;

- O Serviço 4 é implementado por um conjunto de cinco réplicas: uma hospedada na VM1b do Servidor 1, uma na VM2c do Servidor 2, uma na VM3b do Servidor 3, uma na VM4b do Servidor 4 e outra na VM5c do Servidor 5;

O sucesso desta solução depende do bom balanceamento de carga realizado entre os componentes que processam as requisições dos usuários para que o sistema tenha um bom desempenho global. Nessa arquitetura, há três níveis de granularidade para componentes de processamento que podem ser considerados para efeito de balanceamento de carga, a saber: (*i*) servidor, (*ii*) máquina virtual e (*iii*) réplica de serviço. Entretanto, não é aplicável realizar balanceamento de carga baseado em servidor, pois, um serviço não necessariamente está disponível em todos os servidores. Da mesma forma, não é possível realizar balanceamento de

carga em VM, pois um serviço não necessariamente está disponível em todas as VM's. Portanto, o balanceamento de carga deve ser baseado no conjunto de réplicas de cada serviço.

Contudo, o balanceamento de carga baseado no conjunto de réplicas de serviços pode causar o desbalanceamento entre servidores, prejudicando o desempenho global do sistema.

A Figura 1.2 representa o balanceamento de carga realizado entre réplicas de serviços. No contexto de computação em nuvem, a tarefa de atendimento de uma requisição de usuário é, normalmente, denominada *cloudlet.* Observa-se que as réplicas de serviço estão balanceadas, mas entre os servidores há um desbalanceamento: o Servidor 1 executa seis *cloudlets* e o Servidor 2 executa apenas três. Portanto, o balanceamento entre réplicas de serviços não garante o balanceamento entre servidores.

Existem muitos algoritmos de balanceamento de carga de trabalho e, de fato, vários são tradicionalmente empregados em sistemas de gerenciamento de *datacenters*, tais como Round Robin [ZHONG02] e Least Connections, Least Response Time, Least Bandwidth, Least Packets, Custom Load, Token Method [CITRIX11], conforme descrito no Capítulo 2.

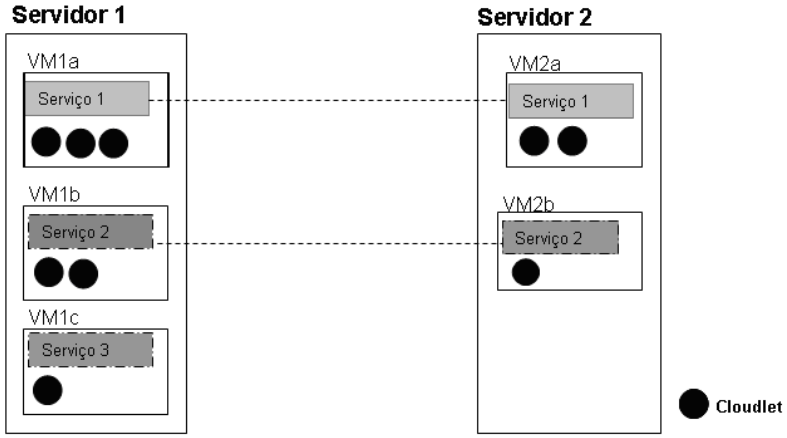

Figura 1.2 Balanceamento de carga entre réplicas de serviços.

Recentemente, um novo algoritmo para o balanceamento de carga de trabalho foi proposto em [CAL10] e [LIMA10], baseado no conceito de *redes magnéticas virtuais,* conforme descrito no Capítulo 4. Esse algoritmo difere dos tradicionais nas seguintes características:

- Enquanto os algoritmos tradicionais possuem apenas um ponto de entrada de *cloudlets*, o algoritmo baseado em redes magnéticas virtuais, por ser totalmente distribuído, permite que qualquer nó do grafo seja um ponto de entrada de *cloudlets*.

- Enquanto os algoritmos tradicionais tomam decisão baseando-se exclusivamente em métricas das réplicas de serviços (portanto, garantindo balanceamento entre réplicas de serviços), o algoritmo baseado em redes magnéticas virtuais, embora também tome decisão baseado nas métricas das réplicas de serviços, propicia balanceamento de carga entre servidores, pois as métricas das réplicas de serviços refletem métricas dos próprios servidores.

O objetivo desta dissertação é a utilização do algoritmo baseado em redes magnéticas virtuais no contexto de computação em nuvem e fazer um estudo comparativo com os algoritmos tradicionais. Para efeito de simplicidade, nesta dissertação, o algoritmo de balanceamento de carga baseado em redes magnéticas virtuais será doravante denominado *MagBalance*.

1.1 Contribuição

Esta dissertação de mestrado traz as seguintes contribuições:

a) Proposta de um modelo de balanceamento de carga para computação em nuvem baseado no algoritmo MagBalance.

b) Verificação do funcionamento do algoritmo MagBalance no contexto de computação em nuvem;

c) Avaliação de custo e do desempenho do algoritmo MagBalance no contexto de computação em nuvem;

d) Comparação entre os algoritmos de balanceamento de carga Round Robin, Least Connections e MagBalance;

A comparação entre algoritmos e a avaliação do algoritmo MagBalance foram realizadas por meio de simulação, utilizando-se do simulador CloudSim [BUY09].

Questões de tolerância a falha e de segurança são deixadas como sugestão de trabalhos futuros.

1.2 Organização

Esta dissertação está organizada da seguinte forma: no Capítulo 2 é apresentado o conceito de balanceamento de carga de trabalho e seu surgimento. Nesse capítulo, estão apresentados trabalhos relacionados, divididos em duas categorias: estáticos e dinâmicos.

Na sequência, no Capítulo 3, são apresentados conceitos de diferentes autores, modelo de serviços, vantagens de utilização de computação em nuvem e uma comparação entre computação em nuvem e computação em *grids* e, no final deste mesmo Capítulo, é descrito o modelo típico de balanceamento de carga de trabalho na computação em nuvem, isto é, algoritmos apresentam como característica um único ponto de entrada. No Capítulo 4, apresenta-se o algoritmo MagBalance, uma descrição detalhada e um exemplo de seu funcionamento. No Capítulo 5, apresenta-se uma descrição do modelo proposto, descrição do simulador (CloudSim) utilizado para experimentos e resultados dos experimentos. No Capítulo 6, conclui-se o trabalho e apresentam-se sugestões para trabalhos futuros.

Capítulo 2

2 Balanceamento de Carga

Segundo [KEN11] balanceamento de carga é criar um sistema que virtualize o trabalho dos servidores físicos que executam aqueles serviços. Uma definição mais básica é a de equilibrar a carga entre vários servidores físicos, fazendo com que eles pareçam um grande servidor para o mundo externo. Há muitos motivos para fazer isso, mas os principais podem ser resumidos em escalabilidade, alta disponibilidade e previsibilidade. A Figura 2.1 ilustra um conjunto de servidores em que não ocorre balanceamento de carga, com grande variação no uso de processador. Em seguida, a Figura 2.2 ilustra o mesmo conjunto de servidores, com sua carga de trabalho balanceada, cuja variação do uso de processador é pequena.

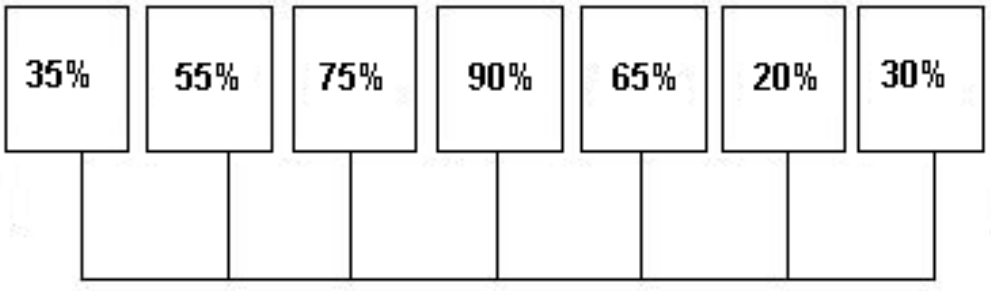

Figura 2.1 Rede servidores não balanceada

Figura 2.2 Rede servidores balanceada

A seguir alguns conceitos chaves serão definidos:

Escalabilidade: É a capacidade de adaptação fácil e dinâmica ao aumento de carga, sem impacto sobre o desempenho.

Alta Disponibilidade: É a capacidade de um *site* manter-se disponível e acessível, mesmo em caso de falha de um ou mais sistemas.

Previsibilidade: É a capacidade de ter confiança e controles na maneira como os serviços estão sendo distribuídos, com relação à disponibilidade e ao desempenho.

O balanceamento de carga surgiu da necessidade de se conectar vários servidores para gerenciar o tráfego gerado pelo acesso na Internet, sendo o primeiro a surgir o Round-Robin.

Nas seções seguintes, são apresentados alguns dos principais algoritmos relacionados a balanceamento de carga, conforme Figura 2.3, classificados nas categorias de estático e dinâmico.

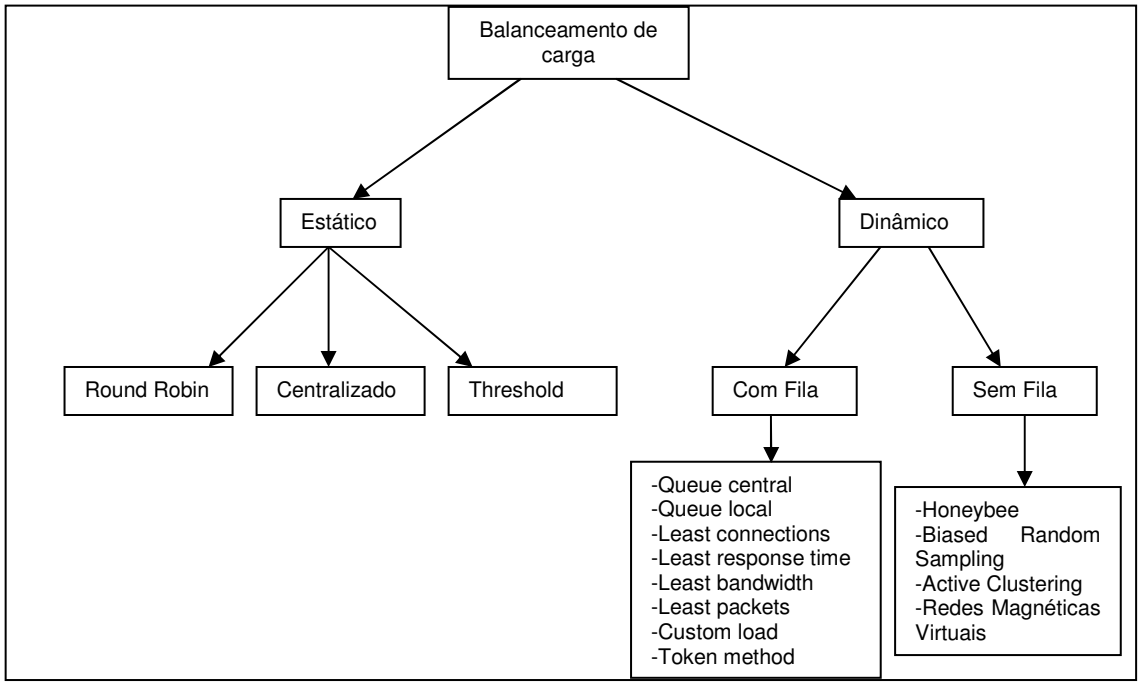

Figura 2.3 Tipos de balanceamento de carga

2.1 Balanceamento de carga estático

Neste método, o desempenho dos processadores é definido no início da execução. O objetivo do método de balanceamento de carga estático é reduzir o tempo de execução total de um programa atual, enquanto minimiza o atraso de comunicação. A desvantagem geral de

todo o esquema estático é que a alocação de um processo é feita, enquanto o processo é criado e não pode ser mudada durante a execução do mesmo.

2.1.1 Round Robin

Segundo [ZHONG02], no algoritmo Round Robin os processos são divididos uniformemente(da mesma maneira) entre todos os processadores. Cada novo processo é atribuído para um novo processador no Round Robin em ordem. A ordem de alocação dos processos é mantida localmente em cada processador, independente da alocação do processador remoto. O algoritmo Round Robin trabalha bem quando o número de processos com uso de processador for similar (processos com consumo médio de CPU parecidos).

A principal vantagem do algoritmo Round Robin é que não ocorre a comunicação interprocessos. O algoritmo Round Robin pode ter um melhor desempenho entre todos os algoritmos de balanceamento de carga para uma aplicação de propósito específico.

2.1.2 Algoritmo gerenciamento centralizado

Segundo [MCE84], no algoritmo centralizado um processador seleciona o servidor para novos processos. O gerenciador de carga seleciona o servidor para um novo processo, para que o processador de carga confirme para o mesmo nível, tanto quanto possível. Esta informação é atualizada por processadores remotos, que enviam uma mensagem cada vez que a carga deles muda.

O gerenciador de carga toma decisão de balanceamento de carga, baseado na informação de carga do sistema, permitindo a melhor decisão enquanto o processo é criado. O alto grau de comunicação interprocessos pode ser o gargalo. Neste algoritmo, é esperado executar melhor em aplicações paralelas, especialmente quando atividades dinâmicas são criadas por diferentes servidores.

2.1.3 Algoritmo Threshold

Segundo [SAN08], no algoritmo Threshold, os processos são atribuídos imediatamente depois da criação do processo no servidor. Servidores para novos processos são selecionados localmente sem enviar mensagens remotas. Cada processador mantém uma cópia privada da carga de todo sistema. A carga de um processador pode ser caracterizada por

um dos três níveis: subcarregado, médio, sobrecarregado: Dois parâmetros podem ser usados para descrever estes níveis: *t_under* e *t_upper*.

Subcarregado	load < t_under
Médio	t_under <= load <= t_upper
Sobrecarregado	load > t_upper

Inicialmente, todos os processadores são considerados estar abaixo da carga. Quando o estado da carga de um processador excede o nível de carga limite, então, ele envia uma mensagem do novo estado de carga para todos os processadores.

Se o estado local não estiver sobrecarregado, o processo é alocado localmente. Caso contrário, um processador remoto subcarregado é selecionado. Se não existir tal processador, o processo é alocado localmente. Este algoritmo apresenta uma alta comunicação inter-processos e um grande número de alocações de processos locais. Outra desvantagem do algoritmo é que todos os processos são alocados localmente quando todos os processadores remotos estão sobrecarregados, o que pode causar o aumento no tempo de execução de uma aplicação.

2.2 Balanceamento dinâmico de carga

No balanceamento de carga dinâmico o trabalho de carga é distribuído entre os processadores durante a execução, conforme descrito abaixo:

2.2.1 Balanceamento dinâmico de carga– com fila

Segundo [MAL00], o processador master atribui o novo processo para o escravo, baseado nas novas informações coletadas. Ao contrário dos algoritmos estáticos, algoritmos dinâmicos alocam os processos dinamicamente quando um processador torna-se subcarregado. Estes são armazenados em uma fila no servidor principal e alocados dinamicamente na solicitação de um servidor remoto.

2.2.1.1 Algoritmo de fila central

Segundo [WIL00], o algoritmo de fila central trabalha sobre os princípios de distribuição dinâmica. O algoritmo armazena novas atividades e requisições não realizadas

em uma fila FIFO, no servidor principal. Cada nova atividade que chega ao gerenciador de fila é inserida na fila. Então, sempre que uma requisição por uma atividade é recebida pelo gerenciador da fila, o algoritmo remove a primeira atividade da fila e a envia para quem solicitou. Caso não tenha nenhuma atividade na fila, a requisição é armazenada, até uma nova atividade estar disponível. O problema deste algoritmo é que existe um único ponto de falha, a fila central.

2.2.1.2 Algoritmo de fila local

Segundo [WIL00], a principal característica deste algoritmo é o suporte à migração dinâmica dos processos.

Inicialmente, novos processos criados no servidor principal são alocados em todos os servidores subcarregados. Daí para frente, todos os processos criados no servidor principal e todos os outros servidores são alocados localmente.

Quando o servidor fica subcarregado, o gerenciador de carga local tenta conseguir processos de servidores remotos. O algoritmo envia requisições aleatoriamente com o número de processos local para o gerenciador de carga. Quando um gerenciador de carga recebe tal requisição, compara com o número local de processos com o número recebido. Se o primeiro for maior que o último, então, alguns dos processos que estão executando são transferidos para o requisitante.

2.2.1.3 Least Connections

Este método de balanceamento de carga seleciona o servidor com o menor número de transações ativas. Transações ativas são requisições enviadas pelo cliente. Segundo [CITRIX11], este método fornece o melhor desempenho entre os algoritmos implementados no NetScaler (ativo de balanceamento de carga da Citrix). A Figura 2.4 representa a arquitetura de balanceamento de carga do algoritmo Least connections, onde se observa que: o serviço http-1 possui três transações ativas, o serviço http-2 possui 15 transações ativas e o serviço http-3 não possui nenhuma transação ativa, portanto, a chegada de novas requisições dos clientes serão distribuídas da seguinte forma: as requisições um, dois e três serão enviadas para o serviço http-3, a requisição quatro será enviada para o serviço http-1 (isto ocorre porque, quando dois ou mais serviços http apresentarem o mesmo número de transações

ativas, aplica-se o algoritmo de balanceamento de carga Round Robin) e assim sucessivamente.

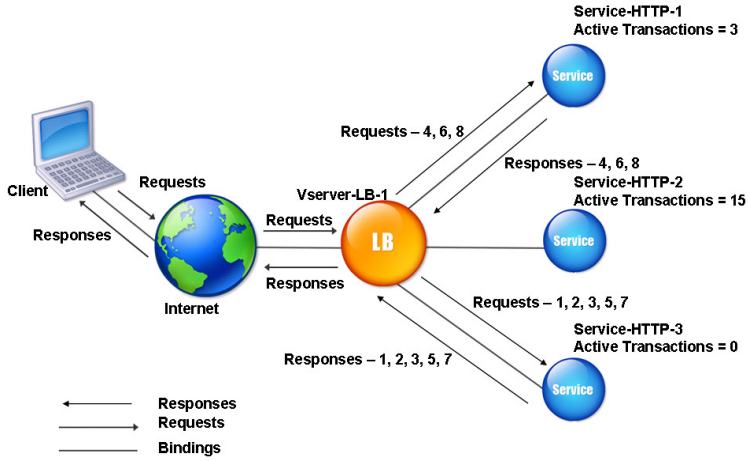

Figura 2.4 Arquitetura do balanceamento de carga Least Connections[CITRIX11].

2.2.1.4 Least Response Time

Segundo [CITRIX11], este método de balanceamento de carga seleciona o servidor com o menor número de transações ativas e a menor média de tempo de resposta. A Figura 2.5 representa a arquitetura de balanceamento de carga do algoritmo Least Response Time, onde se observa que: o serviço http-1 possui três transações ativas e tempo médio de resposta dois, portanto, fator igual a seis; o serviço http-2 possui sete transações ativas e tempo médio de resposta um, portanto, fator igual a sete, e o serviço http-3 possui fator zero, porque não possui nenhuma transação ativa. Com a chegada de requisições dos clientes, o load balancer realiza a distribuição baseado no cálculo do fator de cada um dos serviços http, onde verifica-se que as requisições um, dois e três foram enviadas para o serviço http-3; a requisição quatro foi enviada para o serviço http-1 (ocorrendo empate no cálculo do fator, aplica-se o algoritmo de balanceamento de carga Round Robin) e assim sucessivamente.

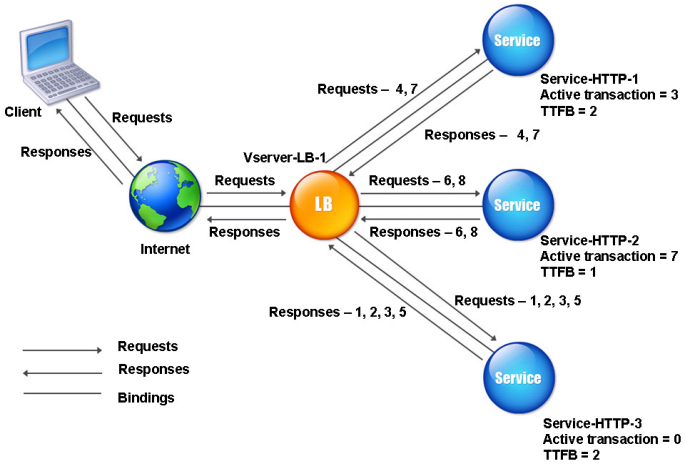

Figura 2.5 Arquitetura do balanceamento de carga Least Response Time [CITRIX11].

2.2.1.5 Least Bandwidth

Segundo [CITRIX11], este método de balanceamento de carga seleciona o servidor que estiver usando a menor quantidade de banda, medido em megabits por segundo. A Figura 2.6 representa a arquitetura de balanceamento de carga do algoritmo Least Bandwith, na qual se observa que: no serviço http-1 o parâmetro bandwith está com 3 Mbps, o serviço http-2 com 5 Mbps e o serviço http-3 com 2 Mbps, com a chegada de novas requisições dos clientes e, considerando que cada requisição requer 1 Mbps de banda, as requisições serão distribuídas da seguinte forma: requisição um enviada para o serviço http-3, requisição dois enviada para o serviço http-2 (ocorrendo empate no parâmetro bandwith, aplica-se o algoritmo de balanceamento de carga Round Robin) e assim sucessivamente.

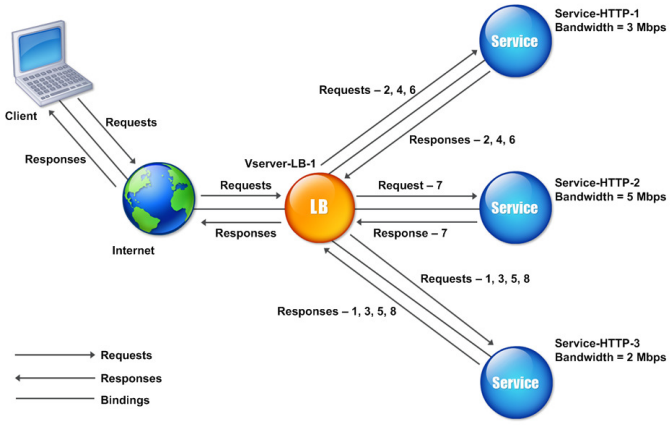

Figura 2.6 Arquitetura do balanceamento de carga Least Bandwidth [CITRIX11].

2.2.1.6 Least Packets

Segundo [CITRIX11], este método de balanceamento de carga seleciona o servidor que manuseou o menor número de pacotes transmitidos e recebidos nos últimos 14 segundos. A Figura 2.7 representa a arquitetura de balanceamento de carga do algoritmo Least Packets, na qual se observa que: no serviço http-1, o parâmetro número de pacotes está igual a três; no serviço hppt-2 está igual a cinco e, no serviço http-3 está igual a dois. Com a chegada de novas requisições dos clientes estas serão distribuídas da seguinte forma: requisição um enviada para o serviço http-3, requisição dois enviada para o serviço http-1 (ocorrendo empate no número de pacotes manuseados, aplica-se o algoritmo de balanceamento de carga Round Robin) e assim sucessivamente.

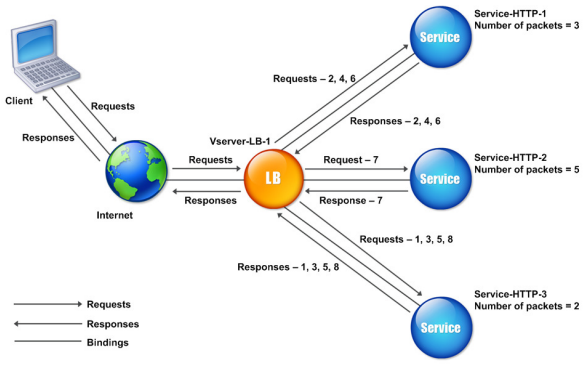

Figura 2.7 Arquitetura do balanceamento de carga Least Packets [CITRIX11].

2.2.1.7 Custom Load

Segundo [CITRIX11], este método de balanceamento de carga possui um monitor previamente configurado enviando, para cada réplica de serviço da rede, um pacote solicitando informações tais como: CPU, memória, e tempo de resposta. Baseado nestas informações, o balanceador de carga define para qual réplica de serviço deverá ser enviada a próxima requisição do cliente. A Figura 2.8 representa a arquitetura de balanceamento de carga do algoritmo Custom Load, na qual se observa que: no serviço http-1 o consumo atual de memória é igual a 20 Mbs, no serviço hppt-2 é igual a 70 Mbs e no serviço http-3 está igual a 80 Mbs. Com a chegada de novas requisições dos clientes é considerada cada uma com consumo de 10 Mbs, estas serão distribuídas da seguinte forma: requisição um, dois, três, quatro, cinco e seis enviadas para o serviço http-1, requisição sete enviada para o serviço http-

2 (ocorrendo empate no parâmetro, aplica-se o algoritmo de balanceamento de carga Round Robin) e assim sucessivamente.

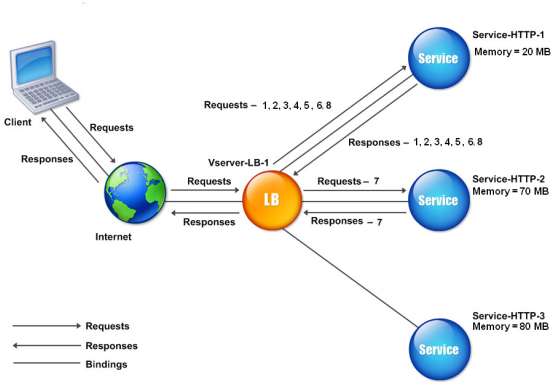

Figura 2.8 Arquitetura do balanceamento de carga Custom Load [CITRIX11].

2.2.1.8 Token Method

Segundo [CITRIX11], este método de balanceamento de carga extrai da tarefa uma ficha com o perfil que o servidor deve possuir para executá-la, então, o balanceador de carga envia a tarefa para o servidor que já executou este tipo de tarefa anteriormente. Este método também pode ser chamado de conteúdo consciente. A Figura 2.9 representa a arquitetura de balanceamento de carga do algoritmo Token Method, em que se observa que: se o Vserver-LB-1 recebe uma requisição do cliente com o token "AA", ele seleciona o serviço http-1 para processar a requisição. Se o Vserver-LB-2 recebe uma requisição diferente com o mesmo token "AA", este direciona a requisição para o serviço tcp-1.

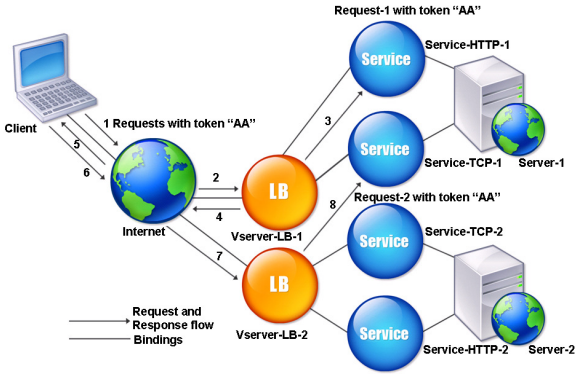

Figura 2.9 Arquitetura do balanceamento de carga Token Method [CITRIX11].

2.2.2 Balanceamento de carga dinâmico – sem fila

O foco deste algoritmo de balanceamento de carga esta na distribuição uniforme das tarefas em tempo de execução conforme descrito abaixo:

2.2.2.1 Honeybee

Segundo [NAK04], o algoritmo de balanceamento de carga Honeybee está baseado no trabalho das abelhas em uma colméia. O autor considera tal cenário altamente aplicável para computação em nuvem e adoção no paradigma "pago pelo uso" (computação de utilidade). Abelhas são enviadas para procurar recursos adequados de comida e quando encontram, retornam para a colméia e anunciam usando um quadro de avisos para conhecimento da colméia como em uma "dança de movimento". Os anúncios dos recursos de comida podem ser derivados da quantidade ou qualidade do néctar colhido ou sua distância da colméia.

2.2.2.2 Amostragem aleatória tendenciosa

Segundo [ABU08], inicialmente a rede é construída com nós virtuais para representar cada nodo servidor, com cada grau do mapeamento para o servidor com recursos livres ou algumas medidas desejáveis. Como tal, um número interno de arestas limites serão criadas, conectado a partir de nós selecionado aleatoriamente. Esta abordagem cria um sistema de rede que fornece uma medida inicial de status de disponibilidade inicial que melhora o uso dinâmico e alocação de trabalho.

Bordas dinâmicas são usadas para direcionar o procedimento de alocação de carga exigido para o esquema de balanceamento. Quando um nodo executa um novo trabalho, ele remove uma aresta de entrada; diminuindo seu grau e indicando que a disponibilidade de recursos é reduzida. Ao contrário, quando um nodo completa uma tarefa, segue um processo para criar uma nova aresta interna; indicando disponibilidade de recursos. Num estado sólido, a taxa em que tarefas chegam se iguala a taxa em que as tarefas são finalizadas.

2.3 Conclusão

Neste capítulo foram descritos alguns dos principais algoritmos de balanceamento de carga que podem ser aplicados em computação em nuvem. Dentre os algoritmos descritos,

optou-se em comparar, através de simulação, o modelo proposto nesta dissertação com os algoritmos Round Robin e Least Connections, motivo pelo qual o primeiro é um dos mais utilizados entre todos os algoritmos de balanceamento de carga e o segundo conforme descrito por fabricante [CITRIX11] de equipamento disponível no mercado é o que apresenta a melhor performance entre os algoritmos implementados em seu equipamento.

Como desvantagem os algoritmos de balanceamento de carga descritos acima apresentam de alguma forma um controle centralizado, a presente dissertação apresenta um algoritmo de balanceamento de carga totalmente distribuído.

Capítulo 3

3 Computação em Nuvem

Segundo [VEC09], com o avanço da sociedade humana moderna, serviços básicos e essenciais são quase todos entregues de uma maneira completamente transparente. Serviços de utilidade tais como água, gás e eletricidade tornaram-se fundamentais para a realização de nossa vida diária e são explorados através do pagamento baseado no uso. As infraestruturas existentes permitem entregar tais serviços em qualquer lugar e a qualquer hora, de forma que possamos, simplesmente, acender a luz, abrir a torneira ou usar o fogão. O uso desses serviços é, então, cobrado de acordo com as diferentes políticas para o usuário final. Recentemente, a mesma ideia de utilidade vem sendo aplicada no contexto da informática e uma mudança consistente, neste sentido, é realizada com a disseminação de computação em nuvem.

Segundo [ALE08], computação em nuvem se refere, essencialmente, à ideia de utilizarmos, em qualquer lugar e independente de plataforma, as mais variadas aplicações através da Internet com a mesma facilidade de tê-las instaladas em nossos próprios servidores.

No tabela 3.1 estão várias definições interessantes de computação em nuvem propostas por muitos experts, o que nos dá uma ideia clara dos diferentes conceitos.

Tabela 3.1 Definição de Computação em Nuvem

Autor/Referência	Ano	Definição
Klems [KLEMS09]	2009	Pode-se escalar sua infraestrutura sob demanda dentro de minutos ou mesmo segundos, em vez de dias ou semanas, desta forma, evitando subutilização (servidores lentos) e super-utilização de recursos em casa.
A. Marino [ALEX09]	2009	Computação em nuvem é o uso de tecnologias baseadas na internet para provisão de serviços.
R. Buyya [BUY09]	2009	Uma nuvem é um tipo de sistema distribuído e paralelo consistindo de uma coleção de servidores interconectados e virtualizados, que são dinamicamente provisionados e apresentados como um ou mais recursos de computação únicos baseados em acordos em nível de serviço, estabelecidos através de negociação entre fornecedores de serviços e clientes.
J. Kaplan [JER08]	2008	Um amplo array de serviços baseados em web auxilia os usuários a obter uma grande variedade de capacidades funcionais numa base pago pelo uso, que previamente exige tremendos investimentos em hardware/software e adquirir habilidade profissional. Computação em nuvem é a realização da mais antiga ideia da utilidade da computação sem a complexidade técnica ou complicadas preocupações com desenvolvimento.
B. Kepes [JER08]	2008	Computação em nuvem é uma mudança de paradigma infraestrutural que habilita a ascensão de SaaS. Ela é um grande array de serviços baseados em Web permitindo usuários obter uma grande variedade de capacidades funcionais na base pago pelo uso que previamente exigia tremendos investimentos em hardware e software e exigia habilidade profissional.
k. Sheynkman	2008	Nuvem esta focada em fazer a camada de hardware consumir a

[JER08]		capacidade de computação e armazenagem sob demanda. Este é um importante primeiro passo, mas para companhias garantirem a potência da nuvem, aplicações completas em infraestruturas necessitam que sejam facilmente configuradas, desenvolvidas, escaladas dinamicamente e gerenciadas neste ambiente de hardware virtualizado.
B. Martin [JER08]	2008	Computação em nuvem engloba qualquer serviço pago para usar ou baseado em assinatura, em tempo real sobre a Internet.
Vaquero [VAQ09]	2009	Nuvem é um grande pool de facilidades de recursos virtualizados acessíveis e disponíveis (tais como hardware, plataformas de desenvolvimento e/ou serviços). Estes recursos podem ser dinamicamente reconfigurados para ajustar para uma carga variável, permitindo também uma ótima utilização de recursos. Este pool de recursos é tipicamente explorado pelo modelo pago pelo uso, no qual, garantias são oferecidas pelo fornecedor de infraestrutura por meios customizados dos SLAs.
Nist[NIST10]	2010	Computação em nuvem é um modelo para habilitar convenientemente sob demanda acesso a redes para um pool compartilhado de recursos de computação configurados (redes, servidores, armazenagem, aplicações e serviços) que podem ser rapidamente provisionados e liberados com o mínimo de esforços de gerenciamento ou interação com o fornecedor de serviços.
Armbrust[ARM09]	2009	*Datacenter* de harware e software que fornecem serviços.

Conforme verificado na tabela acima se pode concluir que muitas são as definições de computação em nuvem e não há um consenso em torno de uma única definição, para esta dissertação e por melhor descrever o termo computação em nuvem, adota-se a definição da NIST [NIST10].

3.1 Entendendo computação em nuvem

Estamos habituados a utilizar aplicações instaladas em nossos próprios servidores, assim como a armazenar neles arquivos e dados dos mais variados tipos. No ambiente corporativo esse cenário é um pouco diferente, já que neste é mais fácil encontrar aplicações disponíveis em servidores e que podem ser acessadas por qualquer terminal com permissão de uma rede.

A principal vantagem desse modelo está no fato de que é possível, na maioria das vezes, utilizar aplicações mesmo sem acesso à Internet ou à rede. Entretanto, todos os dados gerados estarão restritos a esse servidor, exceto quando compartilhado em rede, coisa que não é muito comum no ambiente doméstico. Mesmo no ambiente corporativo, isso pode gerar algumas limitações, como a necessidade de se ter uma licença de um determinado software para cada equipamento instalado.

A evolução constante da tecnologia computacional e das telecomunicações está fazendo com que o acesso à Internet se torne cada vez mais amplo e cada vez mais rápido, e pagando muito pouco. Este cenário cria a situação perfeita para a popularização da computação em nuvem.

Com a computação em nuvem, muitos aplicativos dos usuários, assim como seus arquivos e dados relacionados, não precisam mais estar instalados ou armazenados em seu servidor, pois ficam disponíveis na “nuvem”, isto é, na Internet. Para o fornecedor da aplicação, ficam todas as tarefas de desenvolvimento, armazenamento, manutenção, atualização, backup, escalonamento, etc. O usuário não precisa se preocupar com mais nada disso, apenas com o acessar e usar.

3.2 Modelos de serviços

Segundo [ARM09], encontramos três diferentes modelos de serviços em computação em nuvem que são importantes porque definem o padrão da arquitetura, conforme mostrado na Figura 3.1 e explicado nas seções seguintes. O presente trabalho se encaixa no modelo IaaS.

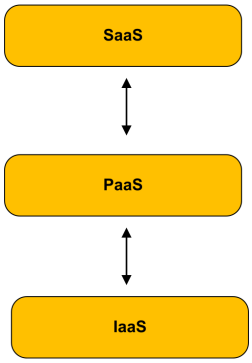

Figura 3.1 Modelos de Serviços. Fonte [ARM09].

3.2.1 Software como Serviço (SaaS)

O modelo Saas oferece softwares com propósitos específicos que estão disponíveis para os usuários através da Internet. Os softwares são acessados através dos dispositivos dos usuários como num navegador Web. No modelo SaaS, o usuário não controla ou administra a infraestrutura subjacente, incluindo rede, servidores, sistemas operacionais, armazenamento, ou mesmo características da aplicação, exceto suas configurações específicas. Desta forma, os desenvolvedores ficam concentrados em novas aplicações e não se preocupam com infraestrutura, desenvolvendo mais rapidamente os softwares.

Como os softwares estão distribuídos na Web, eles podem ser acessados pelos usuários de qualquer lugar e a qualquer momento permitindo, desta forma, uma maior integração entre as unidades de uma empresa ou usuários que compartilham a mesma plataforma de serviços. Assim, os novos recursos desenvolvidos para a aplicação podem ser incorporados de uma maneira transparente e rápida, sem que os usuários percebam estas ações, tornando a evolução e atualização transparente dos sistemas, não exigindo do desenvolvedor que tenha que atualizar diferentes servidores para que os usuários tenham acesso às novas implementações de softwares. O modelo SaaS reduz os custos, pois é dispensada a aquisição de licenças de softwares. Como exemplos de SaaS, pode-se destacar os serviços de CRM on-line da Salesforce [SAL09] e o Google Docs [CIU09].

3.2.2 Plataforma como Serviço (PaaS)

O modelo PaaS disponibiliza uma infraestrutura de alto nível de integração para implementar e testar aplicações desenvolvidas para utilização da computação na nuvem. O usuário não administra ou controla a infraestrutura subjacente, incluindo rede, servidores,

sistemas operacionais ou armazenamento, mas o controle sobre as aplicações implantadas e, possivelmente, as configurações de aplicações hospedadas nesta infraestrutura. O modelo PaaS fornece sistema operacional, linguagens de programação e ambientes de desenvolvimento para aplicações, ferramentas, estas, úteis na fase de desenvolvimento dos softwares e sua implementação, facilitando a integração entre os desenvolvedores.

Em geral, os ambientes são escaláveis para os desenvolvedores, e devem aceitar algumas restrições sobre o tipo de software que se pode desenvolver, desde limitações que o ambiente impõe na concepção das aplicações até a utilização de bancos de dados do tipo chave-valor, ao invés de bancos de dados relacionais. Do ponto de vista do negócio, o modelo PaaS permitirá aos usuários utilizarem serviços de terceiros, aumentando o uso do modelo de suporte, no qual os usuários se inscrevem para solicitações de serviços de TI ou de resoluções de problemas pela Web. Com isso, pode-se descentralizar certa carga de trabalho e responsabilidades nas equipes de TI nas empresas. Como exemplos de modelos PaaS podemos destacar Google App Engine [CIU09] e Aneka [VEC09].

3.2.3 Infraestrutura como serviço (IaaS)

O modelo IaaS é responsável em disponibilizar toda a infraestrutura necessária para o PaaS e o SaaS. O principal objetivo do modelo IaaS é tornar mais fácil e acessível o fornecimento de recursos, tais como servidores, rede, armazenamento e outros recursos de computação, fundamentais para construir um ambiente de aplicação sob demanda, que podem incluir sistemas operacionais e aplicativos. Algumas das características do modelo IaaS são:

- Interface única para administração da infraestrutura;
- API para interação com servidores, switches, balanceadores, roteadores;
- Suporte para adição de novos equipamentos de forma simples e transparente;

O termo IaaS refere-se a uma infraestrutura computacional baseada em técnicas de virtualização de recursos de computação. Conforme a necessidade da aplicação, pode-se adicionar novos recursos de computação, como servidores, de uma forma transparente e dinâmica, alocando-se estes recursos e liberando conforme a demanda.

Como um exemplo de IaaS, podemos citar: Amazon EC2 [ROB08] e o Eucalyptus [LIU07].

Conforme considerado por [ARM10], os termos IaaS e PaaS, juntos, são definidos como computação de utilidade porque a linha entre baixo nível de infraestrutura e alto nível

de plataforma não é puro. Acredita-se que os dois (IaaS e PaaS) são mais semelhantes do que diferentes.

3.3 Modelo de Implantação

Conforme definido por [NIST09], existem quatro modelos bem distintos de implantação de ambientes de computação em nuvem. Estes modelos podem ser distintos baseados no negócio da empresa, no tipo de informação e no nível de visão desejado. Algumas empresas podem desejar que nem todos os usuários possam acessar e utilizar determinadas informações ou recursos do seu ambiente de computação em nuvem, então, surge a necessidade de ambientes mais restritos. Podemos dividir a computação em nuvem em: público, privado, híbrido e comunidade.

3.3.1 Modelo de implantação público

Neste modelo de implantação, na computação em nuvem pode ser aplicada alguma técnica de autenticação e autorização, sendo que toda infraestrutura pode ser acessada por qualquer usuário que conheça de antemão a localização do serviço.

3.3.2 Modelo de implantação privado

Neste modelo de implantação, toda infraestrutura de nuvem será utilizada apenas por uma empresa ou organização, aplicando-se políticas de autenticação e autorização para acesso aos serviços, esta nuvem pode ser administrada pela própria empresa ou por terceiros.

3.3.3 Modelo de implantação híbrido

Neste modelo de implantação, pode existir uma composição de duas ou mais nuvens, os quais podem ser: privadas, públicas ou comunidade e que permitem a portabilidade de dados ou aplicações.

3.3.4 Modelo de implantação comunidade

Neste modelo de implantação, a computação em nuvem pode ser compartilhada por um pool de empresas, na qual compartilham dados de interesse pelas empresas. Nesta nuvem, podem-se aplicar políticas de autenticação e segurança conhecidas por todos os participantes da mesma. A mesma também pode ser administrada por uma das empresas participantes da comunidade ou por uma empresa terceira.

3.4 Papéis na computação em nuvem

Segundo [MAR09], para entender melhor a computação em nuvem pode-se separar os usuários e fornecedores do modelo de acordo com sua participação, conforme mostrado na Figura 3.2.

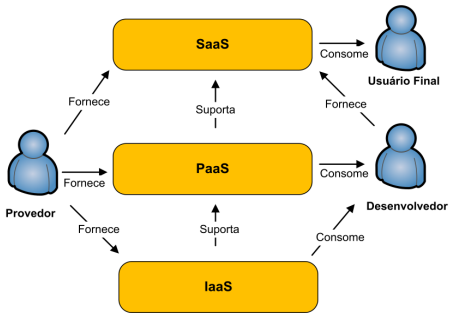

Figura 3.2 Papéis na computação em nuvem. Fonte [MAR09].

O papel do provedor é de fornecer, monitorar e gerenciar toda a infraestrutura para a solução de computação em nuvem, participando nos três níveis do modelo de serviços. O desenvolvedor utiliza os recursos de infraestrutura e plataforma fornecidos e desenvolve aplicações e serviços para o usuário final na camada de SaaS. Nesta distribuição de papéis, fica bem claro qual é a responsabilidade de cada participante na computação em nuvem, sendo o provedor responsável em manter, disponibilizar e gerenciar nas três camadas.

3.5 Vantagens de utilizar computação em nuvem

Na maioria dos casos o usuário pode acessar determinadas aplicações, independentemente do seu sistema operacional ou de hardware.

O usuário não precisa se preocupar com a estrutura para executar a aplicação: hardware, procedimentos de backup, controle de segurança, manutenção e etc, ficam a cargo do fornecedor do serviço.

Compartilhamento de dados e trabalho colaborativo tornam-se mais fáceis, uma vez que todos os usuários acessam as aplicações e os dados do mesmo lugar (nuvem).

Alta disponibilidade, já que, se um servidor parar de funcionar, os demais que fazem parte da estrutura continuam a oferecer o serviço.

Controle de gastos, já que muitas aplicações em computação em nuvem são gratuitas, e quando é necessário pagar, o usuário somente paga em relação aos recursos que usar ou ao tempo de utilização. Não será necessário pagar por uma licença de uso integral, tal como acontece no modelo tradicional de fornecimento de software.

Escalabilidade, quando necessário acrescentar mais servidores para atender à demanda de um determinado serviço, é responsabilidade do fornecedor do serviço e deve estar detalhado no SLA.

Em vez de comprar, instalar e operar seus próprios sistemas, por exemplo, a empresa pode deixar que o provedor da nuvem faça isso por ela. Além disso, os clientes pagam apenas pelos recursos de computação e armazenamento que usarem, em vez de manter um grande conjunto de servidores apenas para os picos de carga. E se forem escritas corretamente, as aplicações podem escalar, aproveitando os enormes *datacenters* oferecidos pelos provedores de nuvem.

A nuvem visa fornecer alta disponibilidade e elasticidade, sendo composta de cinco características essenciais [MELL09]:

1. Auto-Atendimento: o consumidor configura cada recurso computacional conforme sua necessidade, sem exigir interação humana com os provedores de serviço.

2. Amplo acesso à rede: os recursos são disponibilizados na rede e acessados através de mecanismos padronizados. Isto possibilita o uso em diferentes plataformas (ex. celulares, notebooks, palms, smart phones etc.).

3. Pool de recursos: os recursos computacionais do provedor são agrupados. Isto permite servir múltiplos consumidores em um modelo multi-inquilino (multi-tenant). Ou seja, os recursos físicos e virtuais são distribuídos e ou redistribuídos dinamicamente de acordo com a demanda do consumidor.

4. Elasticidade: os recursos podem ser fornecidos rapidamente e, em alguns casos, automaticamente. A quantidade de recursos disponibilizados passa para o consumidor a impressão de que a nuvem possui uma infraestrutura ilimitada.

5. Medição no uso dos serviços: a nuvem controla e otimiza o uso de recursos fornecendo métricas de acordo com o tipo de serviço que está sendo fornecido. Tanto o provedor quanto o consumidor podem monitorar e controlar a utilização dos recursos.

3.6 Desvantagens de utilizar computação em nuvem

Existe uma grande preocupação no uso da computação em nuvem no que diz respeito à segurança e privacidade das informações disponibilizadas em nuvem. Ao utilizar os serviços de um *datacenter* que, disponibiliza toda infraestrutura necessárias de nuvem para ser utilizada, o usuário estará entregando todos os seus dados para serem cuidados por outra empresa, isto causa certa insegurança se for comparado com o modelo tradicional em que o usuário é responsável por manter e guardar seus próprios dados, isso pode causar uma sensação de vulnerabilidade.

Pela facilidade de acesso aos dados disponibilizados em uma infraestrutura de computação em nuvem existe uma grande preocupação com o roubo destas informações.

Outra desvantagem do uso da computação em nuvem é a migração de serviços entre *datacenters* de diferentes fornecedores, como não existe um SLA entre *datacenters*, cada um disponibiliza seus serviços baseados na sua própria infraestrutura de hardware e software, dificultando desta forma a migração de serviços entre infraestruturas diferentes.

3.7 Comparação entre computação em Grade e Nuvem

Uma fonte de confusão ao redor do conceito de computação em nuvem está relacionada com o conceito de computação em Grade. Esta distinção não está clara talvez porque Grade e Nuvem compartilhem visões similares: redução de custos e incremento de flexibilidade e confiança. Antes de mostrar uma comparação entre computação em nuvem e Grade será necessário definir Grade. Em 2002, Ian Foster [ARM09], propôs uma definição de Grade como: “um sistema que coordena recursos que não estão sujeitos ao controle centralizado, usando padrões abertos, interfaces e protocolos de propósitos gerais para entrega de serviços de qualidade não trivial”.

A Tabela 3.2 mostra uma comparação das características entre computação em Grade e computação em nuvem.

Tabela 3.2 Comparação entre características de Grade e Nuvem, fonte [VAR09].

Característica	Grade	Nuvem
Compartilhamento de recursos	Colaboração	Recursos atribuídos não são compartilhados
Recursos Heterogêneos	Agregação de recursos heterogêneos	Agregação de recursos heterogêneos
Virtualização	Virtualização de dados e recursos de computação	Virtualização de hardware e plataformas de software
Segurança	Segurança através de delegação de credencias	Segurança através de isolamento
Serviços de alto nível	Alto nível de serviços	Ainda não tem alto nível de serviço definido
Arquitetura	Orientado a serviço	Usuário escolhe arquitetura
Dependência de software	Aplicação de software depende do domínio	Aplicação de software independe do domínio
Sensibilização de plataforma	O cliente de software deve estar habilitado em Grade	O software trabalha num ambiente customizado.
Fluxo de trabalho de software	Aplicações exigem um serviço pré-definido de fluxo de trabalho de software	Fluxo de trabalho de software não é essencial para a maioria das aplicações
Escalabilidade	Escalabilidade de nós e sites	Nós, Sites e hardware escaláveis
Auto-gerenciamento	Reconfiguráveis	Reconfiguráveis e auto conciliador
Grau centralização	Controle descentralizado	Controle centralizado (até agora)
Usabilidade	Difícil de gerenciar	Convivialidade
Padronização	Padronização e inter-operabilidade	Falta de padrões para inter-operabilidade
Acesso usuário	Acesso transparente para	Acesso transparente para usuário

	usuário final	final
Modelo pagamento	Rígido	Flexível
Garantia de QoS	Suporte limitado, frequentemente somente melhor esforço	Suporte limitado, centrada em disponibilidade e tempo.

Segue abaixo uma descrição para melhor explicar as características da tabela acima.

Compartilhamento de recursos: Grade melhora o compartilhamento de recursos através das organizações, enquanto nuvem fornece recurso que os provedores de serviços exigem sob demanda;

Recursos Heterogêneos: Ambos os modelos suportam agregação de recursos heterogêneos de hardware e software;

Virtualização: Serviços de Grade são fornecidos com interfaces que escondem a heterogeneidade dos recursos abaixo, computação em nuvem adiciona virtualização de recursos de harware;

Segurança: Grade está relacionado à segurança, já que habilita a isolação de ambientes. Enquanto em nuvem, cada usuário tem acesso único para seu ambiente virtualizado;

Serviços de alto nível: Grade oferece uma gama de serviços, tais como: pesquisa de meta dados, transferência de dados e outros. Enquanto nuvem ainda sofre com certa falta de serviços de alto nível, que está relacionado com o baixo nível de maturidade do paradigma;

Arquitetura, dependências e conhecimento de plataforma: Virtualização é a chave habilitadora de conhecimento da arquitetura para aplicações em nuvem;

Fluxo de trabalho de software: Grades são, essencialmente, serviços e trabalhos orientados; implicam na necessidade de realizar a coordenação do fluxo de trabalho de serviços e localização, os quais não são necessários no desenvolvimento sob demanda tais como aqueles em nuvem;

Escalabilidade e auto gerenciamento: Para o desenvolvimento de aplicações para Grade e Nuvem, os programadores estão livres em questões de escalabilidade;

Usabilidade: Nuvem é facilmente usável, enquanto Grade ainda não conquistou esta exigência;

Padronização: Em computação em Grade há muitos esforços para alcançar a padronização em ambos na interface do usuário e nas interfaces internas. Em nuvem, a interface de acesso do usuário é muito frequentemente baseado nas tecnologias padrões, tais como aquelas usadas em Grade. Entretanto, padronização de interfaces internas é ainda a principal questão;

Modelo de pagamento: Serviços de Grade são cobrados usando uma taxa fixa por serviço, de outra forma, usuários de nuvem são usualmente cobrados usando o modelo pago pelo uso;

Qualidade de serviço: Em geral, Grades não estão comprometidos para um concreto nível de QoS além do melhor esforço, por outro lado, QoS é uma característica inerente de muitas aplicações em nuvem.

Apesar das semelhanças, Grade e computação em nuvem diferem, principalmente, na arquitetura, pois em Grade os pedidos de consumidores individuais podem (e deveriam) consumir grandes frações do pool total de recursos. Em computação em nuvem, frequentemente, limita-se o tamanho de uma solicitação individual para ser uma pequena fração da capacidade total disponível no pool de recursos, porém, tem como objetivo ser escalável para atender a um grande número de consumidores simultâneos. Nuvem e Grade também compartilham problemas semelhantes, tendo dificuldades para: gerenciar grandes instalações, definir métodos para que consumidores possam procurar e interagir com novos recursos e serviços, e implementar computação paralela capaz de utilizar os recursos e serviços [FOS08].

A abordagem de Grade surgiu com o intuito de resolver problemas computacionais de grande escala, usando uma rede de máquinas comuns que compartilham recursos. Ou seja, uma infraestrutura distribuída que fornece recursos de armazenamento e processamento. Para suportar a criação de organizações virtuais, por exemplo, Grade fornecem middlewares, ferramentas e um conjunto de protocolos padrão que permitem a construção de serviços. Interoperabilidade e segurança são as principais preocupações em grid, visto que os recursos podem vir de diferentes domínios administrativos, possuírem políticas de uso globais/locais dedicadas, diversas configurações para o hardware, software e plataforma, com disponibilidade e capacidade variadas [PIN05].

3.8 Modelo típico de balanceamento de carga na computação em nuvem

A Figura 3.3 representa o modelo típico de balanceamento de carga de trabalho na computação em nuvem, onde o cliente (CLIENT) está conectado a nuvem através de uma linha que representa uma conexão (qualquer tecnologia de conexão de rede). O balanceador de carga (Load Balancer) é um equipamento físico que recebe toda *cloudlet* enviada pelos clientes e a envia para um Service que está sendo executado em uma determinada VM hospedada em um servidor da nuvem.

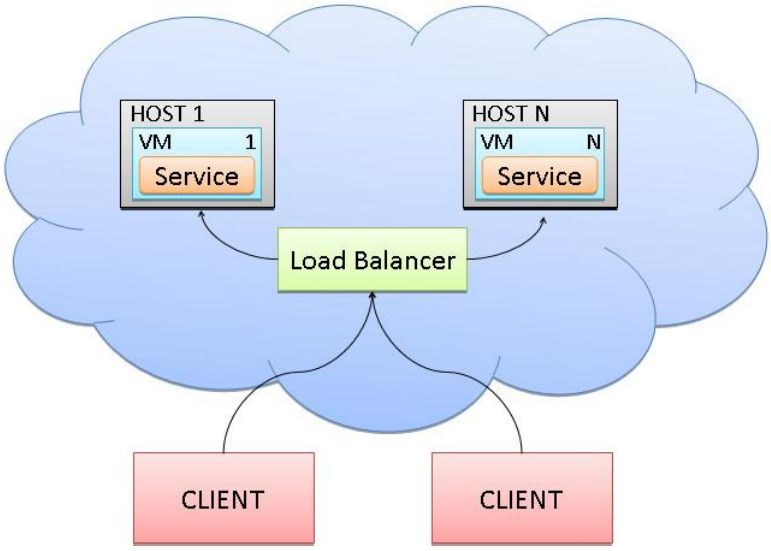

Figura 3.3 Balanceamento de carga na computação em nuvem

Neste modelo, todo serviço do cliente enviado para ser processado na nuvem passa obrigatoriamente pelo balanceador de carga que deverá decidir, através de um algoritmo, qual a melhor VM habilitada para executar aquela *cloudlet*.

Este modelo tem como característica funcionar em uma arquitetura tipicamente centralizada, pois há um único ponto de entrada, por isso encontramos problemas com relação aos seguintes aspectos de sistema:

- Tolerância a falhas: O balanceador de carga é um ponto único de falha;
- Escalabilidade: O balanceador de carga pode se tornar um gargalo para o desempenho do sistema.

A Figura 3.4 representa a arquitetura de balanceamento de carga existente no Load Balancer da Citrix, que inclui um servidor de balanceamento de carga (LB) e múltiplos servidores de aplicação, cada um contendo uma instância de um serviço. O servidor virtual recebe requisições vindas do usuário, usa o algoritmo de balanceamento de carga para

selecionar um servidor de aplicação e direciona a requisição do usuário. O papel do Monitor é verificar se o servidor de aplicação está operando corretamente e informar o NetScaler(LB).

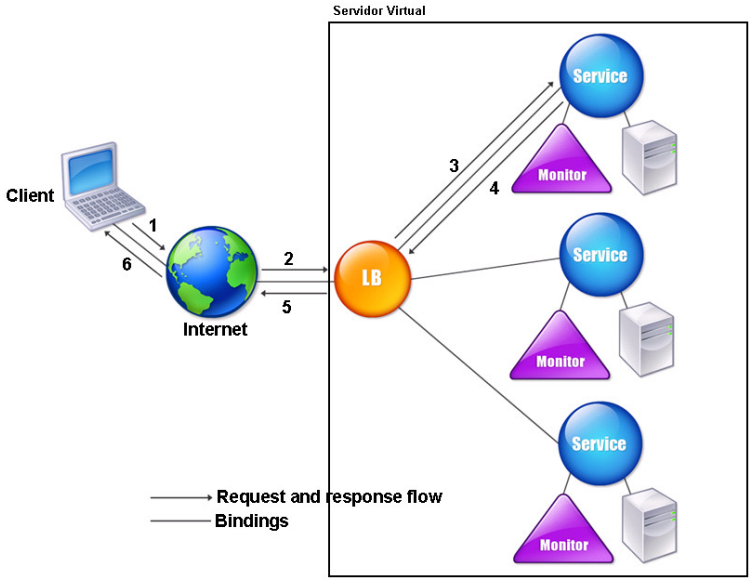

Figura 3.4 Arquitetura de balanceamento de carga – Citrix [CITRIX11].

3.9 Conclusão

Neste capítulo foram apresentados diversos conceitos de computação em nuvem, os modelos disponíveis, uma comparação entre computação em nuvem com grade e as vantagens e desvantagens da utilização de computação em nuvem.

Foi descrito um modelo típico de balanceamento de carga aplicado em computação em nuvem, onde se observa que existe um único ponto de falha e modelo totalmente centralizado.

Capítulo 4

4 Balanceamento de carga baseado em redes magnéticas virtuais: algoritmo MagBalance

O algoritmo de balanceamento de carga baseado em redes magnéticas virtuais proposto em [CAL10] e [LIMA10], que nesta dissertação é simplesmente denominado de MagBalance, pode ser compreendido através de um exemplo, conforme ilustra a Figura 4.1. Cada nó do grafo representa um servidor e estes servidores estão interligados por uma aresta que interliga os nós (a tecnologia empregada na rede é irrelevante). Para melhor ilustrar, os nós foram numerados para prosseguir na definição.

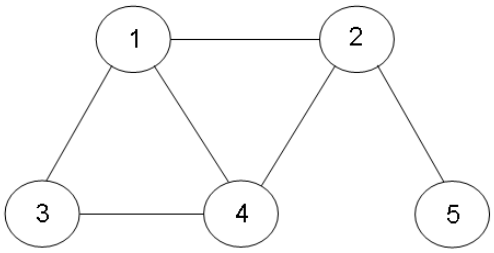

Figura 4.1 Grafo que representa uma rede com ligações físicas. Fonte [CAL10].

Os nós podem estar interligados através da rede, mas isto não significa que todos os nós possuem uma força de atração entre eles, sendo que um nó pode ter força em uma direção, mas não na direção contrária, como mostrado através do exemplo da Figura 4.2.

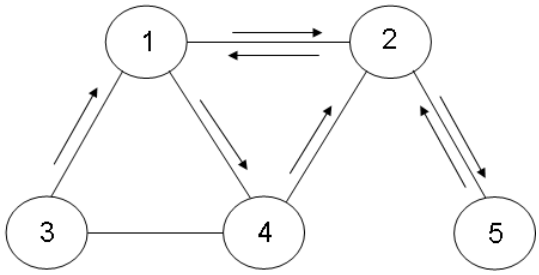

Figura 4.2 Rede com relacionamento de magnetização entre os nós. Fonte [CAL10].

O nó 3 tem uma força de atração em relação ao nó 1, isto significa que uma mensagem enviada ao nó 1 pode ser atraída e executada no nó 3. O mesmo não ocorre entre o nó 3 e 4,

porque mesmo havendo uma ligação física entre eles, não ocorre uma força de atração magnética entre os mesmos. Conforme indicado através das setas na Figura 4.3 ocorre uma força de atração magnética bidirecional entre os nós 1 e 2, significando que mensagens enviadas para qualquer um dos nós podem ser atraídas pelo outro. Quando um nó apresenta uma força de atração magnética em relação a um vizinho dizemos que este nó possui um magneto virtual e seu vizinho está dentro da força de atração de seu campo magnético virtual produzido por tal magneto.

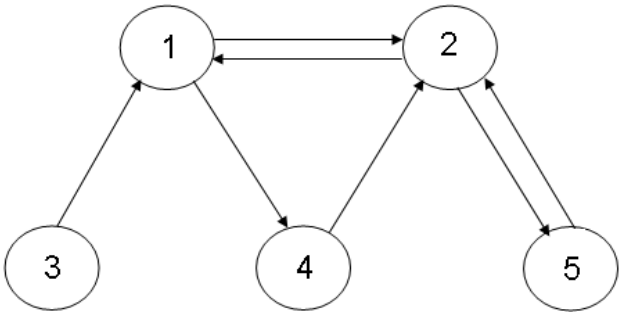

Figura 4.3 Uma rede overlay definida pelo relacionamento de magnetização. Fonte [CAL10].

Após, exemplificado o funcionamento do relacionamento por meio da força de atração dos nós, abstrai-se da rede física e mostra-se apenas a rede overlay e sua força de atração, construída a partir do relacionamento de magnetização, conforme mostrado na Figura 4.4. Observa-se na Figura 4.4 que uma mensagem foi enviada para ser processada no nó 1, baseado na força de atração dos nós esta mensagem deverá ser enviada para o nó 4, este roteamento da mensagem pode ser observado na Figura 4.5 e 4.6.

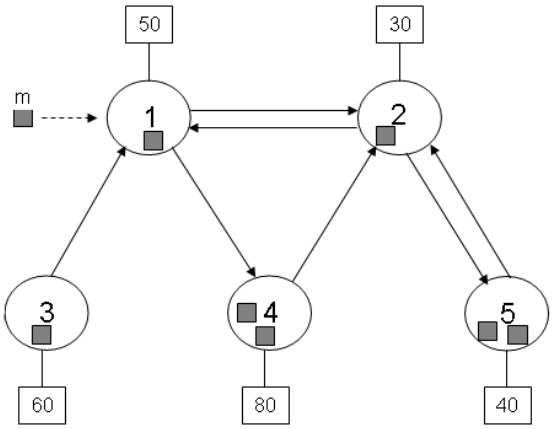

Figura 4.4 Uma rede de magnetização mostrando a força do magneto e mensagens. Fonte [CAL10].

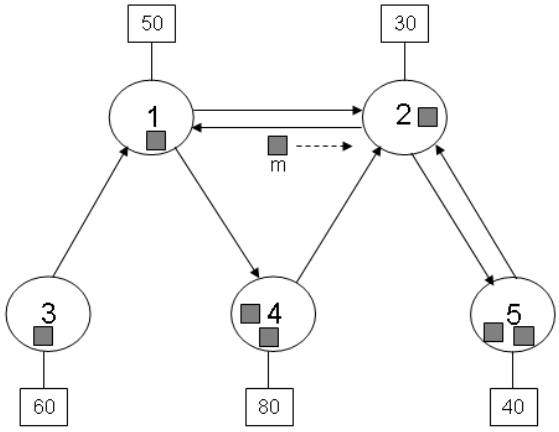

Figura 4.5 Mensagem sendo roteada para o nó com magneto mais forte. Fonte [CAL10].

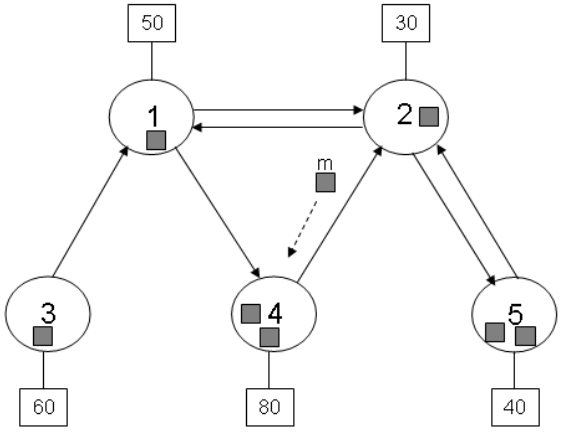

Figura 4.6 Mensagem sendo roteada para o nó pivô. Fonte [CAL10].

Conforme pode ser observado na Figura 4.7 a mensagem foi entregue para ser processada no nó 4, sendo que a força de atração deste nó sofreu uma alteração.

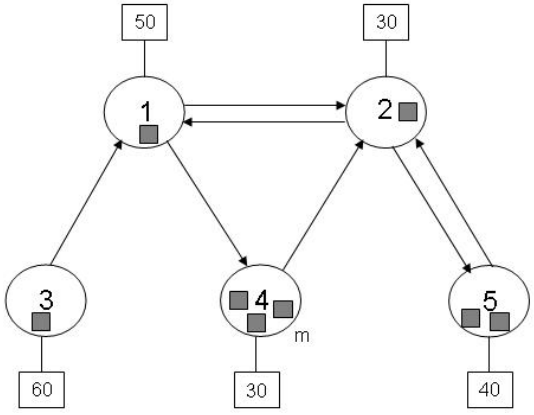

Figura 4.7 Mensagem entregue para um nó e correspondente magneto altera sua força. Fonte [CAL10].

4.1 Definição dos termos

Segue abaixo uma definição dos termos usados para descrever o algoritmo:

- x: representado em S(x), T(x), K(x), P*(x) e M(x): refere-se a um determinado nó.

- F(x): representa a força no nó. Tal força deve ser determinada conforme os objetivos da aplicação.

- S(x): representa o conjunto de nós que magnetizam x diretamente.

- T(x): Representa o conjunto de nós magnetizados diretamente por x.

- K(x): Representa o conjunto de nós conhecidos por x que, direta ou indiretamente magnetizam x. Assim, K(x) é um subconjunto de S*(x). Devido à natureza distribuída do protocolo, um nó x não conhece antecipadamente S*(x), então é descoberto conforme a troca de mensagens em x.

- P*(x): Representa o pivô global de x.

- M(x): Representa o vizinho mais próximo para rotear as mensagens para o pivô. Este nó m pertence a S(x) tal que a força do pivô global para m é a quantidade máxima total para o pivô global para os nós em S(x).

Agora, através da tabela 4.1, visualizam-se quais nós estão magnetizados (o nó pode atrair uma mensagem) e na tabela 4.2 quais nós magnetizam (a mensagem pode ser atraída). Baseado na Figura 4.3, lembrando que o próprio nó mantém um magneto para si próprio.

Tabela 4.1 Nós diretamente magnetizados por cada nó do grafo da Figura 4.3. T(x).

	Nó 1	Nó 2	Nó 3	Nó 4	Nó 5
Nó 1	●	●		●	
Nó 2	●	●			●
Nó 3	●		●		
Nó 4		●		●	
Nó 5		●			●

Tabela 4.2 Nós que magnetizam diretamente cada nó do grafo da Figura 4.3. S(x).

	Nó 1	Nó 2	Nó 3	Nó 4	Nó 5
Nó 1	●	●	●		
Nó 2	●	●		●	●
Nó 3			●		
Nó 4	●			●	
Nó 5		●			●

Como o relacionamento de magnetização é definido de forma transitiva, um nó pode ser magnetizado ou magnetizar direta ou indiretamente outro nó. Por exemplo, baseado na Figura 4.3, dizemos que o nó 1 é atraído pelo nós 3, 2 e 1, portanto, podemos dizer que o nó 2 é indiretamente atraído pelo nó 3, conforme mostrado na tabela 4.3, os nós que são direta ou indiretamente magnetizados por um determinado nó, T*(x), e na tabela 4.4, os nós que direta/indiretamente magnetizam determinado nó, S*(x).

Tabela 4.3 Nós direta ou indiretamente magnetizados por cada nó do grafo da Figura 4.3, T*(x).

	Nó 1	Nó 2	Nó 3	Nó 4	Nó 5
Nó 1	●	●		●	●
Nó 2	●	●		●	●
Nó 3	●	●	●	●	●
Nó 4	●	●		●	●
Nó 5	●	●		●	●

Tabela 4.4 Nós que magnetizam diretamente ou indiretamente cada nó do grafo da Figura 4.3. S*(x).

	Nó 1	Nó 2	Nó 3	Nó 4	Nó 5
Nó 1	●	●	●	●	●
Nó 2	●	●	●	●	●
Nó 3			●		
Nó 4	●	●	●	●	●
Nó 5	●	●	●	●	●

Para determinar a força de atração de cada nó, deve-se estabelecer a mesma, através de critérios adotados, conforme a especificidade da aplicação, podendo ser: Disponibilidade de processamento, espaço em disco, distância entre os nós e outros. A força muda com o processamento, esta nova força deverá ser informada aos nós vizinhos. Na Figura 4.4 é mostrada a força atual de cada nó, representado dentro de uma caixa e ligado ao nó por meio de uma linha continua.

Uma mensagem entregue para ser executada em um determinado nó x, deverá ser atraída para o nó que apresentar a maior força de atração naquele momento, chamado de P*(x) “pivô global”.

É assumido que o ciclo de vida de qualquer mensagem m compreende os seguintes passos:

1. m é criada no nível da aplicação e enviada para o nó x.

2. m é roteada do nó x para o nó y, onde y pertence a S*(x) “conjunto de nós que direta ou indiretamente magnetizam o nó “x”, tal que F(y) “força de y” é maior ou igual a F(k), para qualquer k que pertença a S*(x), isto é, y = P*(x);

3. m é entregue para o nível da aplicação do nó y;

4. m é propriamente tratada no nó y e, dependendo da semântica da aplicação, é eventualmente destruído ou mantido no nó y para sempre.

Na Figura 4.4 as mensagens são representadas através de um quadrado dentro do nó. Assim, os nós 1, 2 e 3 têm respectivamente uma mensagem sendo executada em cada um deles, e os nós 4 e 5 têm duas mensagens, sendo que o nó 1 está recebendo uma nova mensagem para ser executada a qual, através da força de atração do campo magnético, deverá

ser executada no nó 4, mesmo a figura mostrando que não ocorre uma força de atração entre os nós 1 e 4, mas ocorre uma força de atração do nó 4 para o nó 2, e este, uma força de atração para o nó 1, conforme mostrado na Figura 4.5 e 4.6.

Ao receber esta nova mensagem para ser executada, o nó 4 deverá mudar a sua força de atração, que deverá ser comunicado a todos os nós por ele atraídos e o novo pivô global deverá ser recalculado. Para que não ocorra um grande número de mensagens trocadas entre os nós, deverá ser comunicada sua nova força de atração somente quando a mudança de força for significativa para os demais nós; uma boa prática seria configurar um parâmetro conforme a característica da aplicação.

Para que o algoritmo de troca de mensagens entre os pontos funcione corretamente, deve ser armazenada a seguinte estrutura de dados para cada nó x:

- F(x): Força de atração de cada nó.
- S(x): Para cada nó em S(x), os seguintes campos devem estar armazenados:
    - Identificador de s.
    - O identificador para o pivô global para s, que é, P*(s).
    - A distância de P*(s) para x com respeito a rede de magnetização.
    - hora, correspondendo a hora local em s quando P*(s) foi setado pela última vez. Tal hora é útil para descartar mensagens que chegam fora de ordem.
    - Um flag para indicar se a informação do pivô global está mesmo atualizada ou obsoleta.
- T(x): Para cada nó t em T(x), a única informação armazenada é o identificador para t.
- K(x): Para cada nó k em K(x), os seguintes campos estão armazenados:
    - O identificador de k.
    - A força de k, atualmente conhecida de x, denotada como Fx(k).
    - A hora, correspondendo à hora local em k quando sua força mudou para Fx(k).
    - A distância de k para x, com respeito à rede de magnetização.
- $P^{*(x)}$: É determinado de acordo com a informação disponível em K(x).
- M(x): o vizinho mais próximo para rotear as mensagens para o pivô.

4.2 Mensagens do protocolo

Todas as mensagens trocadas para atualização do protocolo fluem de acordo com a rede de magnetização. Para manter a rede magnética atualizada, ocorre a troca de dois tipos de mensagens, a saber: troca de força e troca de pivô, que são explicadas nas seções abaixo.

4.2.1 Mensagem para mudança de força

Quando ocorre mudança de força de um nó, será enviada uma mensagem com os seguintes atributos para os nós atraídos:

- O identificador do nó x que enviou a mensagem;
- O identificador do nó destino y;
- O identificador para um nó s, que é o nó fonte cuja mudança de força é notificada pela mensagem;
- A força de s notificado pela mensagem denotado por F'(s);
- A hora correspondente à hora local em s, quando ele mudou sua força para F'(s);
- A distância de s para y, com respeito à rede de magnetização. Tal distância é útil para dois propósitos: Primeiro, para determinar o caminho de magnetização mais curto entre y e seu pivô global, para o caso de quando existir mais do que um caminho com o mesmo comprimento. Segundo, para detectar loop de mensagens que podem ocorrer devido aos ciclos na rede de magnetização.

4.2.2 Mensagem quando ocorre troca de pivô

Quando ocorre uma troca de pivô, uma mensagem será enviada para todos os nós que são magnetizados, com os seguintes atributos:

- O identificador do emitente da mensagem, nó x;
- O identificador do destino, nó y;
- O identificador para um nó p que é o pivô global para x, notificado pela mensagem;
- A hora correspondente, hora local de x quando p tornou-se o pivô global de x;
- A força de p na hora em que tal nó tornou-se o pivô global de x, denotado por F'(p);
- A hora correspondente, hora local de p quando sua força mudou para F'(p);

- A distância de p para y. Tal distância é útil para determinar o caminho mais curto e para detectar loop de mensagens.

4.3 Tratamento de mensagens

Cada nó pode receber mensagens de ambos os tipos em qualquer hora e estas mensagens devem ser tratadas unicamente com suas informações locais. Abaixo é apresentada uma descrição dos algoritmos empregados para tratar estas mensagens.

4.3.1 Algoritmo quando ocorre mudança de força

Se a mudança de força de s notificada por m(mensagem) é relevante de acordo com a hora em m e K(y) (é o conjunto de nós conhecidos por y que direta ou indiretamente magnetizam y), então:

1. registrar dentro de S(y) (conjunto de nós que diretamente magnetizam y) todos dados sobre s contidos na mensagem

2. Se a força de s baixou então

    Enviar uma mensagem de troca de força para todos t em T(y) (conjunto de nós que diretamente são magnetizados por y) com o objetivo de notificar sobre s

    Se s acontece ser o P*(y) (pivô global de y) então

        Atualiza P*(y) e M(y) de acordo com os dados em S(y)

        Se qualquer dado registrado P*(y) foi mudado então

            Envia uma mensagem de mudança de pivô para todos os nós t em T(y) para notificar sobre P*(y).

4.3.2 Algoritmo quando ocorre mudança de pivô

Uma mensagem de mudança de pivô é enviada do nodo x para o nó y com o objetivo de notificar o pivô global de p pelo nodo x que é manuseado pelo nó y da seguinte forma:

Se a mudança de pivô de x notificado por m é relevante de acordo com a hora na mensagem e S(y), então:

1. Registrar em S(y) todos os dados sobre x e p contido em m;

2. Se a força de p notificada pela mensagem é velha de acordo com a hora na mensagem e K(y) ou a distância de p para y notificado pela mensagem é maior que tal distância registrada em K(y), significando que existe uma mensagem em loop infinito

Marcar x como obsoleto em S(y)

senão

Se a força de p notificada na mensagem é relevante de acordo com a hora em m e K(y) então

2.1 Registrar dentro de K(y) todos os dados sobre p contido na mensagem

2.2 Se a força de p baixou então envia uma mensagem de troca de força para todos os nós t em T(y) para notificar sobre p

Atualizar P(y) e M(y) de acordo com dados em S(y)

Se qualquer dado registrado em P(y) foi alterado, então envia uma mensagem de troca de pivô para todos os nós T em T(y) para notificar sobre P*(y).

Capítulo 5

5 Modelo proposto

Este Capítulo apresenta o modelo proposto de balanceamento de carga na computação em nuvem.

5.1 Descrição

Na computação em nuvem um *datacenter* é constituído por uma rede de servidores com uma topologia qualquer, na qual cada servidor hospeda um conjunto de VM's, e cada VM hospeda uma instância de um serviço, conforme Figura 5.1.

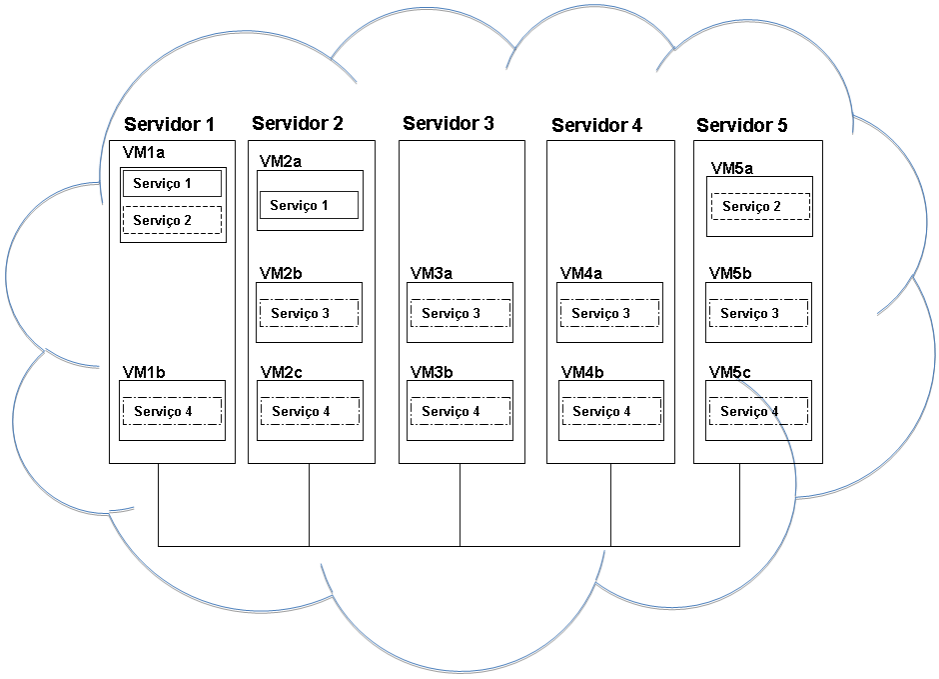

Figura 5.1 Arquitetura de um datacenter.

Uma rede overlay é definida entre todas as VM's que hospedam um mesmo serviço. A topologia desta rede overlay é um grafo aleatório, podendo ser completo ou não, e ainda conexo ou desconexo. A rede overlay é montada respeitando as conexões da rede física, isto é, só pode existir uma conexão entre dois nós(VM's) da rede overlay, se existir uma conexão física entre os servidores que hospedam estas VM's.

As Figuras 5.2, 5.3, 5.4 e 5.5 representam redes overlay construídas sobre a rede física de servidores, ilustrado na Figura 5.1. Cada rede overlay corresponde a um serviço.

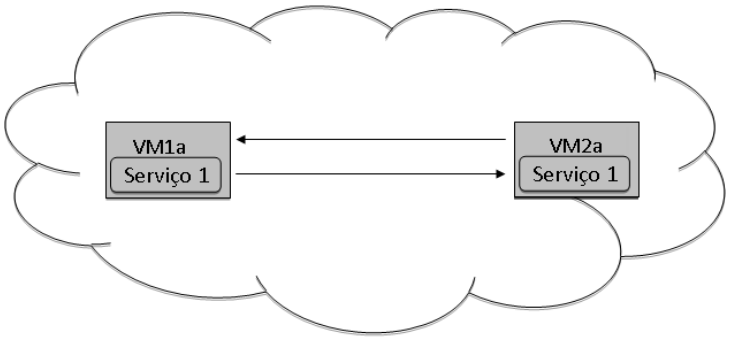

Figura 5.2 Rede overlay com grafo completo, para o Serviço 1.

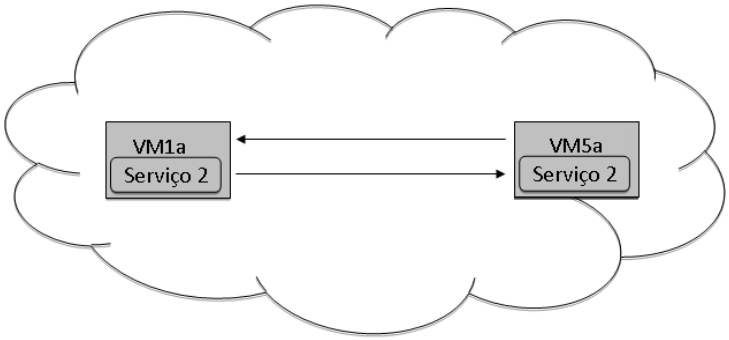

Figura 5.3 Rede overlay com grafo completo, para o Serviço 2.

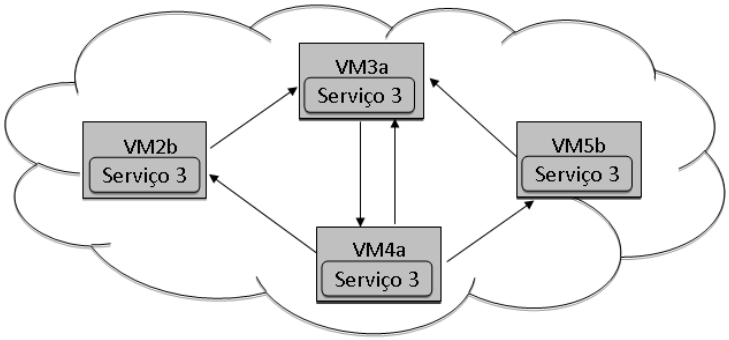

Figura 5.4 Rede overlay com grafo aleatório conexo, para o Serviço 3.

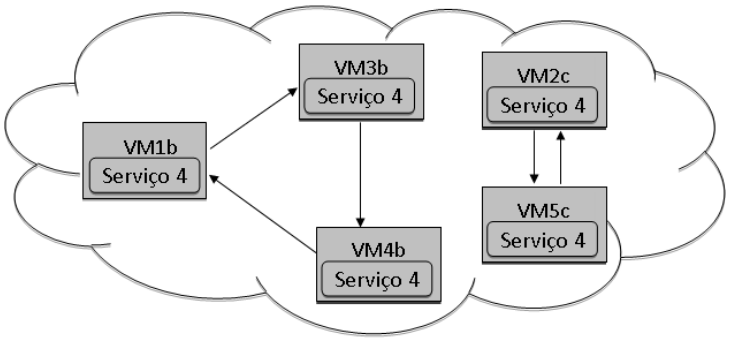

Figura 5.5 Rede overlay com grafo aleatório desconexo, para o Serviço 4.

A implementação deste modelo está baseada no algoritmo MagBalance, conforme descrito no Capítulo 4. Para o modelo proposto, as *cloudlets* serão atraídas e processadas pela VM que estiver, no momento, com a maior força entre todas as VMs disponíveis para executar a solicitação do cliente. A força da VM é calculada baseando-se nos seus recursos disponíveis, os quais refletem os recursos disponíveis no servidor. Assim sendo, o balanceamento no conjunto de réplicas de serviço não irá prejudicar o desempenho global do sistema, não desbalanceando os servidores.

Por este modelo ser totalmente distribuído, o cliente envia uma solicitação de serviço diretamente para uma VM qualquer da correspondente rede overlay. Esta VM, ao receber uma *cloudlet*, verificará qual é sua VM pivô. Na sequência, envia esta *cloudlet* para seu pivô (VM pivô atrai a *cloudlet* para ser processada). A VM pivô, ao receber a *cloudlet*, para ser processada, calcula sua nova força e, quando pertinente, comunica todas as VMs ligadas diretamente para que sejam feitas as devidas atualizações. Neste modelo fica bem claro que não existe apenas um ponto de falha, porque no momento em que um determinado servidor estiver com problemas, ele simplesmente é retirado da rede magnética, desta forma, não recebendo mais solicitações do cliente e a rede magnética deverá se re-configurar. Da mesma forma, este modelo não possui gargalo, o que favorece a escalabilidade na nuvem. O cliente pode enviar suas requisições para qualquer VM que estiver disponível no momento.

5.2 Simulação

Para realizar a simulação de balanceamento de carga em computação em nuvem, foi utilizado o simulador CloudSim. Optou-se em utilizar este simulador porque é atualmente o simulador mais utilizado no meio acadêmico. Comparar o algoritmo MagBalance com os algoritmos Least Connections e Round Robin, este último é um dos mais utilizados entre os

softwares de balanceamento de carga de código aberto e o algoritmo Least Connections foi escolhido porque é considerado pela Citrix [CITRIX11] o de melhor desempenho entre todos os algoritmos implementados em seus ativos. Outras simulações foram realizadas somente com grafos não completos, utilizando-se apenas o algoritmo MagBalance, em que se buscou o custo para montar uma rede overlay, o custo com a troca de mensagens e o tempo gasto para execução em diferentes grafos não completos que são: Grafo com duas, três e quatro arestas de atração com outros nós do grafo. Estes grafos foram gerados através de um programa desenvolvido em Delphi, que gerou os arquivos textos de forma a obter a atração entre os nós aleatoriamente.

5.2.1 CloudSim: Uma nova estrutura para modelar e simular infraestruturas e serviços de computação em nuvem.

O uso de infraestrutura real para realizar testes de um software aplicado na computação em nuvem torna-se muito caro e exige muitos recursos de pessoal com conhecimento nas mais diversas áreas da informática. Para resolver estas questões, uma alternativa é a utilização de ferramentas de simulação que abre a possibilidade de avaliação do desenvolvimento do software num ambiente em que se podem reproduzir os testes, especialmente no caso de computação em nuvem, cujo acesso para a infraestrutura deverá ser pago. Abordagens baseadas em simuladores oferecem significantes benefícios:

Do lado do desenvolvedor:

- Testar seus serviços num ambiente repetitivo e controlável, sem custos;
- Sintonizar o desempenho de gargalos antes do desenvolvimento real;

Do lado do fornecedor:

- Simulação de ambientes permitindo avaliação de diferentes tipos de recursos, avaliando cenários sob diferente distribuição de preços e cargas.

Na ausência de tais plataformas de simulação, desenvolvedores e fornecedores devem acreditar nas avaliações imprecisas e teóricas, ou nas abordagens de tentativa e erro que levam para ineficientes performances de serviço e geração de renda.

Segundo [BUY10], CloudSim é uma estrutura extensível e generalizada de simulação que habilita modelagem, simulação e experimentação da emergente infraestrutura de computação em nuvem e serviços de gerenciamento. Esta estrutura de simulação tem as seguintes características:

- Suporte para modelagem e instanciação de infraestruturas de computação em nuvem em larga escala, incluindo data centers em um nodo de computação física simples e máquina virtual Java;

- Uma plataforma de autoconteúdo para modelar data centers, negociadores de serviço, agendamento e políticas de alocação;

- Disponibilidade de virtualização de máquina, que auxilia na criação e gerenciamento de múltiplos, independentes e virtualizados serviços co-hospedados em um nodo no data center;

- Flexibilidade para trocar entre alocação de espaço compartilhado e tempo compartilhado do processamento central para serviços virtualizados.

Usando CloudSim, pesquisadores e desenvolvedores podem focar nas questões específicas do sistema que querem investigar, sem se preocupar com detalhes de baixo nível, relacionados com serviços e infraestruturas baseados em nuvem.

5.2.1.1 Arquitetura do CloudSim

A Figura 5.6 mostra as camadas de implementação do CloudSim, estrutura de software e componentes da arquitetura.

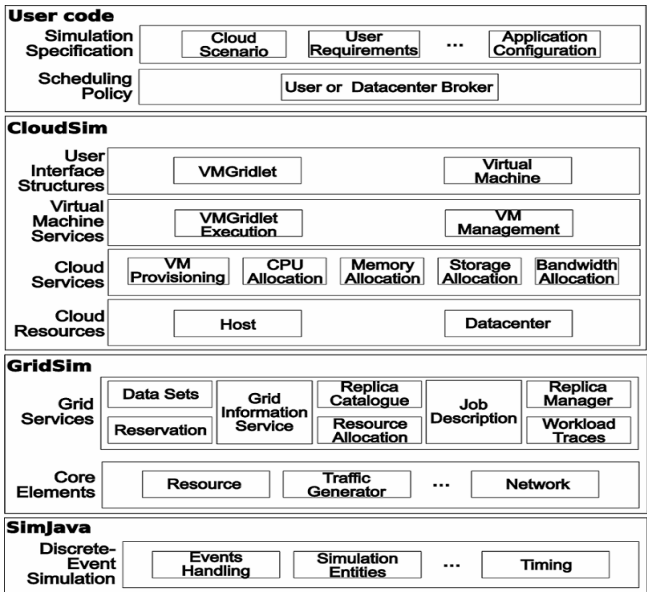

Figura 5.6 Camadas da arquitetura do CloudSim, fonte [BUY10]

Na camada mais baixa está o SimJava, máquina de simulação de eventos discretos, que implementa as funcionalidades centrais exigidas pelo mais alto nível das estruturas de

simulação, tais como filas e processamento de eventos, criação de componentes de sistema (serviços, hospedeiros, centro de dados, negociadores, máquinas virtuais), comunicação entre componentes.

O CloudSim fornece suporte para modelar e simular ambientes em um data center baseado em nuvem virtualizada, tais como: interfaces de gerenciamento dedicado para máquinas virtuais, memória, armazenagem e tamanho de banda. Camadas do CloudSim gerenciam a instanciação e execução de entidades centrais durante o período de simulação. Esta camada é capaz de concorrentemente instanciar e transparentemente gerenciar infraestrutura de nuvem em grande escala consistindo de milhares de componentes de sistemas. As questões fundamentais, tais como: provisionamento da hospedagem para máquina virtual baseado na solicitação do usuário, gerenciamento da execução da aplicação e monitoramento dinâmico são manuseados nesta camada.

Na camada mais alta na pilha de simulação está o código do usuário que expõe configurações relacionadas a funcionalidades do hospedeiro, aplicações, máquinas virtuais, número de usuários e seus tipos de aplicações. Um desenvolvedor de aplicações em nuvem pode gerar um conjunto de solicitações de distribuições de usuários, configurações de aplicações e cenários disponíveis em nuvem nesta camada e realizar testes robustos, baseados nas configurações de nuvem já suportados dentro do CloudSim.

5.2.1.2 Modelando a nuvem

O Serviço de infraestrutura de hardware central relacionado em nuvem é simulado pelo componente *datacenter* para tratar solicitações de serviços. Estas solicitações são elementos de aplicações dentro de VMs. Por processamento VM, entende-se o conjunto de operações relacionadas ao ciclo de vida da VM: Provisionamento de um servidor para uma VM, criação de uma VM, destruição de uma VM e migração de VM.

Um *datacenter* é composto por um conjunto de servidores, que são responsáveis pelo gerenciamento de VMs durante seu ciclo de vida. Servidor é um componente que representa um nodo de computação física dentro da nuvem: A ele é atribuído um processamento pré-configurado na memória, armazenagem e uma política de agendamento para alocação de centros de processamento para máquinas virtuais. O componente servidor implementa interfaces que suportam modelagem e simulação de ambos os nós: simples-core e multi-core.

Alocação de aplicações específicas de VMs para Servidor em um centro de dados baseado em nuvem é de responsabilidade do componente fornecedor de máquina virtual. Este componente expõe um número de métodos customizados para pesquisadores, que auxilia na implementação de novas políticas de provisionamento de VM, baseada nos objetivos de otimização.

Cada componente Servidor instancia um componente escalonador VM que implementa a política de tempo compartilhado ou espaço compartilhado para alocação de núcleos para VM.

5.2.1.3 Modelando a alocação da VM

Para permitir simulação de diferentes políticas sob diferentes níveis de desempenho de isolamento. O CloudSim suporta agendamento de VM em dois níveis: no nível hospedeiro e no nível VM. No primeiro nível, é possível especificar quanto do poder de processamento total de cada núcleo no hospedeiro será designado para cada VM. No próximo nível, as VMs atribuem a quantidade específica do poder de processamento disponível para uma unidade de tarefa individual que está hospedado dentro de sua máquina de execução.

Para cada nível, CloudSim implementa políticas de alocação de recursos de espaço compartilhado e tempo compartilhado. Para melhor ilustrar a diferença entre estas políticas e seus efeitos no desempenho na aplicação, na Figura 5.7 mostra um simples cenário de agendamento. Na figura, duas VMs, e cada uma requer dois núcleos e executa quatro unidades de tarefas: t1, t2, t3 e t4 para ser executado na VM1, enquanto t5, t6, t7 e t8 estão executando na VM2.

Figura 5.7(a) apresenta uma política de espaço compartilhado para ambas VMs e unidades de tarefa: Como cada VM requer dois núcleos, apenas uma VM pode executar em um momento específico de tempo. Portanto, VM2 pode apenas ser atribuído ao centro, uma vez que VM1 finalizou a execução da unidade de tarefa. O mesmo acontece pelas tarefas hospedadas dentro da VM: Como cada unidade de tarefa demanda apenas um núcleo, dois deles executam simultaneamente, e os outros dois são enfileirados até completarem as unidades de tarefas anteriores.

Na Figura 5.7(b), a política de espaço compartilhado é usada para alocação de VMs, mas a política de tempo compartilhado é usada para alocação da unidade de tarefa individual dentro da VM. Então, durante o tempo de vida da VM, todas as tarefas atribuem a ela troca de

contexto dinamicamente até acabarem. Esta política habilita a unidade de tarefa para ser agendada em um tempo anterior, mas, afeta significativamente o tempo para completar a unidade de tarefa que encabeça a fila.

Na Figura 5.7(c), um agendamento de tempo compartilhado é usado para VMs e um espaço compartilhado é usado pelas unidades de tarefa. Neste caso, cada VM recebe um pedaço de tempo de cada centro de processamento e os pedaços são distribuídos para unidades de tarefas na base de tempo compartilhado. Como o núcleo é compartilhado, a quantidade do poder de processamento disponível para a VM é comparativamente menor que os cenários acima mencionados. Como as atribuições das unidades de tarefa são de espaço compartilhado, assim, apenas uma tarefa pode ser alocada para cada núcleo, enquanto as outras tarefas permanecem enfileiradas para uma futura consideração.

Finalmente, na Figura 5.7(d) uma alocação de tempo compartilhado é aplicada para ambas VMs e unidades de tarefas. Assim, o poder de processamento é concorrentemente compartilhado pelas VMs e o compartilhamento de cada VM é concorrentemente dividido entre as unidades de tarefas atribuídas para cada VM. Neste caso, não existem filas entre máquinas virtuais e para as unidades de tarefa.

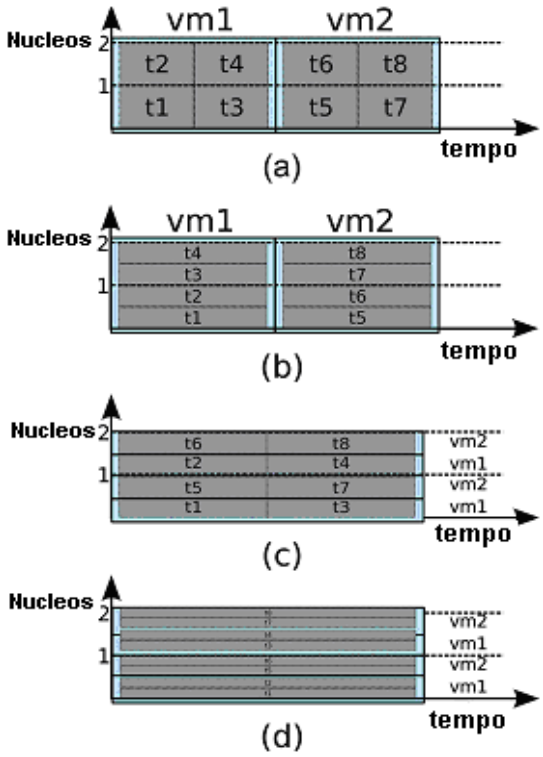

Figura 5.7 Efeitos de diferentes políticas de agendamento na execução de tarefas, fonte [BUY10].

5.2.1.4 Desenho e implementação do CloudSim

O Desenho do diagrama de classe para o simulador é retratado na Figura 5.8

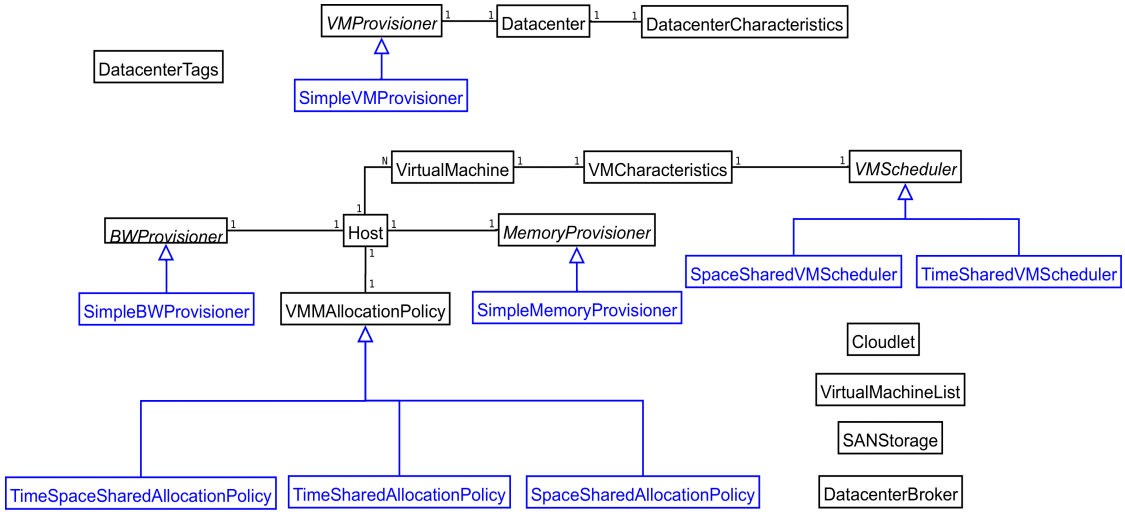

Figura 5.8 Desenho do diagrama de classe do CloudSim, fonte [BUY10]

5.2.2 Experimentos

As VMs no simulador Cloudsim foram criadas e configuradas com 512 MB de memória e duas CPUs. Os servidores (máquinas físicas) foram criados com quatro unidades de processamento, sendo cada um com 200 MIPS (milhões de instruções por segundo). Para a simulação do algoritmo MagBalance, a força inicial de cada VM é de 200 Mips (ou seja, a capacidade de processamento da VM) e vai perdendo sua força conforme o número de *cloudlets* executadas na VM. Cria-se uma VM para cada Servidor, (mesmo número de VMs e de servidores), sendo que cada VM hospeda uma instância do serviço.

Para os experimentos com o algoritmo MagBalance, considerou-se uma arquitetura de grafo completo e grafo não completo, onde cada VM pode ser um ponto de entrada das *cloudlets*.

Nos experimentos foram utilizados intervalos no envio das *cloudlets* entre 10 e 20 milésimos de segundos aleatórios uniformemente distribuídos e o tamanho de uma cloudlet é medido em unidades de processamento (MIPS), com tamanho definido conforme o experimento.

Na simulação supõe-se que, não ocorrem quaisquer tipos de falhas, isto é, os servidores funcionam corretamente de forma ininterrupta e todas as mensagens enviadas são

recebidas corretamente dentro de um tempo aceitável. Além disso, supõe-se que não ocorram quaisquer eventos que possam comprometer a segurança do sistema, bem como, por padrão dos algoritmos Round Robin e Least Connections, somente foi realizado experimento comparativo com grafo completo.

A Tabela 5.1 apresenta os diferentes cenários para as simulações. Os cenários I, II e III foram usados para comparar os algoritmos Round Robin, Least Connections e MagBalance com relação ao tempo de execução de um conjunto de *cloudlets*. O cenário IV foi usado para avaliar o algoritmo de balanceamento de carga MagBalance com relação ao seu custo, medido em função do número de mensagens trocadas em dois momentos:

- Estabelecimento da Rede Magnética Virtual. (estabelecido através da leitura de arquivos de texto, com grafos em que cada nó influencia outros 2, 3 e 4 nós vizinhos);
- Roteamento das *cloudlets* e correspondente atualização das forças de atração.

Além disso, o cenário IV foi usado para avaliar o desempenho do algoritmo MagBalance, com diferentes quantidades de VMs e diferentes quantidades de arestas por VM.

O cenário V foi usado para comparar os algoritmos Round Robin e MagBalance com relação ao tempo de execução de um conjunto de *cloudlets*, onde cada servidor hospeda um conjunto diferente de instâncias de serviço, conforme Figura 5.9. O diferencial deste experimento consiste em que o algoritmo MagBalance considera a carga do servidor para realizar a distribuição das cloudlets, em vez da carga da VM.

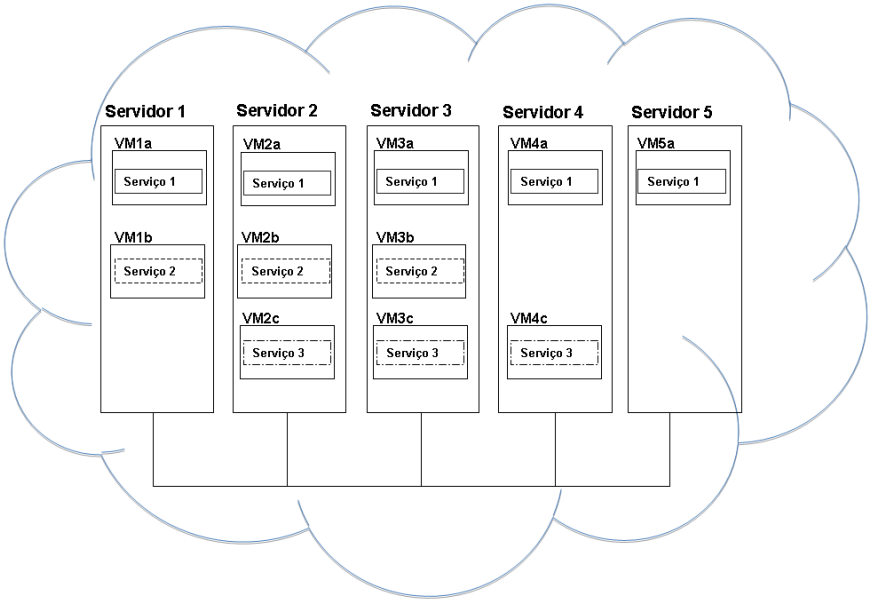

Figura 5.9 Distribuição das instâncias dos serviços para experimento cenário V.

Tabela 5.1 Cenários para as simulações.

Cenário	**Número Repetições**	**Tipo de grafo**	***Cloudlets***	
			Tamanho	**Quantidade**
I	10	Completo 10-100 VMs	60.000 – 80.000	50
II	03	Completo 20-100 VMs	300.000 – 400.000	1000
III	10	Completo 20 VMs	40.000	10-100
IV	03	Não Completo 10-100 VMs, com duas, três e quatro arestas	80.000 – 100.000	900
V	05	Completo	20.000 – 40.000	1.000-5.000

Cenário I

Objetivo: Comparação de desempenho entre os três algoritmos (Round Robin, Least Connections e MagBalance), variando o tamanho da rede com baixa carga.

***Cloudlets*:**

- Quantidade: 50;
- Tamanho: Aleatório com distribuição uniforme entre 60.000 e 80.000 Mips.

Rede: Grafo completo, com número de VMs variável de 10 a 100.

Execução: 10 vezes para cada rede (executa-se 10 vezes porque o envio das cloudlets para a VM é aleatório, isto é, o tempo total de execução pode ser diferente, então, busca-se o intervalo de confiança).

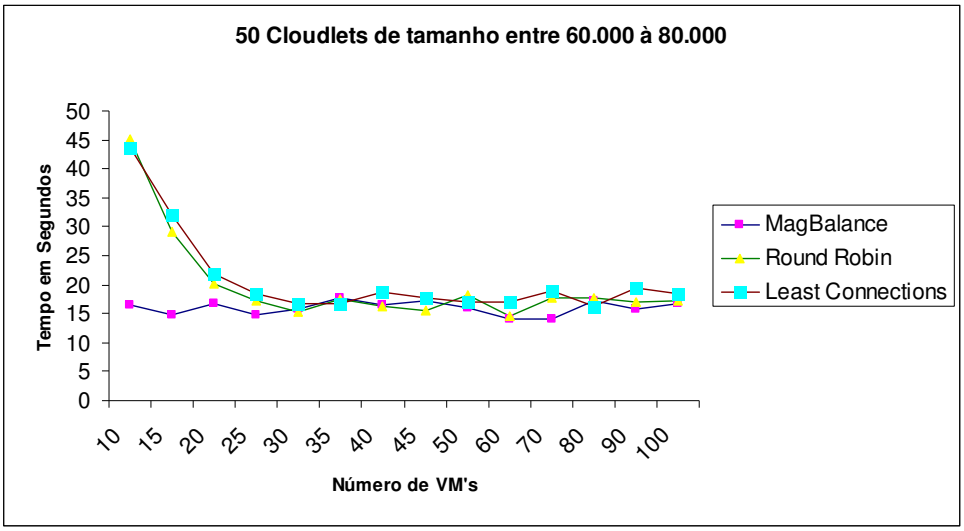

Figura 5.10 Comparativo de desempenho no Cenário I.

Como pode ser observado na Figura 5.10, o algoritmo MagBalance tem um ganho de desempenho em relação aos algoritmos Round Robin e Least Connections até 30 VMs, a partir deste ponto os três algoritmos praticamente se igualam. Isto ocorre porque: o número de *cloudlets* e o número de VMs é praticamente o mesmo, portanto sendo executado uma *cloudlet* em cada VM.

Cenário II

Objetivo: Comparação de desempenho entre os três algoritmos (Round Robin, Least Connections e MagBalance), variando o tamanho da rede, com alta carga.

***Cloudlets*:**

- Quantidade: 1000;
- Tamanho: Aleatório com distribuição uniforme entre 300.000 e 400.000 Mips.

Rede: Grafo completo, com número de VMs variável de 20 a 100.

Execução: Três vezes para cada rede

Com o Cenário II, foi comparado o tempo de execução entre os algoritmos MagBalance, Round Robin e Least Connections com as *cloudlets* consumindo um tempo maior de processamento em relação ao Cenário I.

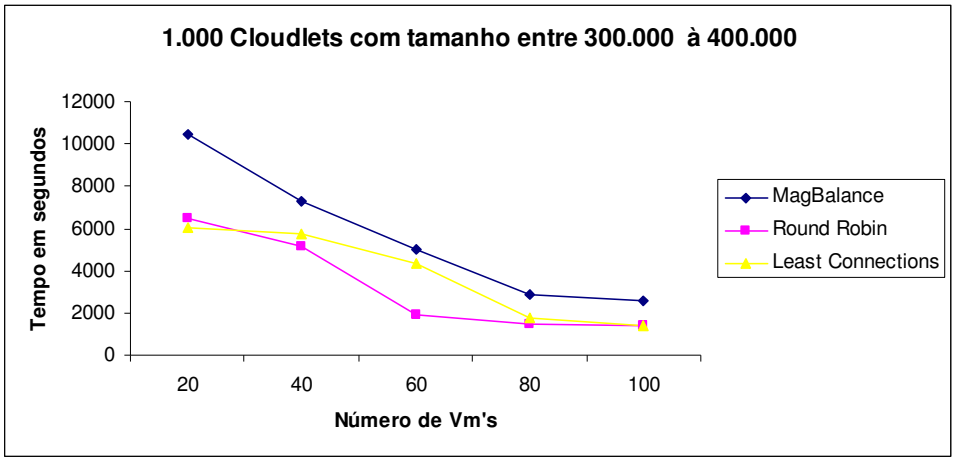

Figura 5.11 Comparativo de desempenho no Cenário II

Como pode ser observado na Figura 5.11, os algoritmos de balanceamento de carga Round Robin e Least Connections têm um melhor desempenho que o algoritmo MagBalance quando submetido para execução *cloudlets* que consomem muita CPU. Isso acontece porque o algoritmo MagBalance tem um alto custo com a troca de mensagens. Como vantagem, o algoritmo MagBalance oferece o recurso de ser totalmente distribuído.

Cenário III

Objetivo: Comparação de desempenho entre os três algoritmos (Round Robin, Least Connections e MagBalance), variando o número de *cloudlets*.

***Cloudlets*:**

- Quantidade: 10 a 100 de (5 em 5 até 50) e (10 em 10 de 50 até 100);
- Tamanho: Fixo de 40.000 Mips.

Rede: Grafo completo, com número de VMs fixo em 20.

Execução: 10 vezes para cada rede

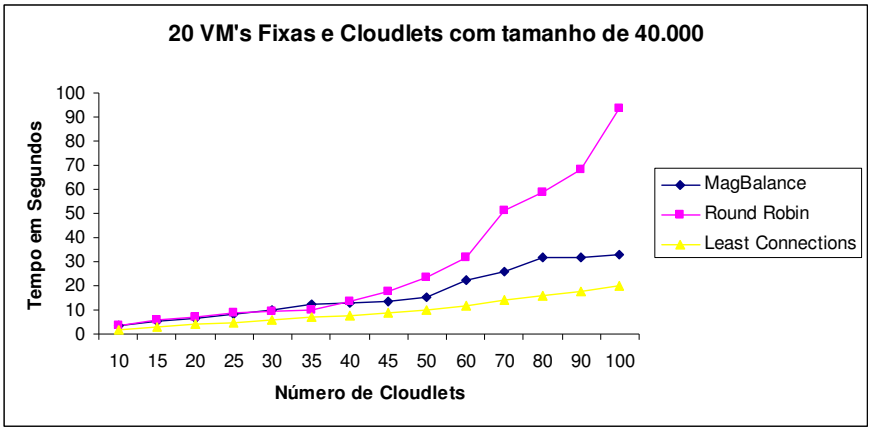

Figura 5.12 Comparativo de desempenho no Cenário III

Como pode ser observado na Figura 5.12, os algoritmos MagBalance, Round Robin e Least Connections têm performances muito parecidas até 40 *cloudlets*. A partir deste ponto, o algoritmo MagBalance tem melhor desempenho em relação ao algoritmo Round Robin e o algoritmo Least Connections tem um desempenho melhor que os outros dois algoritmos. Como pode ser observado no na Figura 5.12 quanto maior o número de *cloudlets* melhor é o desempenho dos algoritmos MagBalance e Least Connections em relação ao algoritmo Round Robin; bem como o desempenho dos algoritmos MagBalance e Least Connections se mantém estável, não ocorrendo grande variação de desempenho.

Embora o algoritmo MagBalance apresente um desempenho inferior ao desempenho do algoritmo Least Connections, tem como vantagem ser totalmente distribuído.

Cenário IV.a

Objetivo: Avaliar o algoritmo de balanceamento de carga MagBalance com relação ao seu custo para estabelecer a rede overlay, medido em função do número de mensagens trocadas.

Rede: Grafo não completo, com número de VMs variável de 10 a 100 com duas, três e quatro arestas por nó.

Execução: Três vezes para cada rede.

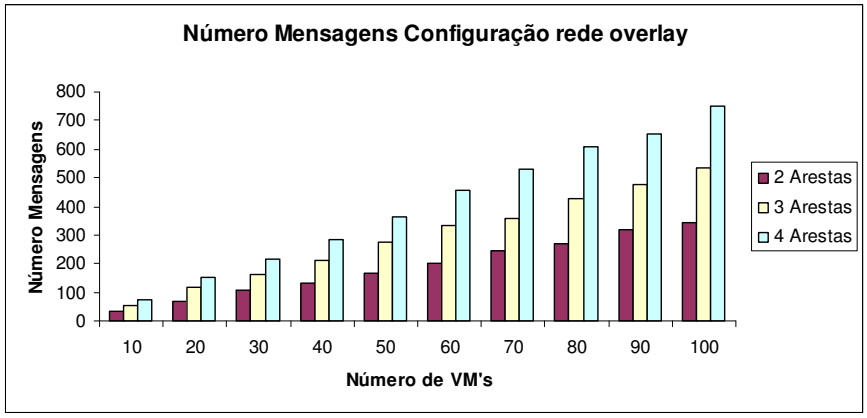

Figura 5.13 Comparativo de desempenho no Cenário IV.a

Como pode ser observado na Figura 5.13, o número de mensagens trocadas entre os nós para configuração de uma rede overlay utilizando do algoritmo MagBalance, conclui-se que, quanto maior o número de arestas que representa ajuda a um determinado nó, maior será o número de mensagens trocadas para estabilizar a rede overlay, mas o número de mensagens enviadas através de cada aresta é aproximadamente constante (duas mensagens para grafo com duas arestas, cinco mensagens para grafo com três arestas, sete mensagens para grafo com quatro arestas).

Cenário IV.b

Objetivo: Avaliar o algoritmo de balanceamento de carga MagBalance com relação ao seu custo no roteamento das *cloudlets* e correspondente atualização das forças de atração.

***Cloudlets*:**

- Quantidade: 900.
- Tamanho: Aleatório, com distribuição uniforme entre 80.000 e 90.000 Mips.

Rede: Grafo não completo, com número de VMs variável de 10 a 100 com duas, três e quatro arestas por nó.

Execução: Três vezes para cada rede.

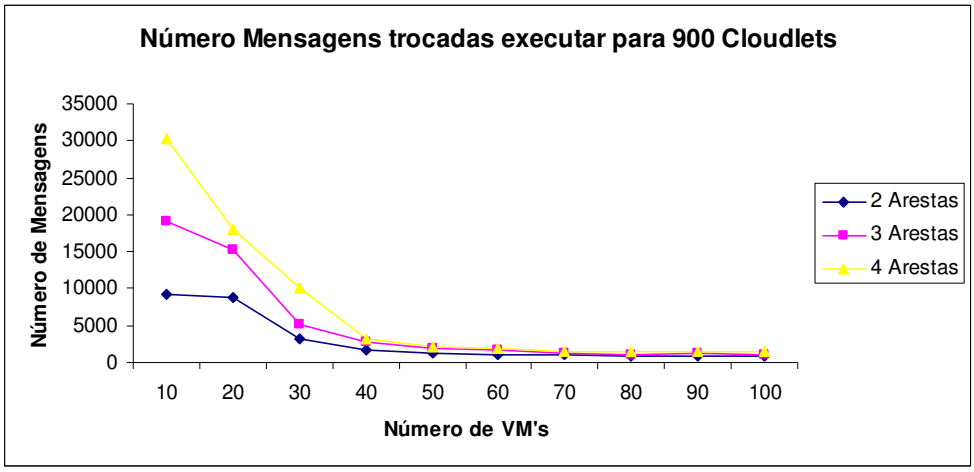

Figura 5.14 Comparativo de desempenho no Cenário IV.b

Como pode ser observado na Figura 5.14, conclui-se que, para executar 900 *cloudlets* o algoritmo se estabiliza em 40 VM's, independente do número de VMs que ajudam diretamente a VM que recebeu a *cloudlet*.

Se 900 cloudlets totalizaram 9000 mensagens, a média foi de, por volta de 10 mensagens disparadas a cada cloudlet executado. Assim sendo, vê-se que são muitas cloudlets para apenas 10 nós processarem (90 cloudlets por nó), o que faz o pivô da rede toda trocar frequentemente e gerar broadcast de mensagens, a fim de atualizar as tabelas do algoritmo. Por outro lado, o número de mensagens cai drasticamente com 30 VMs porque o número de cloudlets que cada nó executa (em média) vai ser de 30 e, além disso, a rede é 200% maior, fazendo com que os pivôs não mudem com tanta frequência, portanto, não havendo necessidade de encaminhar muitas mensagens.

Cenário IV.c

Objetivo: Avaliar o algoritmo de balanceamento de carga MagBalance com relação ao seu desempenho com diferentes quantidades de VMs e diferentes quantidades de arestas por VM.

***Cloudlets*:**

- Quantidade: 900.
- Tamanho: Aleatório com distribuição uniforme entre 80.000 e 90.000 Mips.

Rede: Grafo não completo, com número de VMs variável de 10 a 100 com duas, três e quatro arestas por nó.

Execução: Três vezes para cada rede.

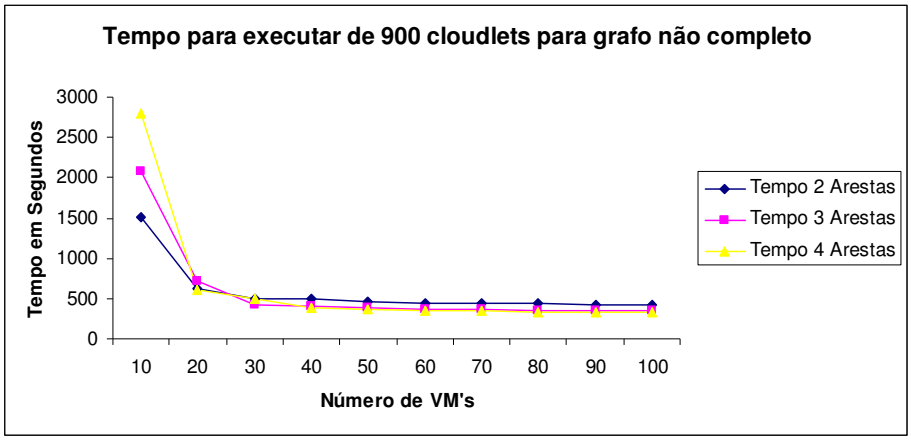

Figura 5.15 Comparativo de desempenho no Cenário IV.c

Como pode ser observado na Figura 5.15, conclui-se que:

- O grafo com três ou quatro arestas, a partir de 30 VM's não ocorre oscilação de tempo, ficando estabilizado. Desta forma, com qualquer número de VM's entre 30 e 100, atende de forma satisfatória 900 *cloudlets*, ou seja, se for utilizado mais do que 30 VMs para atender 900 cloudlets, significa desperdício de recursos.
- Já com grafo com duas arestas, apresenta um bom desempenho até 20 VM's em relação ao grafo de três ou quatro arestas. A partir deste ponto, apresenta um desempenho inferior.

Cenário V

Objetivo: Comparação de desempenho entre os algoritmos (Round Robin e MagBalance) com carga aleatória dos servidores e da distribuição das instâncias dos serviços, baseado na Figura 5.9.

***Cloudlets*:**

- Quantidade: 1.000 a 5.000 de 1000 em 1000.
- Tamanho: Aleatório com distribuição uniforme entre 20.000 e 40.000 Mips.

Rede: Grafo completo.

Execução: Cinco vezes para cada rede (executa-se cinco vezes porque o envio das cloudlets para a VM é aleatório, isto é, o tempo total de execução pode ser diferente, então, busca-se o intervalo de confiança).

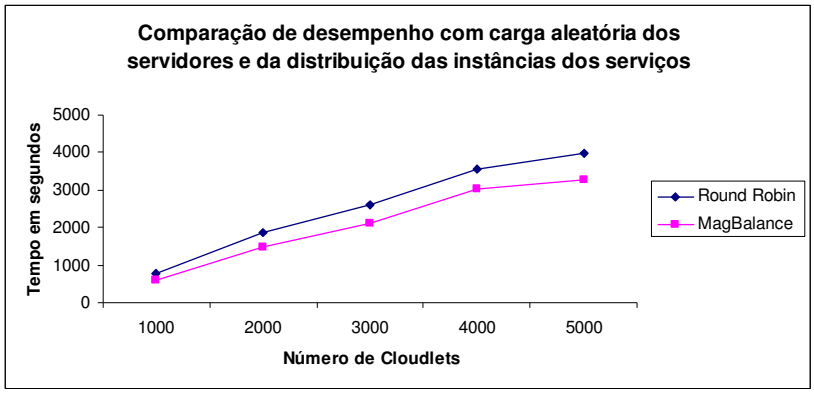

Figura 5.16 Comparativo de desempenho no Cenário V

Como pode ser observado na Figura 5.16, conclui-se que, independente do número de *cloudlets* o algoritmo MagBalance tem melhor desempenho em relação ao algoritmo Round Robin, este ganho de performance ocorre porque o algoritmo MagBalance considera a carga do servidor para a atração das *Cloudlets*. Também como pode ser observado na Figura 5.16 quanto maior o número de Cloudlets sendo executado melhor o desempenho do algoritmo MagBalance.

5.3 Conclusão

Baseado nos experimentos realizados, podemos afirmar que o algoritmo MagBalance pode ser aplicado em balanceamento de carga em computação em nuvem, obtendo um ganho, em determinados casos, em relação ao balanceamento de carga Round Robin, que é um dos algoritmos mais utilizado em softwares livres para balanceamento de carga e em relação ao algoritmo Least Connections, é o mais rápido entre todos os algoritmos de balanceamento de carga implementados no ativo NetScaler da Citrix[CITRIX11].

Mesmo nos casos em que o algoritmo MagBalance não oferece o melhor desempenho de balanceamento de carga de trabalho, pode-se utilizá-lo por ter como vantagem o algoritmo ser totalmente distribuído.

Capítulo 6

6 Conclusão

Este trabalho apresentou uma pesquisa do funcionamento atual dos algoritmos de balanceamento de carga, aplicados em computação em nuvem. Todos os algoritmos verificados possuem um único ponto de entrada, podendo se tornar um gargalo para o desempenho do sistema e o único ponto de falha. Da mesma forma, o presente trabalho de dissertação apresentou como solução o algoritmo MagBalance, aplicado no contexto de computação em nuvem, sendo este totalmente distribuído.

Como análise comparativa realizada através de experimentos, optou-se em comparar o algoritmo MagBalance com os algoritmos Round Robin e Least Connections porque o primeiro é um dos mais utilizados entre todos os algoritmos de código aberto e o segundo algoritmo é o mais eficiente entre todos os algoritmos de balanceamento de carga implementados no ativo NetScaler da Citrix[CITRIX11]. Para realizar os experimentos foram propostos vários cenários, comparando desempenho entre os algoritmos e verificando o custo do algoritmo MagBalance. Observou-se através dos gráficos que o algoritmo MagBalance apresenta desempenho um pouco inferior em relação aos demais algoritmos, principalmente, devido ao maior custo com a troca de mensagens. Entretanto, os benefícios de ser totalmente distribuído, incluindo escalabilidade e tolerância a falhas, podem justificar seu uso em computação em nuvem.

Este trabalho de pesquisa ainda pode ser melhorado com a realização dos seguintes trabalhos futuros:

1. Verificar o comportamento do algoritmo quanto a seu funcionamento correto na presença de falhas;
2. Verificar o comportamento do algoritmo quanto ao seu desempenho, quando ocorre entrada ou saída de nós da rede;
3. Introduzir no algoritmo os necessários mecanismos de segurança, tal como a confiança no nó.

4. Desenvolver um *middleware* baseado no algoritmo MagBalance, aplicado para o balanceamento de carga na computação em nuvem.
5. Desenvolver experimentos para fins de comparação do modelo proposto com outros algoritmos de balanceamento de carga, principalmente com o algoritmo Least Response Time.
6. Desenvolver mais experimentos de validação do modelo proposto, com grandes quantidades de instâncias de serviços.

Referências Bibliográficas

[CAL10] CALSAVARA, ALCIDES; LIMA, L.A.P. Autonomic Application-Level Message Delivery UsingVirtual Magnetic Fields. Journal of Network and Systems Management, Springer, v. 18, n. 1, p. 97-116, 2010.

[LIMA10] LIMA JR., L.; CALSAVARA, A., Routing Based on Message Attraction. In: IEEE.2010 IEEE 24th International Conference on Advanced Information Networking and Applications Workshops. [S.l.], 2010. p. 189-194.

[KEN11] KEN SALCHOW, JR, Balanceamento de carga: a Evolução para os application *Delivery Controller*. http://www.f5networks.com.br/pdf/white-papers/balanceamento-de-carga-a-evolucao-para-os-application-delivery-controller-wp.pdf. acessado em março 2011.

[VEC09] VECCHIOLA, C., CHU, X., BUYYA, R. *Aneka: A Software Platform for .NET-based Cloud Computing*. Australia. 2009.

[ALE08] ALECRIM, EMERSON, http://infowester.com/printversion/cloudcomputing.php. Acessado em novembro 2009.

[TAN07] TANENBAUM, ANDREW S., *Sistemas distribuídos*, ed. São Paulo,Hall, 2007.

[JER08] JEREMY GEELAN. *Twenty one experts define cloud computing. Virtualization*, august 2008. Eletronic Magazine, artigo disponível em http://virtualization.sys-con.com/node/612375.

[RAJ08] RAJKUMAR BUYYA, CHEE SHIN YEO, E SRIKUMAR VENUGOPAL. *Market-oriented cloud computing: Vision, hype e reality for delivering it services as computing utilities*, CoRR, (Abs/0808.3558), 2008.

[VAQ09] LUIS M. VAQUERO, LUIS RODERO-MERINO, JUAN CACERES, MAIK LINDNER. *A Break in the clouds: Towards a Cloud Definition*. Submetido CCR, 2009.

[ARM09] ARMBRUST, M., FOX, A., GRIFFITH, R.,JOSEPH, A. D., KATZ, R. H., KONWINSKI, A., LEE, G., PATTERSON, D. A., RABKIN, A., STOICA, I., AND ZAHARIA, M. *Above the clouds: A berkeley view of cloud computing. Technical report*, EECS Departament, University of California, Berkeley, 2009.

[SAL09] SALESFORCE, *"Salesforce"*. http://www.salesforce.com/, 2009

[CIU09] CIURANA, E. *Developing with Google App Engine*. Appress, Berkely, CA, USA, 2009

[LIU07] LIU, S., LIANG, Y., AND BROOKS, M. *Eucalyptus: a web service-enabled e infraestrutura*. In CASCON '07: Proceedings of the 2007 conference of the center for advanced studies on Collaborative research, pages 1-11, New York, NY, USA. ACM, 2007

[NIST09] NIST. "*National institute of standards and technology draft definition of cloud computing*". http://csrc.nist.gov/groups/SNS/cloud-computing,2009

[MAR09] MARINOS, A. AND BRISCOE, G. *Community cloud computing. CoRR, abs/0907.2485, 2009*

[BUY09] RODRIGO N. CALHEIROS, RAJIV RANJAN, CÉSAR A. F. DE ROSE, RAJKUMAR BUYYA, *CloudSim: A Novel Framework for Modeling ans Simulation of Cloud Computing Infrastructures and Services*.

[NIST10] *NIST Definition of Cloud Computing v15*, csrc.nist.gov/groups/SNS/cloud-computing/cloud-def-v15.doc.

[PAR66] PARKHILL F. DOUGLAS, *The challenge of the computer utility. Addison-Wesley Publishing Company*, 1966

[ARM10] ARMBRUST MICHAEL, FOX ARMANDO, GRIFFITH REAN, JOSEPH D. ANTHONY, KATZ RANDY, KONWINSKI ANDY, LEE GUNHO, PATTERSON DAVID, RABKIN ARIEL, STOICA ION, ZAHARIA MATEI, *A View of Cloud Computing*. Communication of The ACM, 2010

[ZHONG02] ZHONG XU, RONG HUANG, *Performance Study of Load Balancing Algorithms in Distributed Web Server Systems*, CS213 Parallel and Distributed Processing Project Report, 2002.

[MCE84] P. L. MCENTIRE, J. G. O'REILLY, AND R. E. LARSON, *Distributed Computing: Concepts and Implementations.* New York: IEEE Press, 1984.

[SAN08] SANDEEP SHARMA, SARABJIT SINGH, AND MEENAKSHI SHARMA, *Performance Analysis of Load Balancing Algorithms.* Word Academy of Science, Engineering and Technology, 2008.

[MAL00] S. MALIK, *Dynamic Load Balancing in a Network of Workstation*, 95.515 Research Report, 19 november 2000.

[WIL00] WILLIAM LEINBERGER, GEORGE KARYPIS, VIPIN KUMAR, *Load Balancing Across Near-Homogeneous Multi-Resource Servers,* IEEE, 2000.

[NAK04] S. NAKRANI AND C. TOVEY, *On Honey Bees and Dynamic Server Allocation in Internet Hosting Centers*. Adaptive Behavior 12, pp: 223-224, 2004

[ABU08] O. ABU-RAHMEH, P. JOHNSON AND A. TALEB-BENDIAB, *A Dynamic Biased Random Sampling Scheme for Scalable and Reliable Grid Networks*, INFOCOMP - Journal of Computer Science, ISSN 1807-4545, 2008.

[MELL09] MELL, P. E GRANCE, T. *The NIST definition of cloud computing*. National Institute of Standards and Technology, 2009

[PIN05] PINHEIRO JR, J. E KON, F. *Segurança em grades computacionais*. Em Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais, 2005

[FOS08] FOSTER, I., ZHAO, Y., RAICU, I., E LU, S. *Cloud computing and grid computing 360-degree compared.* Em Grid Computing Environments Workshop, 2008

[CITRIX11] http://community.citrix.com/display/cdn/Load+Balancing+Methods, acessado em março 2011.

[GRE09] Greenberg, A. Networking The Clound. ICDCS 2009 keynote.http://www.google.com/corporate/green/datacenters/measuring.html, Março.

[KLEMS09] Markus Klems, Jens Nimis e Stefan Tai, Do cloud computing? A framework for estimating the value of cloud computing, Forschungszentrum Informatik , Alemanha, 2009.

[ALEX09] Alexandros Marinos, Gerard Briscoe, Community Cloud Computing, First International Conference on Cloud Computing, Beijing, China, December 2009.