PROFUSE: UMA ABORDAGEM DE GARANTIA DE QUALIDADE DE SERVIÇO PARA SISTEMAS HOSPEDADOS EM NUVENS IAAS

Luís Fernando Orleans

Tese de Doutorado apresentada ao Programa de Pós-graduação em Engenharia de Sistemas e Computação, COPPE, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Doutor em Engenharia de Sistemas e Computação.

Orientador: Geraldo Zimbrão da Silva

Rio de Janeiro
Setembro de 2013

PROFUSE: UMA ABORDAGEM DE GARANTIA DE QUALIDADE DE SERVIÇO PARA SISTEMAS HOSPEDADOS EM NUVENS IAAS

Luís Fernando Orleans

TESE SUBMETIDA AO CORPO DOCENTE DO INSTITUTO ALBERTO LUIZ COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE ENGENHARIA (COPPE) DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE DOUTOR EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÂO.

Examinada por:

_____
Prof. Geraldo Zimbrão da Silva, D.Sc.

_____
Profª. Marta Lima de Queirós Mattoso, D.Sc.

_____
Prof. Geraldo Bonorino Xexéo, D.Sc.

_____
Profª. Jonice de Oliveira Sampaio, D.Sc.

_____
Profª. Fernanda Araújo Baião Amorim, D.Sc.

_____
Prof. Sérgio Manuel Serra da Cruz, D.Sc.

RIO DE JANEIRO, RJ - BRASIL
SETEMBRO DE 2013

Orleans, Luís Fernando

PROFUSE: Uma Abordagem de Garantia de Qualidade de Serviço para Sistemas Hospedados em Nuvens IaaS / Luís Fernando Orleans. – Rio de Janeiro: UFRJ/COPPE, 2013.

XIV, 99 p.: il.; 29,7 cm.

Orientador: Geraldo Zimbrão da Silva

Tese (doutorado) – UFRJ/ COPPE/ Programa de Engenharia de Sistemas e Computação, 2013.

Referências Bibliográficas: p. 84 - 89.

1. Elasticidade. 2. Computação em Nuvem. 3. Teoria de Controle. I. Silva, Geraldo Zimbrão da. II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia de Sistemas e Computação. III. Título.

Para meu pai Luís Augusto (In memoriam),
minha avó Dalva (In memoriam),
para minha família,
para meus amigos,
para Helen, meu outro mundo.

AGRADECIMENTOS

Agradeço primeiramente a Deus, sem o qual nada seria possível.

Sou grato à minha esposa Helen, que em todos os momentos soube me apoiar, com palavras carinhosas, gestos gentis ou mesmo com sua simples companhia.

Igualmente agradeço à minha família: minha mãe Susan, meu irmão Luís Felipe e sua esposa Renata, meus primos Ricardo, Carolina e Vitória, minha avó Helena, meus tios Luiz Paulo, Susie, Solange, Lázaro, Luciana, Sérgio, Denise, Adriano, Vera e Luiz Cláudio e meu padrasto Adolpho por serem, em todos os momentos, meu porto seguro. Agradeço ao meu sobrinho Pedro que, apesar da pouca idade, enche a família de paixão e ternura.

Agradeço aos meus sogros Pedro Paulo e Marilda e à minha cunhada Fabíola, cuja família agora pertenço.

Obrigado aos meus amigos Fábio, Elaine, André, Dione, Wanderson, Gilmara, Guilherme, Carla, Sérvulo, Danielle, Carlos, Muniky, Vinícius, Monique e Henrique, e aos pequenos Enzo, Ayron, Luana e Lívia, que se mostraram presentes em todos os momentos de necessidade e entenderam a minha ausência em várias ocasiões.

Gostaria de agradecer, ainda, a todos os amigos que fiz ao longo deste trabalho: Carlos Eduardo Mello, Filipe Braida, Fellipe Duarte, Marden Pasinato, Pedro Rougemont, José Rodrigues Neto, Bruno Osiek, Ricardo Barros, Cláudia Susie, Rogério Borba e Luiz Manoel.

Agradeço ao professor e amigo Geraldo Xexéo, sempre solícito, ao professor Jano Moreira de Souza por compartilhar seu conhecimento e sua experiência, às professoras Marta Mattoso e Vanessa Braganholo Murta pelas maravilhosas dicas em meu Exame de Qualificação e que acabaram por nortear o restante deste trabalho e à professora Fernanda Baião, por aceitar fazer parte desta banca.

Sou grato à professora Jonice Oliveira, por toda a ajuda dispendida no início do curso, e ao professor Sérgio Serra, por toda a ajuda dispendida no fim do curso. A ambos, também agradeço por terem aceitado participar da banca de defesa desta tese.

Não menos importante, gostaria de manifestar minha gratidão à Josefina Solange, Cláudia Prata, Maria Mercedes, Roberto Rodrigues, Gutierrez, Sônia Regina, Patrícia Leal, Ana Paula e todos os demais funcionários do PESC, pela ajuda com questões administrativas.

Agradeço aos professores Benaia, Adria Lyra e Leandro Alvim, da UFRRJ, pela compreensão e ajuda.

Por fim, agradeço ao meu orientador Geraldo Zimbrão, pelas conversas, ajudas e conselhos. Foram incontáveis almoços, reuniões e mensagens trocadas, tendo a chance de aprender com uma das pessoas mais inteligentes e humanas que tive o prazer de conviver.

Obrigado.

Resumo da Tese apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Doutor em Ciências (D.Sc.)

PROFUSE: UMA ABORDAGEM DE GARANTIA DE QUALIDADE DE SERVIÇO PARA SISTEMAS HOSPEDADOS EM NUVENS IAAS

Luís Fernando Orleans

Setembro/2013

Orientador: Geraldo Zimbrão da Silva

Programa: Engenharia de Sistemas e Computação

Para sistemas hospedados em nuvens IaaS que visem ao lucro, como *blogs* e sistemas de comércio eletrônico, a métrica mais importante a ser observada deve ser o lucro final. Balancear a QoS para evitar o abandono de clientes com o custo resultante do aluguel de instâncias formam a base do gerenciamento de instâncias para esses cenários. Nesta tese nós demonstramos a viabilidade de utilização de uma Máquina de Inferência Nebulosa como mecanismo capaz de manter a QoS experimentada pelos clientes. Foi criada, ainda, uma técnica que reduz o custo com o aluguel de servidores ao aumentar a capacidade computacional total de modo incremental, uma unidade por vez. Para evitar as consequências de rajadas imprevisíveis de requisições, denotadas por ruídos, utilizamos monitores chamados de ganchos. Todas essas técnicas formam o PROFUSE, um Modelo de Elasticidade completo. Nossos experimentos avaliaram o PROFUSE sob diversas cargas de trabalho e mostraram que ele é capaz de manter o tempo de processamento das requisições abaixo de um limite pré-estabelecido, impedindo, dessa forma, a insatisfação dos clientes. Ao mesmo tempo, o PROFUSE reduz o custo total com alocação de instâncias, mesmo que o custo-benefício entre diferentes configurações não seja linear.

Abstract of Thesis presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Doctor of Science (D.Sc.)

PROFUSE: A QUALITY OF SERVICE LAYER FOR IAAS CLOUD HOSTED SYSTEMS

Luís Fernando Orleans

September/2013

Advisor: Geraldo Zimbrão da Silva

Department: Systems and Computer Engineering

For systems hosted in IaaS clouds that target profit, like blogs and e-commerce systems, the final revenue should be the most important metric. Balancing the QoS experienced by clients as a manner to avoid their drop-out against the cost related to leasing instances form the basis of a good instance management for those scenarios. In this thesis, we demonstrate the feasibility of using a Fuzzy Logic Inference System as a tool for maintaining the QoS. Furthermore, it was created a method for reducing the overall cost related to instances leasing using an incremental algorithm, acquiring one computational unit at a time. Finally, we used hooks to handle unpredictable burst of requests, called here as noises. Those techniques together form the PROFUSE Elasticity Model. Our experiments evidenced that PROFUSE can keep the response time of all requests below a pre-stablished threshold, denoted here as deadlines, avoiding customer unsatisfaction. At the same time, PROFUSE diminishes the total cost of servers lease, even when the cost-benefit among different configurations is not linear.

Sumário

Lista de figuras

Lista de tabelas

Lista de Siglas

ANS – Acordo de Nível de Serviço

CdT – Carga de Trabalho

CN – Computação em Nuvem

DCdT – Diferença de Carga de Trabalho

FCFS – Primeiro a Chegar, Primeiro a ser Servidor

FIFO – Primeiro a Chegar, Primeiro a Sair

IaaS – Infraestrutura como Serviço

MIN – Máquina de Inferência Nebulosa

MPL – Nível de Multiprogramação

NN – Número de Núcleos

PaaS – Plataforma como Serviço

PPD – Percentual de Perda de *Deadlines*

QoS – Qualidade de Serviço

RN – Região Nebulosa

SaaS – Software como Serviço

SAM – Módulo de Aquisição de Servidores

SED – Simulador de Eventos Discretos

TCC – Taxa de Conversão de Clientes

TCR – Teoria de Controle com Retroalimentação

TFR – Tamanho da Fila de Requisições

TPPC – Tempo Para Próxima Categoria

UF – Utilização Fixa

US – Utilização do Sistema

VL – Variável Linguística

VM – Máquina Virtual

WOM – Módulo Observador da Carga de Trabalho

Capítulo 1 - Introdução

A popularização da internet levou a um aumento expressivo do número de sistemas de comércio eletrônico, ou seja, sistemas que obtêm suas receitas *online* (Zhang; Cheng; Boutaba, 2010), contribuindo para uma enorme variedade de novos modelos de vendas: sistemas de compras coletivas, leilões de 1 centavo, promoções relâmpago, anúncios, entre outros, inovando a maneira como se realizam negócios na internet.

Um dos problemas enfrentados pelas empresas responsáveis por estes novos modelos de negócio, comumente chamadas de *startups* (“Startup company”, 2013), está em planejar como os servidores (ou instâncias, termos usados sem distinção neste trabalho) que hospedam seus sistemas reagirão em situações de pico, uma vez que a quantidade de potenciais clientes, isto é, a *carga de trabalho*, é desconhecida. Um dos efeitos indesejados desse fenômeno acontece quando o número de pessoas que acessam o sistema é subdimensionado, levando a uma condição de saturação do sistema com consequente perda de desempenho (Orleans, L. F.; Furtado, 2007), (Schroeder *et al.*, 2006), (Armbrust *et al.*, 2010). Esta questão é particularmente importante, pois os usuários de sistemas de comércio eletrônico são extremamente impacientes, onde (Armbrust *et al.*, 2010) e (“Selfish, mean, impatient customers: New Thinking: Gerry McGovern”, [s.d.]) apontam que, caso o sistema demore tempo demais para processar suas requisições, é provável que o usuário o abandone e procure um concorrente.

O efeito contrário ao descrito anteriormente consiste em superdimensionar a quantidade de clientes, comprando mais servidores do que é realmente necessário (Armbrust *et al.*, 2010), onde a maioria destes permanecem desligados ou subutilizados.

Em comum nos dois cenários descritos acima está o fato de que os servidores pertencem à empresa que mantém o sistema de comércio eletrônico, sendo ela a responsável não só pelo planejamento prévio de suas aquisições, como pela manutenção, configuração, tarefas administrativas (*backup*, redundância, etc.) e ajustes finos de desempenho (*fine-tunning*). Em muitos casos, as *startups* são empresas jovens, com orçamentos limitados, sendo custoso para elas não somente adquirir esses servidores e toda a infraestrutura necessária, como manter uma equipe inteira responsável pela sua manutenção.

Dessa forma, muitas destas empresas vêm optando por hospedar seus sistemas em

provedores de nuvem, utilizando as vantagens proporcionadas pela Computação em Nuvem (CN), dentre as quais podemos destacar:

- Processamento “elástico”: é possível aumentar o número de instâncias alocadas sem interrupção do serviço, garantindo, assim, a satisfação dos usuários;
- Armazenamento “elástico”: o espaço para armazenamento de dados pode variar de GB a PB em questão de minutos, sem que haja indisponibilidade de serviços;
- Alta disponibilidade: redundância garante a disponibilidade dos recursos sempre que são necessários;
- Migração de tarefas de infraestrutura para o provedor de serviços em nuvem, como realização de cópias de segurança, compactação de *log*, etc.

Provedores de Computação em Nuvem possuem centrais de processamento (*datacenters*) com centenas ou, às vezes, milhares de servidores com diferentes configurações interligados, permitindo que um cliente, no caso a empresa de comércio eletrônico, possa hospedar seu sistema em quantas instâncias desejar. Esses provedores disponibilizam interfaces que podem ser utilizadas para alugar (liberar) servidores, sendo necessários poucos segundos para aumentar (diminuir) a capacidade computacional total disponível, em um modelo que é conhecido como *pague-pelo-uso* (*pay-per-use*, em inglês) (Zhang; Cheng; Boutaba, 2010), (Armbrust *et al.*, 2010), (Oliveira, 2012), (Ogasawara, 2012) e (Mao; Humphrey, 2012). Assim, a empresa que hospedar seu sistema em um provedor de nuvem somente arcará os custos de aluguel de servidores por tempo, medidos em *instância-hora*.

Dessa forma, cabe à empresa de comércio eletrônico monitorar a utilização de seu sistema e, quando necessário, solicitar ao provedor de nuvem novas instâncias, como forma de manter a qualidade do serviço (QoS) necessária para que seus usuários não abandonem o sistema e procurem um concorrente. Contudo, tal tarefa é manual, sendo necessária uma pessoa para realizar essa vigilância.

O desafio que se apresenta é como automatizar este processo de alocação/liberação de instâncias, como meio de manter a qualidade de serviço e impedir a fuga de clientes. Uma das maneiras seria utilizar *força-bruta* e alugar o máximo de servidores que o provedor de nuvem permitir, criando o cenário de superdimensionamento no número de clientes descrito

anteriormente, o que resulta em um gasto desnecessário com o aluguel de instâncias que ficariam ociosas. Outro desafio está relacionado ao custo-benefício das configurações disponibilizadas pelo provedor de nuvem, que nem sempre possuem um relacionamento linear, implicando que uma instância com capacidade de processamento *x* pode custar menos do que a metade de uma instância com capacidade de processamento *2x*. Neste trabalho, a capacidade computacional das instâncias é medida pelo número de núcleos (*cores*, em inglês) de processamento. A não-linearidade implica em preços por núcleo diferentes para configurações diferentes.

Uma maneira mais interessante para resolver o problema seria através da utilização de Teoria de Controle com Retroalimentação (TCR, *Feedback Control Theory*, em inglês) (Doyle; Francis; Tannenbaum, 1992) como ferramenta de monitoramento do sistema. A retroalimentação, realizada periodicamente, seria utilizada para que fosse averiguada a necessidade de aumentar ou diminuir a capacidade computacional do sistema, evitando, dessa forma, que servidores fiquem ociosos. A TCR é capaz de auxiliar, inclusive, na previsão de demanda, isto é, prever momentos em que o sistema ficará saturado e, com isso, alugar instâncias *antes* que tais situações aconteçam, evitando uma queda na QoS.

Porém, mesmo com a utilização de TCR para monitorar o sistema, existem casos onde o aumento do número de clientes é imprevisível e a retroalimentação pode não ser suficiente para que o sistema detecte esta mudança. Como exemplo, o texto (Armbrust *et al.*, 2010) cita que Animoto[1], uma empresa de criação de vídeos online, teve sua demanda aumentada de 50 servidores para 3.500 em 3 dias, ou seja, praticamente dobrava a cada 12 horas. Tal aumento ocorreu quando a empresa disponibilizou seus serviços via a rede social Facebook[2]. Apesar de cenários como esses serem raros, é possível que aconteçam em sistemas de comércio eletrônico. Mais provável, contudo, é que existam rajadas de novas requisições ocasionadas, por exemplo, quando a empresa realiza promoções-relâmpago com larga divulgação em redes sociais. Assim, a técnica a ser utilizada para monitorar o sistema deve prever estes *ruídos* e ser capaz de reagir a eles.

Em um ambiente tão competitivo como o de comércio eletrônico, a manutenção da QoS é de crucial importância. Assim, a empresa dona do sistema disponibilizado na nuvem pode oferecer a seus clientes Acordos de Nível de Serviço (ANS, *Service Level Agreements*,

1 http://animoto.com/
2 http://www.facebook.com

em inglês). Tais documentos preveem penalizações financeiras diretas (sob a forma de multas) ou indiretas (sob a forma de negociações não-concretizadas) caso a QoS prevista em contrato não seja cumprida (Freitas; Parlavantzas; Pazat, 2012), (Dan *et al.*, 2004), (Leitner *et al.*, 2012), (Boloor *et al.*, 2010), (Sakr; Liu, 2012), (Goiri *et al.*, 2012). Dessa forma, manter a QoS estabelecida nos ANS implica em evitar penalizações financeiras e reduzir os custos de manutenção do sistema. Contudo, mesmo sem a existência de um ANS, a manutenção da QoS impede que os clientes abandonem o sistema, causando não uma penalização, mas uma possível perda de receita.

Em sistemas paralelos tradicionais, onde o número de instâncias é fixo, a principal maneira de manter a QoS é fazer com que as requisições passem por um controle de admissão (Orleans, L. F.; Furtado, 2007), (Orleans, L.; Zimbrão; Furtado, 2008), (Orleans, L.; Oliveira, de; Furtado, 2007), (Schroeder *et al.*, 2006) no qual, possivelmente, algumas requisições seriam rejeitadas ou priorizadas/escalonadas. Em sistemas hospedados em nuvem, por outro lado, existe a possibilidade de aumentar rapidamente o número de instâncias alugadas, reduzindo a utilização do sistema e adaptando-o às mudanças da carga de trabalho sem a necessidade de sacrificar requisições.

1.1. Caracterização do Problema

Nesta seção o problema a ser resolvido nesta tese será descrito.

Considere um provedor de sistema (*sp*) cuja aplicação de comércio eletrônico está hospedada em uma nuvem IaaS (a seção 2.1. apresenta os tipos existentes de provedores de nuvens) mantida por um provedor de nuvem (*cp*) como, por exemplo, a Amazon EC2[3]. Assim, *cp* disponibiliza instâncias com diferentes configurações e preços para *sp*. Os servidores já alugados formam o *array* de instâncias de *sp*. *Cp* fornece a *sp* uma interface que permite a este último alugar novas instâncias de uma determinada configuração ou devolver um servidor para *cp.* A maneira como *sp* realiza os novos aluguéis e devolve as instâncias para *cp* formam o Modelo de Elasticidade seguido.

O valor devido pelo aluguel de cada instância é medido em instâncias-hora, ou seja, multiplicando o valor do custo de uma hora de uso pelo número de horas (arrendondado para

3 http://aws.amazon.com/pt/ec2/

cima) em que a instância ficou sob o poder de *sp*.

Considere, ainda, um cliente *c* que utiliza a aplicação mantida por *sp* . Esses papéis estão descritos na figura 1 e serão considerados ao longo de todo o trabalho.

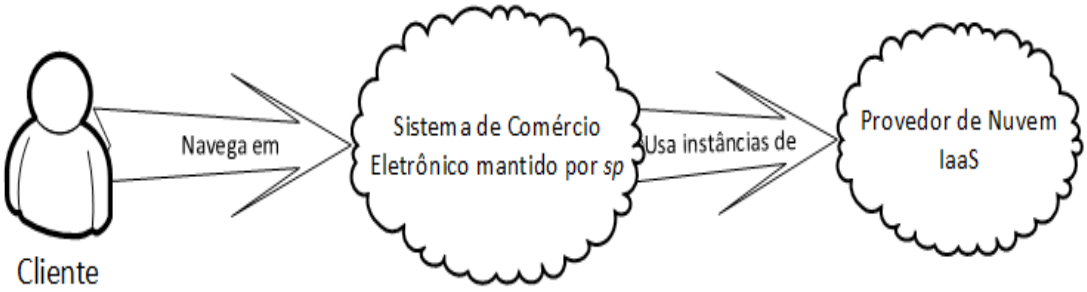


Figura 1: Modelo estudado.

O sistema hospedado na nuvem funciona da seguinte maneira: enquanto navega pelo sistema, o cliente envia diversas requisições (ao pesquisar produtos, visualizar comentários de outros clientes, enviar detalhes do cartão de crédito, etc.). Sempre que uma requisição é recebida, esta é processada por um servidor que esteja ocioso ou, no caso de todas as instâncias alugadas estarem ocupadas, é enviada para uma fila de requisições do tipo "Primeiro a Entrar, Primeiro a Sair" (*FIFO*, do inglês *First-In, First-Out*).

Caso suas requisições demorem muito para serem processadas, *c* abandona o sistema sem gerar qualquer lucro para *sp*. Este tempo máximo para o processamento de uma requisição é chamado neste trabalho de *deadline*.

Existe uma Taxa de Conversão de Clientes (TCC), isto é, o percentual de visitas ao sistema que se transformam em vendas (#vendas / #visitas), conhecida. Em cada visita *c* envia em média $\overline{n}$ requisições até a sua finalização, isto é, até abandonar o sistema ou concretizar a compra e o valor da transação ser debitado de seu cartão de crédito. Em média, cada venda gera uma receita bruta $\overline{rb}$. Assim, podemos determinar o valor médio de cada requisição, $\overline{v}$, como:

$$\overline{v} = \begin{cases} \overline{rb} \times \overline{TCC} \times \frac{1}{\overline{n}} & \text{, se não ultrapassar o deadline} \\ 0\text{, caso contrário.} \end{cases} \quad (1)$$

Uma outra forma de calcular a média de *v* seria introduzindo a probabilidade *q* de o tempo de processamento da requisição não ultrapassar o *deadline*. Assim:

$$\overline{v} = \overline{rb} \times \overline{TCC} \times \frac{1}{\overline{n}} \times q \tag{2}$$

Considerando que o sistema recebe uma média de $\overline{h}$ visitas por dia e que cada visita contenha, em média, $\overline{r}$ requisições, podemos calcular a receita bruta média que o sistema gera em um dia através de

$$\overline{rb_{dia}} = \overline{h} \times \overline{r} \times \overline{v} \tag{3}$$

Há outra possibilidade para calcular o valor médio de *v*, que seria utilizando o mesmo método descrito em (Mazzucco; Dyachuk; Dikaiakos, 2011), que consiste em dividir a média da receita obtida pelo número médio de requisições de um dia (equação 4).

$$\overline{v} = \frac{\overline{rb_{dia}}}{\overline{h} \times \overline{r}} \tag{4}$$

Contudo, a equação 4 torna-se inapropriada quando a distribuição da receita pelas requisições é cauda-longa (Crovella, 2001), isto é, pouquíssimas requisições são responsáveis pela maior parte da receita. Repare, contudo, que a equação 4 se aplica a cenários onde a receita obtida pelo sistema vem do número de visualizações de página, típico de *blogs* e outros sistemas cuja fonte principal de receita vem de publicidade por visualizações de página. Em nossos experimentos consideramos ambos os cenários.

Um dos nossos objetivos principais é aumentar *q* e, desta forma, majorar a receita bruta final obtida por *sp*. Outra variável que pode afetar a receita ao final de um dia é o custo *i* associado à alocação de instâncias, que é medido por hora, criando a unidade *instância-hora*. O relacionamento entre essas duas variáveis, *q* e *i*, se dá pelo fato de que, para diminuir a probabilidade de o tempo de resposta de uma requisição ultrapassar o *deadline*, deve-se aumentar o número de servidores contidos no *array* e, com isso, aumentar o valor de *i*. Assim, deve-se estudar tal relacionamento para tornar possível diminuir ambas as variáveis. Um dos

caminhos possíveis é utilizar a Teoria de Filas (Bocharov *et al.*, 2004) e observar as seguintes variáveis: média da taxa de chegadas de requisições ao sistema (λ), tempo médio de processamento de cada requisição (β) e tamanho da fila de requisições (δ). É possível combinar λ e β em uma única variável ρ, que representa a *utilização do sistema*, através de

$$\rho = \lambda \times \beta \tag{5}$$

Note que 1/β representa a taxa média de *saída* do sistema, isto é, o número de requisições que são processadas por unidade de tempo. Neste trabalho, consideramos que β é o tempo médio que uma requisição leva para ser processada por 1 núcleo de processamento. Dessa forma, a equação 5 torna-se

$$\rho = \frac{\lambda \times \beta}{s} \tag{6}$$

onde *s* é o total de núcleos existentes no *array* de instâncias. O *s* núcleos são agrupados em instâncias, que podem ter 1, 2 ou 4 núcleos, cada configuração possuindo seu próprio preço. As configurações podem apresentar preços por núcleo distintos, caracterizando um custo-benefício não linear.

Se ρ possuir um valor alto, o sistema está sob alta utilização, indicando que há mais requisições chegando ao sistema do que os servidores contidos no *array* são capazes de processar. Cabe ressaltar que altas taxas de chegada levam a problemas de desempenho em um sistema devido ao número excessivo de *page-faults* e *thrashing* do lado servidor, culminando em aumento na taxa de requisições que ultrapassam o tempo máximo de execução (Schroeder *et al.*, 2006). Por outro lado, baixa utilização significa que há recursos ociosos e que o custo com o aluguel de servidores pode ser reduzido.

Por fim, com uma probabilidade *z*, há uma rajada imprevisível de requisições, ocasionadas por razões diversas, incidindo sobre o sistema, elevando λ em um percentual z_p, com duração média de $\bar{t}$ segundos. Essa rajada é chamada, neste trabalho, de *ruído*. Os ruídos podem ser, por exemplo, a resposta a uma promoção-relâmpago realizada pela empresa de comércio eletrônico, ou um alto número de leitores de uma notícia exclusiva recém-lançada em um *blog*.

1.2. Objetivos

O principal objetivo deste trabalho é criar um Modelo de Elasticidade, que consiste em um grupo de técnicas e algoritmos capazes de reduzir o custo total das empresas que hospedam seus sistemas em nuvens IaaS, implicando na redução, ao mesmo tempo, da perda financeira relacionada à má QoS e do custo com alocação de instâncias

Ao mesmo tempo, a solução deve ser robusta o suficiente para detectar ruídos com rapidez e evitar que estes afetem a QoS experimentada pelos usuários do sistema.

1.3. Proposta

A proposta deste trabalho consiste em utilizar Teoria de Controle com Retroalimentação – mais precisamente Lógica Nebulosa (Berkan; Trubatch, 1997), (Wang, 1992) – para atingir o objetivo principal descrito na seção 1.2..

O relacionamento entre as variáveis q, δ e ρ não é simples de modelar. Não há um meio direto de determinar o grau de interferência entre elas, uma vez que o tempo de processamento de uma requisição pode ser influenciado por diversos fatores, como capacidade computacional disponível, quantidade de memória, largura de banda, etc..

Uma boa forma de capturar esse relacionamento, então, é através da utilização de uma Máquina de Inferência Nebulosa (MIN, ver seção 2.2.), que tem como principal vantagem tomar decisões baseadas em dados imprecisos. A MIN é uma técnica de TCR e permite determinar se é necessário alterar a quantidade de instâncias alocadas por *sp*. Ainda além, a MIN pode ser utilizada para prever variações na carga de trabalho, alugando novos servidores antecipadamente. Finalmente, como a MIN utiliza regras do tipo “SE-ENTÃO”, sua utilização permite ajustes finos por parte de especialistas, uma vez que estas regras são entendíveis por humanos e podem ficar em um arquivo-texto em separado.

Em nosso cenário, a ação tomada pela MIN para corrigir o número de instâncias se dá através de seu aluguel ou liberação, fazendo com que ela não produza resultados imediatos. Assim, uma vez tomada uma medida corretiva, a ferramenta entra em um estado de espera, ao fim do qual irá verificar se o número de instâncias adquiridas/liberadas foi eficaz ou não. O tempo entre tais medições é chamado nesta tese de *latência de medição*. Caso a latência seja

alta, o sistema vai demorar mais a reagir a qualquer variação na carga de trabalho. Caso a latência seja baixa, o sistema pode realizar duas medições – e duas operações corretivas – sem que haja tempo de a correção resultante da primeira medição ter sido consolidada. Assim, a latência deve ser calculada de forma que nenhum desses fenômenos aconteça.

Em conjunto com a MIN, foram criadas técnicas que realizam de forma eficiente as aquisições de servidores solicitadas por ela, aumentando a capacidade computacional total ao utilizar instâncias com as melhores relações de custo-benefício, buscando sempre a configuração menos custosa que seja capaz de atender aos requisitos de QoS. De modo análogo, as liberações de servidores são feitas avaliando a *oportunidade de liberação* de cada instância, isto é, devolve-se ao provedor de nuvem a instância que estiver mais próxima de aumentar o seu custo, que é medido por hora.

Para a detecção de ruídos, foram criados monitores, chamados nesta tese de *ganchos*, que verificam em tempo real, isto é, no processamento de cada requisição, se alguma das variáveis q, δ e ρ possui um valor alto atípico. Se for o caso, o sistema intervém e a rotina de expansão é chamada.

A esse conjunto de técnicas foi dado o nome de PROFUSE e a intenção é que elas possam ser utilizadas por empresas cujas fontes de receita são sistemas hospedados em nuvens IaaS.

O PROFUSE se baseia em dados históricos – mais precisamente em um *log* de acessos do sistema – para prever situações futuras e antecipar o comportamento da carga de trabalho. Tais dados históricos são, basicamente, a quantidade de requisições por hora (podendo ser outra unidade de tempo) que o sistema experimentou no passado, formando um Histograma de Acessos dividido por hora. Com isso, é possível determinar qual a carga de trabalho para um determinado momento e ajustar o *array* de instâncias de acordo. Como os ruídos são considerados aumentos imprevisíveis e raros, não há como capturá-los no Histograma de Acessos.

1.4. Metodologia

O objetivo principal desta tese é desenvolver uma técnica que possa ser utilizada independentemente de provedor IaaS. Assim, resolveu-se criar um simulador para que fosse

possível alterar parâmetros e restrições aqui estudados com agilidade. Como exemplos da utilidade desta abordagem, pode-se citar a possibilidade de alternar o custo-benefício das diferentes configurações das instâncias entre linear e não-linear, bem como remover a restrição do número máximo de instâncias que um cliente pode alugar.

Partindo da premissa descrita acima, para validar os algoritmos propostos, foi criado um Simulador de Eventos Discretos (SED) (Banks; Carson, 1984). Chamado de Cloud-Simulator, a ferramenta implementa todas as técnicas utilizadas pelo PROFUSE, bem como o cenário de funcionamento do sistema descrito na seção 1.1..

Para que a simulação apresentasse resultados corretos, utilizamos as informações contidas em (Mao; Humphrey, 2012), que descreve detalhes sobre o comportamento das máquinas virtuais em ambientes de nuvens IaaS, como, por exemplo, o tempo médio de iniciação de uma nova instância.

Segundo (Felipini, 2003), um sistema de comércio eletrônico possui uma Taxa de Conversão de Clientes (TCC), isto é, o percentual de visitas que se tornam efetivamente em vendas, entre 0,5% e 4%, considerando uma taxa de 1% como um valor “bem razoável para o primeiro ano de atividade”. Como um dos públicos-alvo em potencial para esta tese são *startups*, utilizamos essa taxa em nossos experimentos.

Ademais, os seguintes parâmetros utilizados na simulação tiveram seus valores extraídos de outros trabalhos:

1. *Cargas de trabalho*: Foram utilizados dois tipos de cargas de trabalho distintas, ambas descritas em (Mao; Humphrey, 2011) como sendo típicas de sistemas hospedados em nuvens. Uma das cargas de trabalho utilizadas nesta tese pertence a um sistema real, servindo como um critério de avaliação oficial da proposta.

2. *Tempo do deadline*: A informação relacionada ao tempo máximo de cada requisição foi depreendida de (“Selfish, mean, impatient customers: New Thinking: Gerry McGovern”, [s.d.]), texto no qual se afirma que a paciência dos usuários de sistemas *web* em geral dura 4 segundos.

3. *Valor agregado de cada requisição para sistemas com publicidade estática*: O cálculo

do valor agregado de cada requisição nestes casos foi extraído de (Mazzucco; Dyachuk; Dikaiakos, 2011), onde os autores dividem a receita total pelo número de requisições.

1.5. Resumo dos Resultados

Em nossos experimentos, o PROFUSE manteve a QoS – isto é, processou todas as requisições dentro do *deadline* – em vários cenários, inclusive com a presença de ruídos grandes, que aumentaram a carga de trabalho em 40%. Nossos resultados mostram que o PROFUSE é capaz de prever e reagir às variações da carga de trabalho de um sistema, tomando decisões acertadas sobre aluguel e/ou liberação de novos servidores.

Igualmente, os experimentos realizados mostram que a estratégia utilizada pelo PROFUSE para alugar novos servidores junto ao provedor de nuvem é altamente eficiente, resultando em um custo nesta atividade de até 2 ordens de magnitude inferior ao do superdimensionamento, que consiste em alugar o máximo de servidores possíveis, todos contendo a configuração mais poderosa.

1.6. Divisão do Texto

A tese está dividida da seguinte maneira, além desta introdução: o capítulo 2 fornece o embasamento teórico necessário para o perfeito entendimento da proposta.

O capítulo 3 lista os trabalhos relacionados, diferenciando-os deste.

O capítulo 4 apresenta o Modelo de Elasticidade PROFUSE, descrevendo suas funcionalidades, algoritmos, arquitetura, detalhes sobre a criação da Máquina de Inferência Nebulosa utilizada, como funciona a técnica de aluguel de novos servidores e como e quando os ganchos são ativados pelos ruídos.

O capítulo 5 apresenta os resultados obtidos com os experimentos realizados, evidenciando a eficácia e a eficiência do PROFUSE.

Por fim, o capítulo 6 apresenta as conclusões obtidas com esta tese e aponta direções para pesquisas futuras tendo esta como base.

Capítulo 2 - Revisão Teórica

2.1. Computação em Nuvem

Por ser um tema recente de pesquisa, diversas fontes definem Computação em Nuvem (CN) de forma diferente. Neste capítulo, descreveremos brevemente a motivação para a CN, bem como os papéis existentes neste novo modelo. Definiremos o conceito de CN que adotamos para este trabalho, citando as principais características deste novo paradigma, como ele se diferencia da Computação Distribuída tradicional, as possíveis formas que pode assumir e, por fim, listaremos alguns dos principais serviços de CN comerciais.

Motivação

A popularização da internet e dos meios de divulgação, como as redes sociais, tornou extremamente difícil prever a quantidade de clientes que um novo serviço pode receber. Por exemplo, um site de leilão online, dependendo da oferta e da divulgação feita pelos clientes em redes como Twitter[4] e Facebook, pode receber milhões de acessos em poucos minutos. Esta imprevisibilidade leva as empresas a superestimarem a quantidade de recursos necessária para atender à demanda dos clientes, gerando gastos desnecessários com energia, *hardware* e licenças de *software* que ficarão ociosos em situações que não sejam de pico. (Mazzucco; Dyachuk; Dikaiakos, 2011), (Dyachuk; Mazzucco, 2010), (Greenberg *et al.*, 2008), (Mazzucco; Dumas, 2011), (Buyya *et al.*, 2009).

Os provedores de CN trouxeram uma solução para este problema, possibilitando que as empresas alugassem seus servidores e lá hospedassem seus sistemas. Em situações de pico inesperado, os administradores poderiam solicitar a adição de novos recursos – liberando-os quando não fossem mais necessários – e pagar somente pelo tempo em que eles foram utilizados. Desta forma, tanto faz possuir um servidor executando por mil horas ou mil servidores executando por uma hora, o preço a ser pago é o mesmo. Esta característica de aumentar e diminuir a quantidade de recursos alocados para um sistema e de forma rápida é conhecida como *elasticidade,* sendo um dos conceitos-chave desta tese. As políticas para alocação e liberação de recursos em CN constitui um problema em aberto e vem atraindo

4 http://www.twitter.com

grande atenção da comunidade científica (Rajavel; Mala, 2012), (Konstanteli *et al.*, 2012), (Freitas; Parlavantzas; Pazat, 2012), (Leitner *et al.*, 2012), (Boloor *et al.*, 2010), (Dyachuk; Mazzucco, 2010), (Suleiman *et al.*, [s.d.]), (Sakr; Liu, 2012), (Dutta *et al.*, 2012) e (Goiri *et al.*, 2012).

Papéis na Computação em Nuvem

Segundo (Armbrust *et al.*, 2010) e (Zhang; Cheng; Boutaba, 2010), é possível definir três papéis típicos em Computação em Nuvem (figura 2):

1. *Provedor da nuvem*: sua função é disponibilizar servidores para que serviços/sistemas sejam instalados. Normalmente este papel é encarregado, ainda, de definir algum meio para que o usuário da nuvem (ver abaixo) possa solicitar serviços pertinentes, como realização de backups ou alteração da configuração dos recursos;

2. *Usuário da nuvem*: utiliza os serviços do provedor da nuvem, através do conceito que é conhecido como *computação utilitária*, no qual os recursos (servidores, banda de rede, etc) são cedidos pelo provedor ao usuário através de um modelo de "pague-pelo-uso". Em outras palavras, os sistemas que estão hospedados na nuvem podem começar utilizando poucos servidores e, caso seja necessário, o usuário solicita ao provedor a cessão de novas instâncias para atender à demanda. Quando esta diminuir, o usuário as devolve para o provedor e só é cobrado pelo tempo em que estes recursos foram utilizados.

3. *Usuário final*: é o cliente do serviço que está hospedado na nuvem.

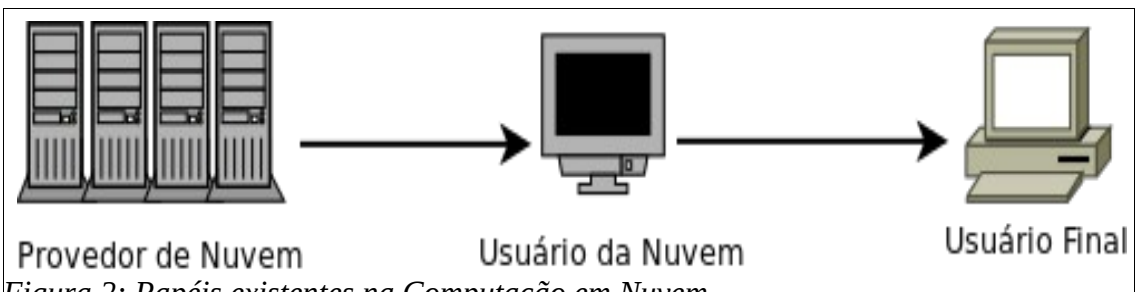


Figura 2: Papéis existentes na Computação em Nuvem

Definição

O termo “nuvem”, apesar de não ser novo (era utilizado para descrever grandes redes de caixas eletrônicos nos anos 90), foi utilizado pelo CEO da Google, Eric Schmidt, para descrever o modelo de negócios de sua empresa, a disponibilização de diversos serviços na internet, em 2006. A partir de então, ganhou popularidade e passou a receber atenção da comunidade científica, recebendo diversas definições formais. Para este trabalho, foi adotada a definição de Computação em Nuvem extraída de (Zhang; Cheng; Boutaba, 2010):

> Computação em Nuvem é um modelo que possibilita o acesso via rede, de modo conveniente, a um conjunto de recursos computacionais que são, por sua vez, configuráveis e compartilhados, como, por exemplo, redes, servidores, armazenamento, aplicações e serviços. Estes recursos devem ser provisionados e/ou liberados rapidamente, com o mínimo esforço de gerenciamento possível.

Outra definição para o termo Computação em Nuvem é extraída de (Armbrust *et al.*, 2010):

> Computação em Nuvem se refere tanto às aplicações que são disponibilizadas como serviços através da internet quanto ao *hardware* e *software* nas centrais de processamento onde tais serviços estão hospedados.

Através das definições acima podemos afirmar que a aquisição e a liberação de recursos de forma dinâmica, isto é, de forma rápida – ou até mesmo *automática*, está no coração deste novo paradigma. Se pensarmos que a aquisição de recursos pode ser infinita, outra característica da CN nos é apresentada: *computação infinita*, ou seja, é possível assumir que a quantidade de recursos disponíveis é infinita.

Arquitetura da Computação em Nuvem

A CN possui uma arquitetura em 3 camadas, cada uma executando sobre a outra

(figura 3). As camadas principais, do nível mais baixo para o mais alto, são: *infra-estrutura*, que envolve *hardware*, sistema operacional e virtualização, *plataforma*, englobando framework de desenvolvimento, sistemas de armazenamento, etc., e a *aplicações*, que são as aplicações hospedadas na nuvem.

Cada camada é um conjunto coeso, que pode ser oferecido como um serviço independente. Assim, os serviços em nuvem que podem ser contratados dividem-se em:

1. *Infra-estrutura como Serviço (IaaS*, sigla em inglês): é o provedor que fornece o *hardware*, o serviço de virtualização, se preocupa com a manutenção e aquisição do *hardware*, entre outros. Um exemplo de provedor IaaS é o Amazon EC2.

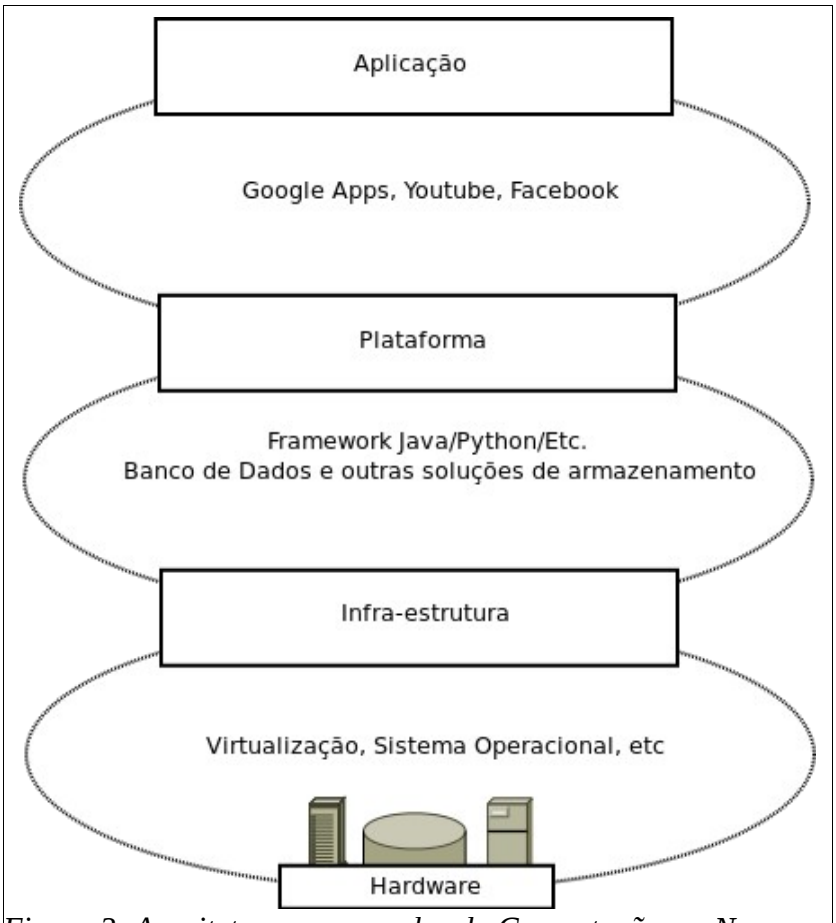


Figura 3: Arquitetura em camadas da Computação em Nuvem.

2. *Plataforma como Serviço (PaaS*, sigla em inglês): refere-se ao provedor que fornece algum tipo de plataforma que as aplicações hospedadas irão usufruir, como um

servidor web ou algum framework/linguagem de programação específicos. Como exemplo de um provedor PaaS, pode-se citar o Google App Engine.

3. *Software como Serviço (SaaS*, sigla em inglês): são sistemas oferecidos na internet, hospedados em uma nuvem. Como exemplos de provedores SaaS, existem Google Apps, Facebook, Youtube, entre outros.

Tecnologias Relacionadas a Computação em Nuvem

A CN geralmente é comparada com as tecnologias abaixo, principalmente pelo fato de elas compartilharem algumas características:

1. *Computação em Grade*: A Computação em Grade (CG) é um paradigma de computação distribuída que coordena recursos coordenados para atingir um objetivo comum. A CN se assemelha à CG pelo fato de também utilizar recursos distribuídos. Contudo, a CN vai um passo além ao utilizar tecnologias de virtualização para alcançar o provisionamento dinâmico de recursos (elasticidade).

2. *Computação Utilitária:* A Computação Utilitária representa um o modelo de prover recursos sob demanda e cobrar os clientes baseado na utilização. A CN pode ser vista como a concretização da Computação Utilitária.

3. *Virtualização:* A Virtualização é a tecnologia que abstrai os detalhes físicos de *hardware* e possibilita a associação de *hardware virtualizado* para as aplicações. A virtualização, com a sua capacidade de alocar recursos virtuais de forma dinâmica, forma a base da CN.

Principais Características da Computação em Nuvem

A seguir estão listadas as principais características da CN.

1. *Múlti-hóspede:* Esta propriedade, em inglês chamada de *multi-tenancy*, indica a

habilidade de serviços de vários provedores ficarem hospedados no mesmo datacenter. A preocupação com desempenho e gerenciamento destes serviços são divididas entre os provedores de serviços e o provedor de infra-estrutura. Esta característica difere-se da *virtualização* por se referir a clientes compartilhando aplicações, sistema operacional, *hardware*, etc, enquanto aquela se refere à arquitetura onde os componentes são abstratizados, permitindo que cada cliente tenha a impressão de estar utilizando um servidor dedicado.

2. *Utilização compartilhada de recursos*: o provedor da infra-estrutura oferece um conjunto de recursos que podem ser associados dinamicamente a vários consumidores, permitindo uma maior gerência por parte desse dos custos operacionais. Por exemplo, um provedor IaaS pode descobrir uma maneira melhor de realizar migração de máquinas virtuais de um servidor para outro, aumentando, assim, a utilização total de seus recursos e, consequentemente, diminuindo o custo operacional total.

3. *Orientação a serviços*: Em CN, cada provedor IaaS, PaaS ou SaaS oferece seus serviços tendo como base um Acordo de Nível de Serviço, onde são estabelecidas algumas métricas que devem ser garantidas pelo provedor, como tempo de resposta, disponibilidade mínima, etc., sob pena de sanções financeiras. Assim, seguir os ANS é um objetivo crítico para os provedores.

4. *Precificação baseada em utilização*: A CN adota um modelo de precificação que é baseado na utilização dos recursos pelo cliente, com esta utilização sendo, normalmente, medida por **tempo** (CPUs por hora) ou por **transferência de dados** (quantidade de bytes transmitida pela rede ou em um disco)**.** Desta forma, um fenômeno interessante ocorre: a *associatividade de custo*, onde o cliente pagará a mesma quantia para utilizar mil servidores por uma hora ou um servidor por mil horas.

5. *Elasticidade*: Esta característica refere-se à habilidade de responder rapidamente às mudanças inesperadas na carga de trabalho de um sistema. Podemos tomar como exemplo um sistema de comércio eletrônico, onde a medida mais importante é o

tempo de resposta de cada requisição enviada pelos consumidores. A elasticidade, neste caso, seria responsável por aumentar o número de recursos utilizados pelo sistema, deixando-o capaz de manter os tempos de resposta baixos mesmo que o número de requisições simultâneas seja alto.

Tipos de Elasticidade

O provisionamento de recursos de forma rápida é normalmente atingido através de *virtualização*, ou seja, utilização de máquinas virtuais. Desta forma, a expansão ou retração de recursos pode acontecer de duas maneiras:

1. *Expansão Horizontal*:caracteriza-se por alocar novas instâncias de máquinas virtuais a um serviço hospedado. Normalmente utiliza-se uma *imagem* (similar às imagens ISO de CDs) de toda a pilha de *softwares* e sistema operacional de forma a agilizar a criação das novas instâncias. Sua utilização é simples e de baixo risco: basta iniciar novas instâncias de máquinas virtuais com a imagem do sistema e incluí-las no balanceador de carga. Na literatura, este espelhamento de máquinas virtuais também é chamado de *clonagem.* Neste trabalho, utilizamos ambos os termos como sinônimos. Quando uma ou mais instâncias não forem mais necessárias, basta desligá-las e, claro, excluí-las do balanceamento de carga. A figura 4 ilustra este tipo de expansão.

2. *Expansão Vertical*: este tipo, introduzido recentemente pela habilidade de alguns gerenciadores de máquinas virtuais em alterar suas configurações sem a necessidade de desligamento, caracteriza-se por aumentar a quantidade de recursos nas máquinas virtuais já existentes, como, por exemplo, a quantidade de memória ou CPUs. Sua utilização é de grande complexidade e risco, uma vez que a adição de novos recursos ocasiona uma diminuição temporária do desempenho daquela máquina virtual, causando um impasse: a expansão de recursos é solicitada por causa do fraco desempenho do sistema e esta solução estrangula ainda mais o sistema. A redução na configuração das máquinas virtuais, ou seja, a diminuição de recursos disponíveis, como memória, necessita de uma análise criteriosa do novo cenário. Em algumas

situações, a quantidade de recursos restante pode ser insuficiente para continuar atendendo a carga de trabalho de forma eficiente. A figura 5 ilustra este tipo de expansão.

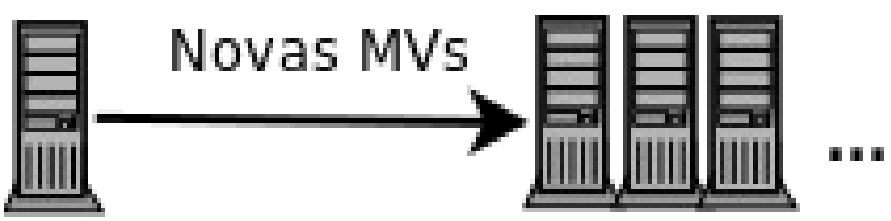


Figura 4: A expansão horizontal se dá através da alocação de novas máquinas virtuais.

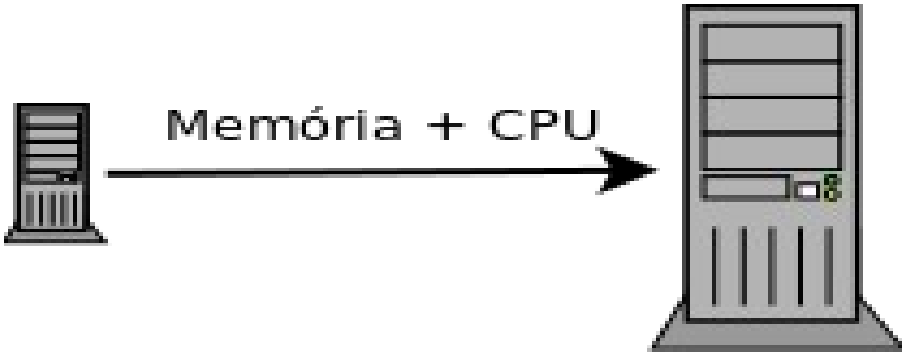


Figura 5: A expansão vertical se dá ao acrescentar/remover recursos à máquina virtual sem a necessidade de desligá-la.

Existe, ainda, a possibilidade de utilizar as duas técnicas de expansão, vertical e horizontal, em conjunto. O trabalho (Ardagna *et al.*, 2012) apresenta diversos tipos de padrões de expansão para Computação em Nuvem, combinando multi-hospedagem com os tipos possíveis de expansão. Abaixo, listamos e explicamos sucintamente cada um dos padrões.

1. Padrão Plataforma Simples (SPP ou Single Platform Pattern, em inglês): o mais básico dos padrões. Cada servidor possui apenas um hóspede, o que o torna altamente gerenciável e, normalmente, apresenta baixa utilização dos servidores.

2. Padrão de Plataforma Compartilhada (ShPP ou Shared Platform Pattern, em inlgês): cada servidor hospeda mais de um serviço. Apresenta alta utilização, mas de gerência mais difícil. Utiliza expansão vertical.

3. Padrão de Plataforma em Cluster (CPP ou Clustered Platform Pattern, em inglês): uma única plataforma, com suporte à clusterização, é criada e compartilhada entre todos os hóspedes. Quando necessário, utiliza expansão horizontal. Segundo os autores do trabalho, este é o modelo de expansão utilizado pelo Google App Engine.

4. Padrão de Plataforma Compartilhada Múltipla (Multiple ShPP, em inglês): é uma extensão do ShPP, utilizando tanto a expansão vertical como a horizontal. Há um algoritmo de decisão, para escolher qual dentre os dois modos de expansão será utilizado. Este padrão é o utilizado no trabalho (Dutta *et al.*, 2012), onde a ferramenta SmartScale é apresentada.

5. Padrão de Plataforma em Cluster Múltplia (Multiple CPP, em inglês): é uma extensão do CPP, provendo uma expansão horizontal em dois níveis de abstração. Todos os hóspedes são instalados em um cluster inicial que, caso seja necessário, será aumentado horizontalmente. Caso um ou mais serviços necessitem de mais poder computacional, um novo cluster é criado e alguns destes hóspedes são migrados.

Nesta tese é utilizado o termo Modelo de Elasticidade para se referir ao algoritmo utilizado para determinar *quantos* servidores devem ser alugados/liberados em conjunto com o algoritmo utilizado para escolher *como* os servidores serão alugados (quais configurações e em qual ordem).

2.2. Teoria de Controle com Retroalimentação

Teoria de Controle com Retroalimentação (TCR, *Feedback Control Systems*, em inglês) é um ramo onde engenharia e matemática que lida com o controle de sistemas dinâmicos com entradas. Seu objetivo usual constitui no cálculo de medidas corretivas que

resultem em levar o sistema novamente a um estado de estabilidade (Doyle; Francis; Tannenbaum, 1992). A figura 6 apresenta o formato de um sistema que empregue TCR.

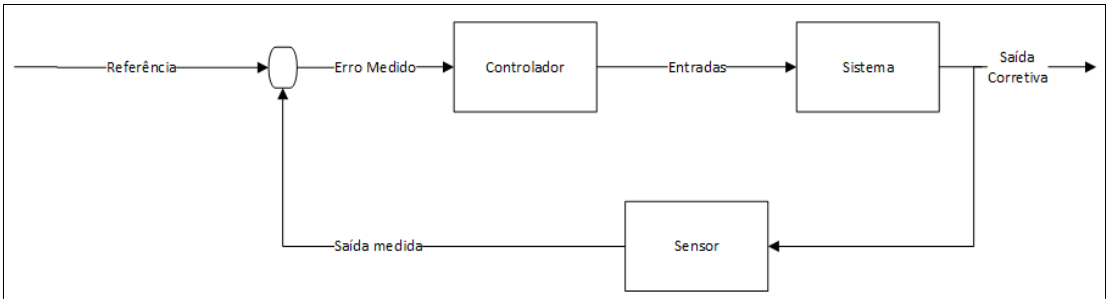


Figura 6: Arquitetura típica de um sistema que utilize TCR.

Basicamente, esta técnica funciona da seguinte maneira: a entrada externa, chamada de *referência*, é enviada pelo controlador para o sistema. Este tem a responsabilidade de realizar ajustes para devolver a estabilidade ao sistema. Um sensor realiza, periodicamente ou em tempo real, medições que servem de entrada para a próxima tomada de decisão.

A maneira como o sistema realiza o cálculo do ajuste necessário para estabilização pode ser feito com diversas ferramentas: Redes Neurais (Gurney, 1997), Árvores de Decisão (Rokach, 2007), Inferência Nebulosa (Berkan; Trubatch, 1997), Inferência de Bayes (Kelly; Smith, 2011), entre outras.

Neste trabalho optamos pelo uso da Inferência Nebulosa, que tem como principal vantagem o fato de poder ser definida em um formato legível e entendível por humanos. Esta característica permite que o cálculo final da inferência seja ajustado por um especialista, caso haja necessidade de uma intervenção humana. Esta possibilidade é melhor explicada na seção 4.4., quando a utilização da Inferência Nebulosa na proposta é detalhada.

Lógica Nebulosa

No mundo atual, o conhecimento de um ser-humano tem se tornado cada vez mais importante – nós o acumulamos através de experiências vividas e utilizamos nosso cérebro para raciocinar e criar uma ordem para a massa de informação que chega. Como somos todos limitados em nossa habilidade de perceber o mundo e raciocinar de forma profunda, nos encontramos o tempo todo confrontados pela *incerteza*, que é o resultado de falta de

informação (impressão léxica, incompletude), em particular, imprecisão das medidas.

Outra limitação na busca pela precisão da informação está no fato da utilização de uma linguagem natural para descrever/compartilhar conhecimento, comunicações, etc.. Apesar de conseguirmos nos comunicar através dela, a linguagem natural é vaga, permitindo a ocorrência de maus-entendidos, ambiguidades, e outros fenômenos indesejáveis.

Nossa percepção de mundo real é rodeada de conceitos que não possuem fronteiras bem definidas – por exemplo *muitos, alto, muito maior que, jovem*, são conceitos que são verdadeiros até um certo grau, e que são igualmente falsos para outros graus. Estes conceitos, ou fatos, podem ser chamados de *fuzzy* (difusa ou nebulosa, em português), e nosso cérebros conseguem trabalhar com eles, enquanto computadores podem ser ineficientes para processar este tipo de informação, uma vez que trabalham com sequências de 0s e 1s. Linguagens de programação são de um nível muito mais baixo do que as linguagens naturais e, por serem mais próximas de uma linguagem que o computador entenda, não são nebulosas. Chamamos este tipo de linguagem de *crisp* ou, em português, de nítida.

A seguir, descrevemos algumas das principais características dos sistemas difusos, necessárias para o entendimento deste trabalho. Faremos comparações com sistemas nítidos, quando for pertinente.

Figura 7: Sistema de Lógica Nebulosa

Conjuntos Nebulosos

Os conjuntos clássicos associam, através de suas funções características, os valores 1 ou 0 para os elementos neles contidos, discriminando, dessa forma, aqueles que são membros do conjunto daqueles que não são. Por exemplo, se consideramos o seguinte conjunto:

$$x \in X \mid x \text{ é par.}$$

Poderíamos reescrevê-lo da seguinte forma: multiplique por 1 todo número par; multiplique por 0 todo número ímpar. Este princípio segue a lógica Booleana, onde um elemento pertence ou não a um determinado conjunto.

Os conjuntos nebulosos, por sua vez, não são tão rígidos. Com eles, é possível definir graus de pertinência aos elementos, informando o quanto eles pertencem a um determinado conjunto. O grau de pertinência μ de um elemento a um conjunto nebuloso varia de 0 (não está contido) a 1 (completamente contido), sendo μ um número real. Abaixo segue um exemplo de um conjunto nebuloso:

Analisando a figura 8, é possível perceber que, à medida que X cresce, seu grau de pertinência também aumenta, até μ chegar a 1, que é o valor de X que indica que X está totalmente contido no conjunto nebuloso. A partir deste ponto, conforme X continua a crescer, seu grau de pertinência diminui, até chegar a 0.

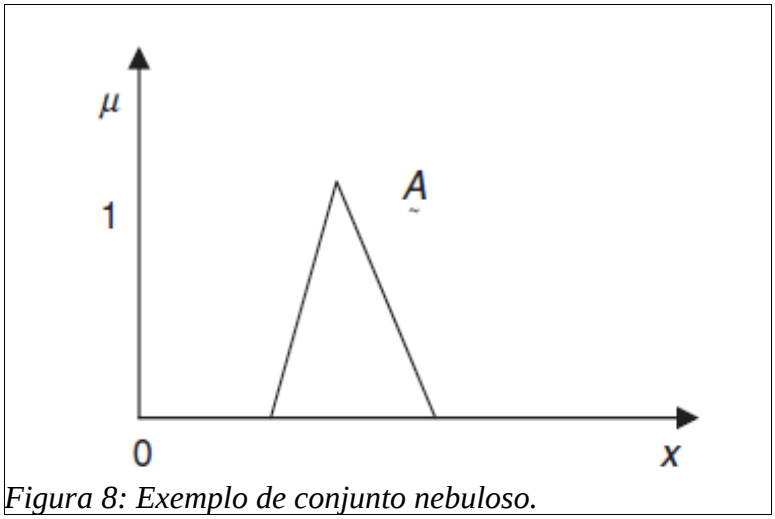


Figura 8: Exemplo de conjunto nebuloso.

Fazendo um paralelo com a percepção humana, podemos pensar no gráfico acima como um conjunto nebuloso para pessoas de *estatura mediana*, sendo X a altura de um indivíduo. Quando X é muito baixo, a percepção é de um indivíduo baixo e, conforme seu valor vai aumentando, a percepção de que sua estatura é mediana começa a crescer, até atingir um ápice. A partir daí, quanto maior sua estatura, o indivíduo vai deixando o conjunto dos medianos para entrar no conjunto das pessoas altas.

Neste trabalho, utilizamos os termos conjunto nebuloso e *região nebulosa* como

expressões análogas.

Funções de pertinência

O grau de pertinência de um indivíduo a uma região nebulosa está diretamente ligado à sua função de pertinência. A função de pertinência pode ser apresentada através de sua representação gráfica, podendo assumir diferentes formas. O modo com que a função de pertinência assume revela muito sobre a percepção daquela região nebulosa, devendo, portanto, ser escolhida com cuidado.

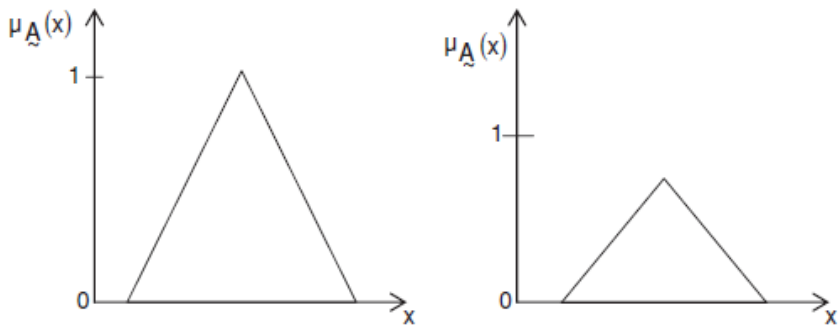
As funções de pertinência podem ser classificadas de acordo a forma que assume, conforme abaixo:

1. *Regiões nebulosas normais:* são as regiões em que os elementos nela contidos possam estar totalmente inseridos. Em outras palavras, a função de pertinência, em algum momento, alcança o valor 1. A figura 9a apresenta um exemplo de um conjunto nebuloso normal.

2. *Regiões nebulosas subnormais*: são as regiões em que os elementos nela contidos não chegam a ficar totalmente inseridos, ou seja, sua função de pertinência não alcança o valor 1. A figura 9b apresenta um exemplo de conjunto nebuloso subnormal.

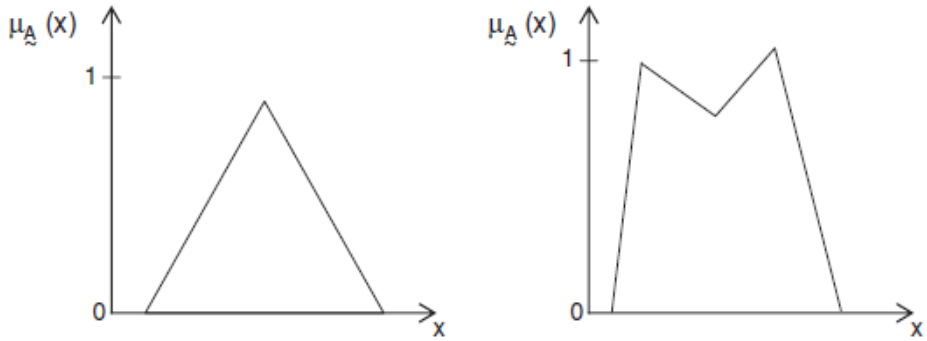
3. *Regiões nebulosas convexas*: se uma função de pertinência possui valores que aumentam ou diminuem monotonicamente, ou ainda se os valores aumentam ou diminuam monotonicamente conforme o valor de X aumenta, então esta região nebulosa é chamada de *convexa*. A figura 10a mostra um exemplo de conjunto nebuloso convexo.

4. *Regiões nebulosas não-convexas*: se uma função de pertinência não possui valores que aumentam ou diminuem monotonicamente, ou ainda se os valores não aumentam ou diminuam monotonicamente conforme o valor de X aumenta, então esta região nebulosa é chamada de *não-convexa*. A figura 10b mostra um exemplo de conjunto nebuloso não-convexo.



(a) *(b)*

Figura 9: Exemplos de região nebulosa normal (a) e subnormal (b).



(a) *(b)*

Figura 10: Exemplos de regiões nebulosas convexa (a) e não-convexa(b).

Ainda, é importante notar que, diferentemente de conjuntos nítidos, as regiões nebulosas não são mutualmente exclusivas. Um elemento pode estar presente em dois conjuntos ao mesmo tempo. Por exemplo, um indivíduo pode possuir grau de pertinência para o conjunto das pessoas de estatura mediana 0.6, enquanto possui grau de pertinência para o conjunto das pessoas altas de 0.4. A figura 11 mostra um exemplo de sobreposição de regiões nebulosas.

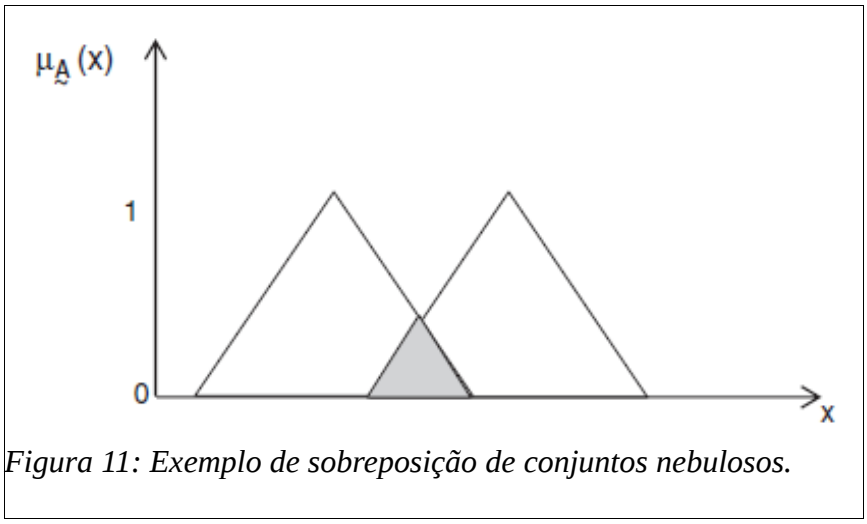


Figura 11: Exemplo de sobreposição de conjuntos nebulosos.

Outro conceito importante relacionado a Sistemas Nebulosos é o processo de *fuzzyficação*, que consiste em associar valores nítidos a graus de pertinência. Existem diversos meios de fazer isso, entre eles *intuição* e *inferência*. Analogamente, o processo de *defuzzyficação* consiste em transformar um valor nebuloso em um valor nítido. Dentre as técnicas utilizadas para este processo, estão a do *centróide* e o *princípio da pertinência maior.*

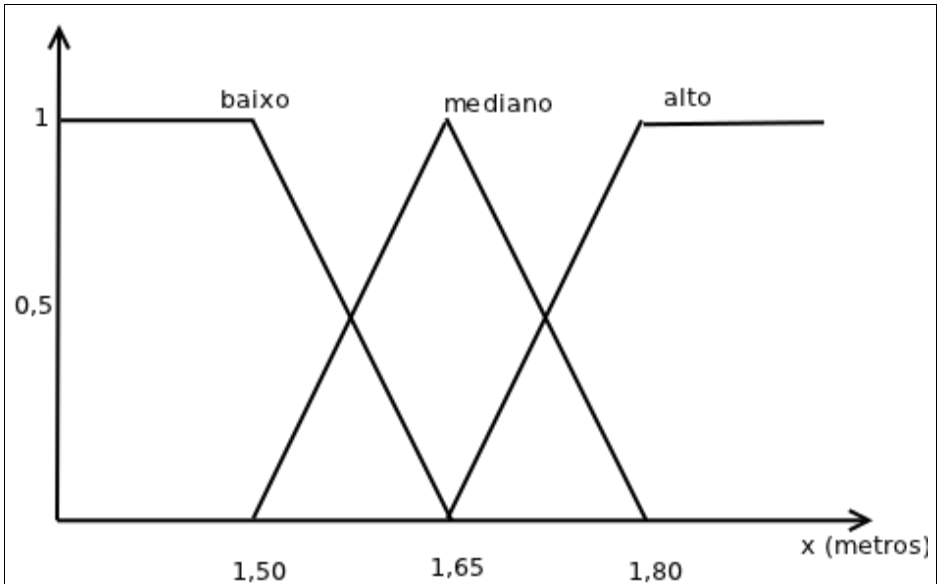


Figura 12: Regiões nebulosas para a variável linguística "altura".

As técnicas utilizadas neste trabalho para fuzzyficação e defuzzyficação são explicadas com mais detalhes na seção abaixo, que apresenta os Sistemas Nebulosos

Baseados em Regras – aqui chamados de Máquinas de Inferência Nebulosas, proposto por (Wang; Mendel, 1992).

Máquina de Inferência Nebulosa

O trabalho em (Wang; Mendel, 1992) apresenta um modelo de predição baseado em sistemas nebulosos, através da construção de regras do tipo IF-THEN a partir de um conjunto inicial de dados numéricos. Em outras palavras, o trabalho explica como criar uma máquina de inferência utilizando lógica nebulosa. Como todo sistema nebuloso, há dados de entrada e dados de saída e, para montar esta máquina de inferência, é necessário que este tipo de informação seja coletada previamente. Este modelo é constituído de 5 passos principais:

1. *Dividir os espaços de entrada e saída em regiões nebulosas*: este passo consiste em decidir pela quantidade de regiões nebulosas que se vai utilizar para cada uma das variáveis envolvidas sejam elas de entrada ou de saída. Por exemplo, podemos dividir a variável linguística *altura* em *baixo, mediano e alto.*

2. *Gerar regras nebulosas, do tipo SE-ENTÃO, a partir de um conjunto inicial de dados*: este passo consiste em analisar os dados iniciais, fuzzyficá-los e criar regras do tipo SE-ENTÃO. Por exemplo, se a entrada de nossa máquina de inferência consistir em 2 variáveis (peso e altura) e a saída consistir em uma variável (risco de doenças coronarianas), a seguinte linha:

Tabela 1: Exemplo de dados iniciais.

Peso (Kgs)	Altura (metros)	Risco (%)
110	1,54	80

Poderia ser transformada na seguinte regra:

SE PESO É ALTO E ALTURA É BAIXA ENTÃO RISCO É ALTO

O processo de fuzzyficação sugerido pelo trabalho é associar cada entrada à região nebulosa onde ela contenha o maior grau de pertinência.

3. *Associar a cada regra um peso*, para que seja possível resolver conflitos entre as regras. No passo anterior, é possível que sejam criadas regras que tenham a mesma entrada mas saídas diferentes. Como exemplo, tomemos a seguinte linha de dados em adição à anterior:

Tabela 2: Exemplo de dados iniciais.

Peso (Kgs)	Altura (metros)	Risco (%)
108	1,56	70

Produzindo a seguinte regra:

SE PESO É ALTO E ALTURA É BAIXA ENTÃO RISCO É MÉDIO

A parte antes do “SE” é igual nas duas regras, contudo as saídas são diferentes. Para resolver esta questão, o trabalho propõe a resolução destes conflitos utilizando opiniões de especialistas, já que os dados coletados anteriormente podem conter algumas distorções.

4. *Criar uma base de conhecimento inicial*, combinando as regras geradas no passo 2 com algumas regras criadas por pessoas especialistas. Em nosso exemplo, um cardiologista, após analisar as regras geradas anteriormente, poderia ter notado a ausência de uma importante regra:

SE PESO É BAIXO EALTURA É ALTA ENTÃO RISCO É BAIXO

5. *Determinar o mapeamento do espaço de entrada para o espaço de saída* baseado na base de conhecimento (obtida no passo anterior), utilizamos um método de defuzzyficação apropriado. O trabalho propõe o método de defuzzyficação do centróide, que consiste na aplicação da seguinte fórmula:

$$z^* = \int \frac{\mu_{\underset{\sim}{C}}(z)\, z \mathrm{d}z}{\mu_{\underset{\sim}{C}}(z)\, \mathrm{d}z}, \tag{5}$$

Basicamente, o que esta função faz é encontrar o centro de gravidade da região nebulosa. A figura mostra graficamente qual seria o escalar retornado no processo de defuzzificação utilizando o método do centróide:

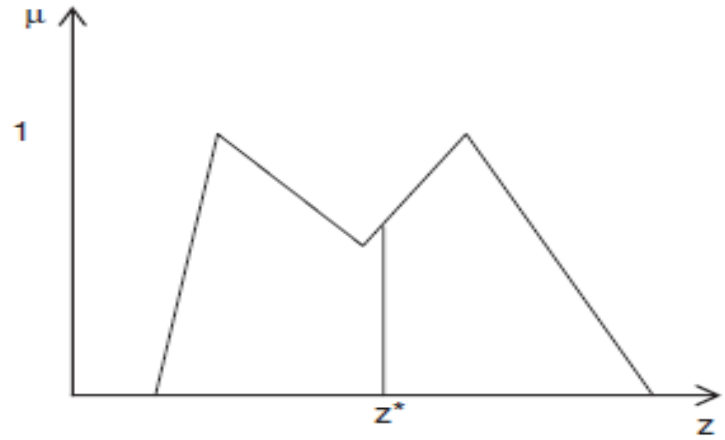


Figura 13: Defuzzificação utilizando o método do centróide.

2.3. Discussão

Este capítulo apresentou uma breve descrição sobre o que é a Computação em Nuvem, suas principais características, sua arquitetura típica, os possíveis tipos de expansão e alguns dos principais serviços em nuvem que existem hoje.

Contextualizando com esta tese, foram criadas técnicas que permitissem automatizar a elasticidade horizontal, tanto de forma reativa quanto de forma proativa. Desta forma, este trabalho foi desenvolvido com o intuito atender um problema típico de sistemas hospedados em provedores IaaS: manter a qualidade do serviço oferecido aos usuários finais, independente de haver variações (imprevisíveis ou não, bruscas ou não) na carga de trabalho sem, contudo, permitir que instâncias fiquem ociosas devido a um superdimensionamento de

demanda.

A elasticidade, hoje, é um conceito fundamental para manter a competitividade de sistemas de vendas disponíveis na internet. Segundo o estudo apresentado em (“Selfish, mean, impatient customers: New Thinking: Gerry McGovern”, [s.d.]), os clientes de sítios onde as requisições demorem mais do que 4 segundos tendem a procurar serviços concorrentes e nunca mais voltam. Desta forma, uma boa cláusula para um ANS seria a garantia, por parte do provedor de serviço, de tempos de resposta **abaixo** deste tempo-limite.

Neste trabalho, são apresentadas ferramentas que conseguem garantir ANS em até 100% das vezes mantendo uma utilização média dos servidores acima de 75%. Em contraste, a utilização média dos servidores de datacenters comerciais varia entre 5% e 20% (Siegele, 2008).

Neste capítulo foi apresentado, ainda, brevemente o conceito de Teoria de Controle com Retroalimentação e, em mais detalhes, as Máquinas de Inferência Nebulosas. Esta técnica foi a escolhida nesta tese por 2 razões: (i) o relacionamento entre as variáveis de entrada não é nítido, sendo difícil encontrar uma expressão matemática que capturasse este relacionamento, enquanto as MIN têm como característica principal tomar decisões com dados imprecisos; (ii) as Variáveis Linguísticas, suas Regiões Nebulosas e as Regras que definem as tomadas de decisão são escritas em texto, permitindo ajuste e intervenção humana, caso necessário.

O trabalho apresentado em (Wang; Mendel, 1992) é extremamente importante para a condução desta proposta. Nele, os autores mostram que uma máquina de Inferência Nebulosa consegue prever com uma precisão muito grande (o trabalho não mostra a acurácia, apenas gráficos visuais) o comportamento de uma série temporal caótica.

No capítulo seguinte a proposta deste trabalho é apresentada, unindo os conceitos dos últimos dois capítulos.

Capítulo 3 - Trabalhos Relacionados

Este capítulo lista os principais trabalhos relacionados ao desenvolvido nesta tese, apontando as principais diferenças e contribuições de cada um para a criação do PROFUSE. É importante ressaltar que diversos trabalhos relacionados são recentes, indicando que este é um problema em aberto para a comunidade científica.

3.1. Elasticidade e QoS

Em (Boloor *et al.*, 2010), os autores consideram um cenário de *datacenters* geograficamente distribuídos, formando um sistema de computação em nuvem coletivo, capaz de hospedar múltiplos sistemas baseados em Arquitetura Orientada a Serviços, cada sistema possuindo um Acordo de Nível de Serviço a cumprir. O ANS de cada aplicação requer que um percentil das requisições possua seu tempo de resposta abaixo de um valor especificado, formando um conceito semelhante ao dos *deadlines* utilizados nesta tese. Se o provedor de nuvem não for capaz de cumprir este ANS, este ficará impedido de cobrar pelo uso de suas máquinas.

O trabalho apresenta um algoritmo de ranqueamento dinâmico das requisições como forma de identificar aquelas que devem gerar o maior lucro para o provedor de nuvem; uma técnica de alocação de requisições com escalonamento gi-FIFO, capaz de aumentar globalmente a receita cobrada pelo provedor do serviço de nuvem. O algoritmo de escalonamento utilizado funciona da seguinte maneira: na fila de espera, escolhe-se as requisições da classe de usuário que vão gerar a maior receita para o provedor. Destas, verifica-se quais ainda não ultrapassaram o deadline enquanto aguardavam a execução na fila. Caso todas as requisições tenham ultrapassado o deadline, executa-se a requisição que possui o maior tempo de espera. Repare que neste passo é necessário que, para garantir que uma tarefa seja executada antes de seu *deadline*, é necessário que sua duração seja conhecida previamente, uma premissa assumida neste trabalho, representando uma informação que nem sempre é possível de se obter. Por fim, é apresentado o algoritmo de alocação das tarefas às nuvens distribuídas. Como esta tarefa de alocação é reconhecidamente um problema NP-difícil, o algoritmo se baseia em heurísticas. Os experimentos comprovam a eficiência do

método proposto, principalmente quando comparado com algoritmos de escalonamento de tarefas estáticos, como FIFO ou *Weighted Round-Robin.*

Comparando com esta tese, (Boloor *et al.*, 2010) possui objetivos diferentes dos representados aqui, apesar de ambos os trabalhos terem como objetivo a redução do número de requisições que ultrapassam um *deadline*. Contudo, aqui são consideradas somente nuvens IaaS não-distrbuídas, cujas durações das requisições são desconhecidas. Ainda além, diversos cenários analisados nesta tese não são previstos pelo referido trabalho, como os ruídos e a não-linearidade do custo-benefício das unidades computacionais mínimas. A nossa preocupação é garantir que o dono do serviço/sistema hospedado em uma nuvem IaaS consiga manter a QoS como forma de evitar a fuga de clientes, ao mesmo tempo em que se evita o superdimensionamento de servidores e o custo excessivo com a alocação de servidores.

Em (Ghanbari *et al.*, 2012) é apresentado uma nova abordagem para gerenciar a quantidade de servidores utilizados por um sistema, onde dinâmicas tanto da nuvem IaaS como da aplicação nela hospedada são modeladas como um problema de controle estocástico preditivo. O método explora o impasse entre satisfazer objetivos de desempenho enquanto o custo com os servidores é minimizado, apresentando resultados obtidos em uma simulação.

Naquele trabalho é considerado um modelo de aluguel de instâncias com 4 categorias distintas (reserva imediata, reserva antecipada, melhor esforço e baseada em leilão). A seguir, uma função de custo, levando em conta cada uma das categorias, é definida. Os autores afirmam que utilizar o algoritmo de otimização guloso não seria suficiente para minimizar o custo com aluguel de instâncias por este considerar somente o estado atual do sistema. De forma semelhante, os autores afirmam que um algoritmo reativo seria ineficaz, já que este agiria somente quando o sistema já estivesse saturado e, por isso, não atenderia aos requisitos de QoS estabelecidos no ANS.

Os autores propõem a utilização de um algoritmo de otimização, a ser invocado periodicamente e baseado em *convex optimization solver*, para a função de custo levando em consideração o comportamento futuro do sistema, onde a possibilidade de reserva antecipada de instâncias é explorada como forma de minimizar o custo com aluguel de servidores. Ainda, o método proposto no trabalho mantém as instâncias durante o período inteiro de sua locação, sem devolvê-las antes do tempo. Os resultados dos experimentos mostram que a técnica é eficaz, conseguindo reservar as instâncias antecipadamente e garantindo custos reduzidos.

Contudo, o trabalho não aborda a questão da presença de ruídos na carga de trabalho, tampouco realiza um estudo para averiguar qual o impacto da latência de medição no resultado final. Ainda em contraste com a proposta apresentada nesta tese, o método dos autores se baseia em análise estocástica enquanto utilizamos TCR, através da Lógica Nebulosa, para determinar quantos núcleos de processamento devem ser adquiridos ou liberados.

O trabalho (Rajavel; Mala, 2012) discute sobre a importância da política de escalonamento de trabalhos (*jobs*) em sistemas hospedados em nuvens e propõe um novo método, hierarquizado, para realizar realizar tal tarefa. O mesmo artigo propõe, ainda, a priorização de requisições que tenham ANS vinculados a si, como forma de garantir que os acordos não serão quebrados. O trabalho propõe uma organização hierárquica das instâncias, onde há vários controladores de *clusters* (componentes que gerenciam recursos locais) sendo coordenados por um controlador de nuvem (componente que gerencia os controladores de *clusters.* Cada controlador de *cluster* possui um conjunto de nós (instâncias) e, por fim, cada instância possui o seu próprio controlador. O algoritmo proposto neste trabalho funciona da seguinte maneira: todas as novas requisições são enviadas ao controlador de nuvem que, utilizando algumas informações sobre a requisição e o estado de cada *clusters*, decide para qual controlador de *cluster* a requisição deve ser enviada. Neste passo, o controlador de *cluster* recebe a requisição e avalia a qual classe pertence a requisição (prioritária ou não), enviando para uma instância responsável por processar requisições daquele tipo. Os experimentos comparam o desempenho do método hierárquico proposto com os escalonamentos FIFO e *Shortest Job First* (SJF), se mostrando mais eficiente em todos os resultados e sendo capaz de cumprir os tempos de resposta estabelecidos nos ANS.

Apesar de o trabalho relacionado em questão ter como objetivo a manutenção da QoS de sistemas hospedados em nuvens, seu cenário estudado pode ser considerado demasiado simples. Não há nenhuma preocupação com redução de custos na alocação de servidores, em identificar ruídos nas cargas de trabalho. Nossa solução se baseia em Teoria de Controle com Retroalimentação para identificar a necessidade de alterar a configuração do *array* de instâncias, enquanto aquele trabalho tem como objetivo somente alocar as tarefas nas instâncias, um problema de maior relevância em sistemas paralelos ou grades computacionais.

Em (Dutta *et al.*, 2012) é apresentada uma ferramenta da IBM chamada SmartScale,

que utiliza uma combinação de expansão horizontal e vertical para garantir que a aplicação está configurada de forma a otimizar tanto a utilização de recursos como o custo de reconfiguração, mantendo os ANS. A metodologia apresentada pelos autores é proativa e converge rapidamente para o nível de escalabilidade mesmo quando a intensidade da carga de trabalho se altera drasticamente. A ferramenta apresentada utiliza uma Árvore de Decisão (AD) para determinar o tipo de alteração a ser feita nas instâncias do *array*, uma técnica similar à utilização da MIN. Em cada nó da AD pode-se escolher entre diversas configurações de expansão vertical e horizontal.

O algoritmo de expansão proposto e utilizado no SmartScale é executado periodicamente e funciona em duas fases: na primeira, é assumido que a demanda pelos recursos é extremamente alta e cada máquina virtual pode operar em sua utilização ótima. Desta forma, dado um conjunto de servidores nos quais pode-se executar as máquinas virtuais, calcula-se o tamanho ótimo da máquina virtual e a sua taxa de vazão em cada servidor. Na segunda fase, determina-se o número mínimo de instâncias que satisfaçam a taxa de vazão calculada na fase 1, utilizando uma busca binária para encontrar a solução ótima. Repare que neste trabalho o termo instância refere-se a máquinas virtuais e cada servidor pode hospedar mais de uma instância.

Na descrição de sua arquitetura, é dito que o SmartScale utiliza um “mix de cargas de trabalho previamente carregadas nele, que foram identificadas utilizando o algoritmo de clusterização *k-means*”. A partir desta informação, os autores utilizam um classificador para identificar a qual carga de trabalho o SmartScale está sendo submetido. Apesar de não ser claro neste ponto, o entendimento é que, para cada possível carga de trabalho há uma AD diferente implementada na ferramenta. Em nossa ferramenta, utilizamos uma abordagem semelhante, criando uma MIN diferente para uma CdT específica.

Os experimentos foram conduzidos utilizando uma carga de trabalho sintética, um *benchmark* para sistemas em nuvens que representa uma rede social, chamado Olio. Os gráficos apresentados nos resultados são confusos (é impossível distinguir entre dois métodos sendo comparados, pois foi utilizada a mesma formatação para suas linhas nos gráficos), contudo a descrição dos resultados atesta a eficiência do SmartScale, mostrando-se mais eficiente na tarefa de cumprir os ANS (no artigo, quebras de ANS implicam em penalizações financeiras) e reduzir os custos.

Em comparação com esta tese, o trabalho não apresentou preocupações sobre a existência de ruídos, ou em estudar os impactos utilizando diferentes intervalos de atuação da ferramenta. Ainda, os autores propõem um algoritmo que compreende expansão vertical e horizontal, enquanto neste trabalho é estudado somente o caso com expansão horizontal. A expansão vertical, contudo, é ainda desaconselhada por alguns autores (Ardagna *et al.*, 2012) e (Armbrust *et al.*, 2010) que advertem sobre a temporária queda de desempenho de aumentar, em tempo de execução, os recursos de uma máquina virtual.

3.2. Execução de Workflows Científicos em Nuvens

A tese (Oliveira, 2012) apresenta uma abordagem para gerência da execução paralela de workflows científicos em ambientes de nuvem de forma adaptativa chamada SciCumulus. O SciCumulus verifica a capacidade computacional disponível, ajusta dinamicamente a distribuição das tarefas e dimensiona o ambiente de nuvem para alcançar um melhor desempenho. O SciCumulus cria automaticamente máquinas virtuais e *clusters* virtuais com base nas restrições impostas pelos cientistas. Além disso, o SciCumulus gera os dados de proveniência em tempo de execução, ou seja, registra informações sobre o *workflow* que está sendo executado enquanto o mesmo se encontra em execução.

Através de uma função de custo elaborada considerando confiabilidade, custo financeiro e tempo de execução, o SciCumulus realiza uma etapa de otimização, utilizando um algoritmo guloso, capaz de encontrar um plano de execução onde são reduzidos o tempo de execução, o custo monetário e as chances de falha dos experimentos. Ainda, é possível encontrar um escalonamento balanceado, onde todas as variáveis são levadas em consideração. Os experimentos mostraram os benefícios do SciCumulus, que apresentou um aumento de desempenho de até 37,9% frente a abordagens tradicionais de paralelismo em nuvens.

A dissertação (Viana, 2012) apresenta uma extensão para o SciCumulus e propõe o SciCumulus-ECM (SciCumulus *Environment Cost Model*), um serviço baseado em um modelo de custo, que determina a melhor configuração possível para a execução de *Workflows* Científicos de acordo com restrições impostas pelos cientistas, através de um otimizador baseado em algoritmo genético. Nos experimentos, é possível perceber que o

SciCumulus-ECM encontra uma solução robusta para a execução do experimento com tempo linear, independente do número de instâncias.

Ainda no tema de execução de *workflows* científicos, a tese (Manuel Serra da Cruz, 2011) apresenta uma estratégia de apoio à gerência dos descritores de proveniência, baseada no desenvolvimento de uma solução computacional que envolve a integração de uma ontologia de proveniência intitulada OvO (*Open proVenance Ontology*), e de um sistema de coleta de descritores de proveniência intitulado *Matriohska*. Este sistema é capaz de ser acoplado aos *workflows* científicos executados em ambientes distribuídos e heterogêneos do tipo nuvens de computadores. A estratégia apresentada também possibilita a execução de consultas sobre os descritores de proveniência de variadas granulosidades obtidos a partir de experimentos executados nesses ambientes.

Comparando os trabalhos citados acima com o desenvolvido nesta tese, pode-se dizer que os cenários estudados são diferentes e possuem objetivos distintos. Apesar de inúmeras semelhanças, como redução de custos, tempo de execução e busca pela configuração ótima de instâncias para executar uma demanda, um sistema de comércio eletrônico possui preocupações outras das presentes em processamento de experimentos científicos. Como exemplo, pode-se citar (i) a preocupação com a manutenção da QoS para evitar o abandono de clientes e a perda financeira consequente; (ii) a eliminação da restrição com o custo: é possível que, ao final do dia, o dono do sistema hospedado em nuvem possua um balanço *negativo*, indicando que teve prejuízo; e (iii) a necessidade de averiguar periodicamente mudanças na carga de trabalho e constantemente a ocorrência de ruídos.

3.3. Aumento da Eficiência Energética

O trabalho (Dyachuk; Mazzucco, 2010) aborda o problema de maximizar a receita de um provedor de nuvem utilizando políticas de alocação que executem o menor número de servidores necessários, satisfazendo as necessidades dos usuários em termos de desempenho. O objetivo do artigo era, portanto, aumentar o número de clientes em cada servidor sem comprometer a QoS final, de modo que fosse possível desligar alguns servidores. Os autores propõem duas abordagens: uma adaptativa, que encontra a solução ótima através de uma busca binária entre as possíveis configurações, e um baseada em heurística, chamado de QEP,

que encontra uma solução sub-ótima de forma mais rápida. Os experimentos realizados através de simulação confirmam a eficiência de ambas as soluções (ótima e sub-ótima), alavancando o lucro final do provedor de nuvem. Ao aumentar a utilização nos servidores, é possível deixar outros desligados, reduzindo o consumo de energia total do *datacenter.*

O objetivo de (Mazzucco; Dyachuk; Dikaiakos, 2011) é o mesmo do anterior: reduzir a quantidade de servidores sendo utilizados no *datacenter*, sendo capaz, assim, de reduzir o consumo de energia necessária para manter os computadores ligados, resfriamento, etc.. Os autores elaboram um modelo probabilístico para determinar quantos servidores são necessários para atender às demandas dos clientes. Em seguida, criam um modelo para estimar a quantidade de energia elétrica necessária pela configuração determinada anteriormente, criando uma função de custo unimodal, com apenas um máximo. Por fim, a técnica proposta reavalia periodicamente a configuração de servidores utilizada, calculando, através de uma busca binária, o máximo da função de custo.

Os experimentos apresentados foram obtidos através de simulação e evidenciam a redução no consumo de energia dos *datacenters*, sendo possível manter servidores desligados sem comprometer a QoS oferecida.

Em comparação com o nosso trabalho, podemos dizer que possuem objetivos similares para personagens diferentes: enquanto os autores daqueles trabalhos possuem como objetivo principal a maximização do lucro dos provedores de nuvem, elaborando técnicas que levam em consideração a quantidade de energia elétrica gasta pelo *datacenter*, esta tese se preocupa em alavancar o lucro de provedores dos sistemas hospedados em nuvens. Contudo, em ambos os objetivos há a preocupação de aumentar a utilização dos servidores/instâncias como forma de reduzir os custos, sem haver redução na QoS.

Sobre a implementação das técnicas, enquanto os autores dos referidos artigos propuseram modelos probabilísticos para representar os relacionamentos entre as diversas componentes do sistema e realizar os cálculos, nesta tese foi utilizada uma MIN para realizar tal função.

3.4. Controle de Admissão

Os nossos trabalhos anteriores em (Orleans, L.; Zimbrão; Furtado, 2008) e (Orleans,

L. F.; Furtado, 2007) estudam como manter a QoS de sistemas hospedados em ambientes puramente paralelos, com o número de servidores fixo. Nestes artigos, as tarefas possuem *deadlines* e, como forma de manter o estado do sistema sob controle, era utilizado uma etapa de controle de admissão. Naqueles cenários, era preferível a recusa de uma requisição ao seu aceite sem a garantia de que esta seria processada antes do *deadline*. De um modo geral, o módulo de controle de admissão estimava o tempo que a tarefa levaria para ser processada, levando em consideração o tamanho da fila. Caso fosse constatada a viabilidade, a tarefa era admitida no sistema. Caso contrário, era rejeitada.

O artigo (Schroeder *et al.*, 2006) apresenta um estudo sobre o impacto de fixar o número de requisições concorrentes em um servidor. Quando este número, chamado de *MultiProgramming Level* (MPL), é atingido, as requisições seguintes são enviadas para uma fila FCFS. Os resultados mostram que com um número baixo para o MPL é possível atingir uma taxa de vazão similar a de sistemas sem nenhum controle de admissão.

Em comparação com esta tese, nenhum dos trabalhos referenciados até aqui nesta seção tem como objetivo reduzir o custo com alocação de servidores, uma vez que os cenários se referem a computação paralela, mais aplicada a Grades Computacionais. A manutenção da QoS nestes trabalhos se dá através da recusa de novas tarefas, uma abordagem desnecessária em Computação em Nuvem, que possui o conceito de Elasticidade.

O trabalho (Konstanteli *et al.*, 2012) apresenta um teste de controle de admissão, decidindo se é viável ou não admitir um conjunto de serviços novos em uma nuvem e, no caso de aceite, obter a alocação ótima para cada um das componentes dos serviços. No modelo proposto, o foco é criar uma abordagem de elasticidade que seja capaz de lidar com mudanças nas necessidades de recursos dos clientes de forma dinâmica, dependendo das variações do número de usuários e padrões de requisições. Para achar a alocação ótima, o teste de controle de admissão apresentado utiliza um modelo de otimização que incorpora regras de negócio, como requisitos de QoS, eco-eficiência e custo. Os autores formalizam o problema como um Problema de Programação Linear e utilizam um *software General Algebraic Modeling System* (GAMS) para modelá-lo e outro *software*, *Branch and Reduce Optimization Problem* (BARON) para resolvê-lo.

O referido trabalho estuda a viabilidade de aceitar ou não conjuntos de serviços, enquanto nesta tese estudamos como manter a QoS para cada *requisição* através da

Elasticidade. Nossa preocupação é em criar um Modelo de Elasticidade que seja capaz de manter a QoS e reduzir os custos com aluguel de servidores evitando, assim, a rejeição de tarefas, um cenário diferente do estudado no artigo.

A tabela 3 exibe a comparação das funcionalidades entre os principais trabalhos relacionados.

Tabela 3: Comparação das funcionalidades entre os trabalhos relacionados

	PROFUSE	**SmartScale**	**SciCumulus**	**(Ghanbari et al., 2012)**	**(Rajavel; Mala, 2012)**	**(Boloor et al, 2010)**
Garante *deadlines*?	X	X	X	X	X	X
Balanceia Custo e QoS?	X	X	X	X		X
Configurável?	X		X			
Trata ruídos?	X					
Utiliza Elasticidade?	X	X	X	X		X
Sistemas Comerciais?	X	X		X	X	X
Prevê alterações na CdT?	X	X		X		
Diferencia Tarefas?					X	X
Nuvens IaaS Distribuídas?	X	X	X	X		X

Capítulo 4 - PROFUSE – Alocação Proativa De Servidores Através Do Uso De Máquinas De Inferência Nebulosa

Em paradigmas anteriores à CN, um sistema paralelo/distribuído sobrecarregado seria obrigado a rejeitar o processamento de novas requisições como forma de evitar a quebra excessiva de contratos de ANS (Orleans, L.; Zimbrão; Furtado, 2008), (Orleans, L. F.; Furtado, 2007), (Orleans, L.; Oliveira, de; Furtado, 2007), (Schroeder *et al.*, 2006). Em alguns cenários estudados nesses trabalhos, era preferível a recusa de novas requisições ao aceite de uma nova tarefa que não seria executada dentro do tempo máximo estipulado, sendo a causa principal de tal restrição o número fixo de servidores.

Em sistemas instalados em *clusters* tradicionais, a configuração de *hardware* (número total de CPUs, memória total disponível, etc.), após seu início, não era mais alterada. Entretanto, a CN trouxe consigo o conceito de Elasticidade, permitindo que novos recursos sejam adicionados em tempo de execução. Dessa forma, a rejeição de uma tarefa poder ser impedida, tendo em vista que, ao notar que o sistema está se tornando saturado, novos recursos podem ser adicionados e disponibilizados em questão de minutos.

4.1. Visão Geral

Após os capítulos anteriores, é possível notar que a alocação de servidores em tempo de execução, de modo a manter a taxa de quebra de contratos a mais baixa possível e, ao mesmo tempo, manter a utilização dos servidores em seu patamar mais alto possível, é um problema que pode ser resolvido através da Teoria de Controle. A utilização da retroalimentação torna possível ajustar a configuração do conjunto de instâncias, contribuindo para reduzir o número de perdas de *deadlines* e os consequentes abandonos de clientes. Ainda além, como a aquisição de novas instâncias pode demorar alguns minutos, é recomendável que a técnica utilizada seja capaz de prever alterações na carga de trabalho e que possa sinalizar a necessidade de aumentar o conjunto de instâncias previamente. Dessa forma, no momento em que a carga de trabalho aumentar, o sistema já estará configurado de acordo e conseguirá garantir a qualidade do serviço prestado aos clientes.

Nessa linha, optou-se pela utilização de uma Máquina de Inferência Nebulosa nesta tese principalmente pela possibilidade de ajuste fino por parte de um especialista através da

recalibração das regras, das variáveis linguísticas e dos limites de cada região nebulosa. Nesta tese, considera-se que a descrição da MIN, isto é, o seu conjunto de VLs, as respectivas regiões nebulosas e as regras, pode ser alterada a qualquer momento como forma de refletir alterações na CdT e/ou na QoS desejada, sem a necessidade de recompilação de código-fonte ou etapas adicionais de aprendizado.

Criamos uma técnica, chamada PROFUSE (do inglês *Proactive Fuzzy Logic-based Servers Allocation*), que dadas algumas informações sobre o ANS e a CdT histórica é capaz de fazer ajustes no número de servidores de forma tanto reativa (para consertar diferenças entre a CdT informada e a real) quanto proativa (utilizando a CdT histórica como base). O PROFUSE constitui um Modelo de Elasticidade completo, uma vez que possui módulos independentes que (i) calculam, utilizando uma MIN, a diferença de poder computacional para que a qualidade de serviço e a alta utilização do sistema sejam preservadas; e (ii) realizam, através de um conjunto de algoritmos e métodos criados, a comunicação com o provedor IaaS, sendo, ainda, responsável pela alocação/liberação de instâncias, seguindo a premissa de diminuição dos custos relacionados ao aluguel de servidores.

4.2. Premissas

O perfeito funcionamento do PROFUSE se baseia em algumas premissas básicas:

1. *Tarefas independentes*: é assumido que toda e qualquer requisição tratada pelo PROFUSE é independente, o que implica que esta pode ser associada a qualquer servidor e que as requisições podem ser processadas em qualquer ordem;

2. *Orçamento para alocação de instâncias*: como esta tese trata de sistemas de comércio eletrônico, *blogs* ou qualquer sistema que vise ao lucro, o orçamento para alocação de servidores junto ao provedor IaaS não é limitado. Isso significa dizer que, ao longo de um dia, o provedor do sistema pode possuir um balanço negativo.

4.3. Funcionamento do PROFUSE

O PROFUSE possui dois módulos principais:

1. *Módulo Observador de Carga de Trabalho (WOM):* este módulo é responsável por analisar periodicamente a carga de trabalho e, com o auxílio da MIN, calcular o total de poder computacional necessário para que as tarefas sejam atendidas dentro do tempo máximo. Este módulo é composto por duas componentes:

   a) *Componente Gerador*: responsável por tomar como entrada o *log* de um sistema, que representa o comportamento histórico da CdT, e, a partir dele, gerar o Descritor da Máquina de Inferência Nebulosa (MIN), um arquivo contendo as descrições das variáveis linguísticas, suas regiões nebulosas iniciais e as regras utilizadas pela MIN. Esse arquivo serve como entrada para o próximo módulo. Sua arquitetura pode ser vista na figura 14.

   b) *Componente de Execução*: utiliza a saída do Módulo Gerador como entrada e cria,

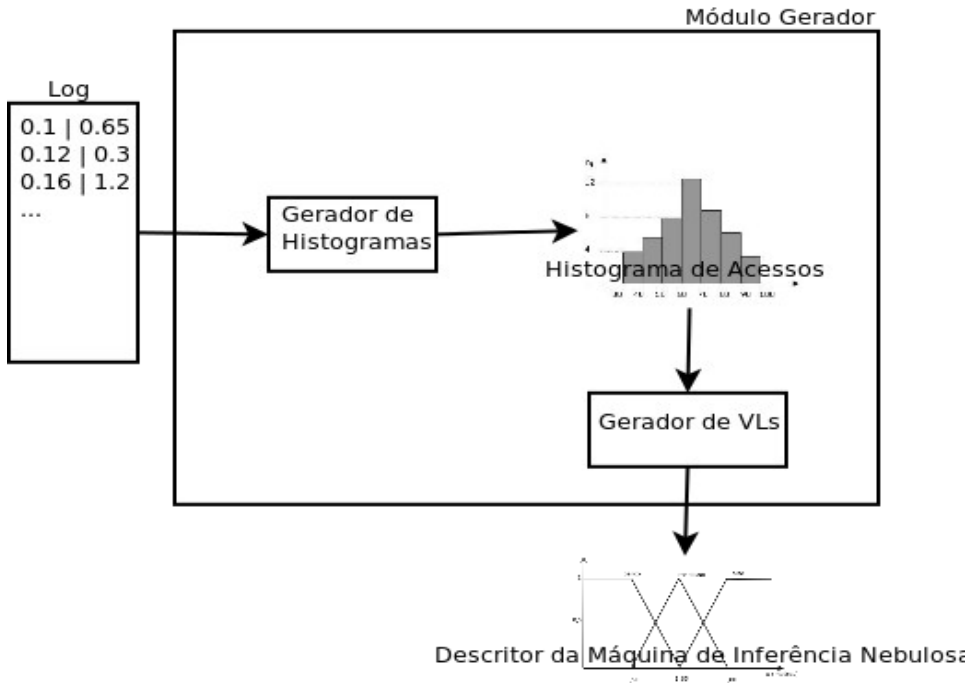


Figura 14: Arquitetura do Componente Gerador, parte do Módulo Observador de Carga de Trabalho.

de fato, a MIN. Sua arquitetura pode ser vista na figura 16. Seu funcionamento consiste em monitorar a carga de trabalho a qual o sistema está submetido, extraindo os valores atuais de cada VL. Logo após insere-as na MIN e repassa a sua saída para o próximo módulo.

2. *Módulo de Aquisição de Servidores (SAM):* este módulo analisa a saída da MIN e realiza as aquisições/liberações de instâncias. Tais operações são efetuadas de forma a garantir que a configuração do conjunto de servidores somente será alterada se for mais lucrativo para o Provedor do Serviço.

Os módulos WOM e SAM serão descritos com mais detalhes nas próximas seções.

4.4. Módulo Observador de Carga de Trabalho

O WOM (do inglês *Workload Observer Module*) é o responsável por analisar a carga de trabalho e detectar variações significativas, que impactem diretamente no lucro final do provedor do sistema. O WOM opera em 4 passos distintos, descritos a seguir.

Passo 1: Criação do Histograma de Acessos

Primeiramente, o WOM analisa um *log* de acessos típico do sistema, buscando pelas seguintes informações: hora do dia (em segundos) em que a requisição chegou e seu tempo de duração (também em segundos). Assim, o WOM é capaz de criar um histograma contendo o

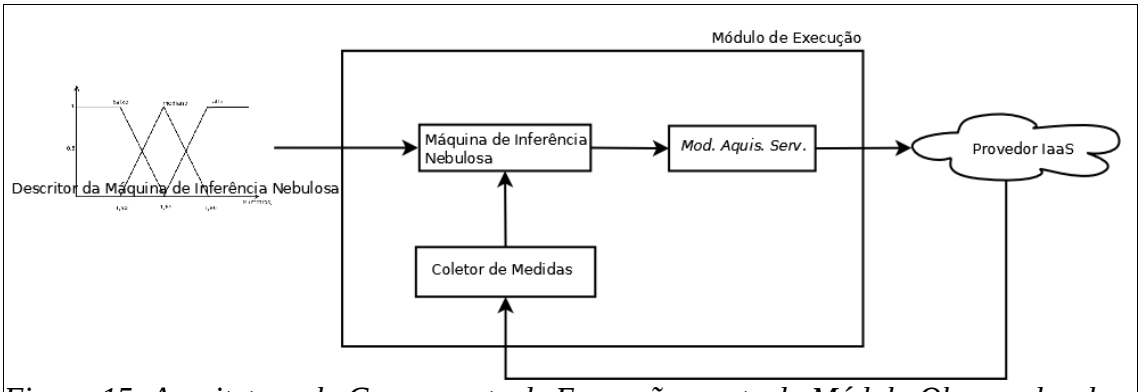


Figura 15: Arquitetura do Componente de Execução, parte do Módulo Observador de Carga de Trabalho.

total de requisições recebidas pelo sistema em cada hora do dia. Repare que o intervalo de tempo representado neste histograma pode ser variável, refletindo o comportamento do sistema ao longo de horas, dias ou mesmo de períodos maiores, como semanas e meses. Igualmente, o histograma não precisa mostrar o total de requisições por *hora*, podendo ser utilizadas outras unidades de tempo. Contudo, neste trabalho as requisições foram agrupadas em horas, pois esta é a unidade de medição para cobrança das instâncias alugadas junto ao provedor IaaS. A tabela 4 exibe um exemplo de *log* que o WOM utiliza como entrada. O histograma gerado (aqui denominado de *histograma de acessos*) possui o formato apresentado na tabela 5. Repare que existe no histograma um campo adicional, chamado de Diferença de Carga de Trabalho (DcdT), que possui um campo com valor calculado no qual é armazenada a diferença das cargas de trabalho entre a hora seguinte e a corrente. Ou seja,

$$DCdT_h = CdT_{h+1} - CdT_h \quad (7)$$

onde h é a hora corrente e $h+1$, a hora seguinte. Esse campo é o principal responsável pelo comportamento proativo do PROFUSE, uma vez que torna possível saber previamente como a carga de trabalho do sistema varia ao longo das horas. Dessa forma, se é previsto que a taxa de chegada de requisições vai aumentar em um curto espaço de tempo, é possível alocar novas instâncias *antes* que o sistema fique saturado.

Tabela 4: Exemplo de *Log* de Acessos

Tempo de chegada da requisição (s)	**Duração da requisição (s)**
0.12	3.1
0.15	0.87

Tabela 5: Exemplo de Histograma de Acessos

Hora do Dia	**Total de Acessos**	**Diferença de CdT**
h_0	100	-50
h_1	50	150
h_2	200	...

Passo 2: Geração das Variáveis Linguísticas e Suas Regiões Nebulosas

Em seguida, o WOM analisa o histograma de acessos gerado e cria um arquivo de descrição das variáveis linguísticas e suas regiões nebulosas. Neste trabalho, assumimos que a MIN é definida em um arquivo-texto, entendível por humanos. Por ser escrito em formato de texto, é possível editá-lo para refinar os valores das regiões nebulosas, assim como alterar o conjunto de regras. O arquivo inicial utilizado pelo PROFUSE considera as seguintes variáveis linguísticas e as suas respectivas regiões nebulosas:

1. *Diferença de CdT (DCdT):* campo calculado, indicando a diferença entre a CdT da hora atual e da próxima ($CdT_{próxima} - CdT_{atual}$). As regiões desta VL foram calculadas da seguinte forma: analisa-se o Histograma de Acessos e identifica-se a menor e a maior variação das CdTs – estes serão os valores mínimo e máximo desta VL. Na área compreendida pela diferença entre tais valores são criadas 5 regiões nebulosas, a saber: (i) grandeNegativo, identificando uma queda brusca de requisições na CdT; (ii) negativo, indicando uma queda suave; (iii) neutro, que aponta uma manutenção da CdT; (iv) positivo, indicando um aumento suave na CdT; e (v) grandePositivo, quando há um aumento brusco na CdT.

2. *Tempo para próxima categoria (TPPC):* esta VL indica quanto tempo falta até chegar a próxima hora e, consequentemente, a próxima variação na CdT, possibilitando, assim, que o PROFUSE altere a configuração dos recursos de acordo com a mudança da CdT. Esta VL possui duas regiões: (i) curto, indicando que a próxima hora se encontra próxima; e (ii) longo, apontando que a próxima hora não se encontra próxima. Na prática, esta VL é uma das responsáveis pela proatividade do PROFUSE.

3. *Percentual de perda de deadlines (PPD)*: indica o percentual de quebra de *deadlines* (e de abandonos de clientes) desde a última verificação. A utilização da TCR leva o WOM a fazer verificações periódicas e, cada vez que realiza essa checagem, esse percentual é zerado. Esta VL se divide em três regiões: (i) baixa, indicando uma taxa de perdas de *deadline* tolerável; (ii) alta, indicando que a taxa está no limite; e (iii)

muito alta, indicando que o PROFUSE deve tomar alguma medida corretiva, como alocar novos servidores imediatamente. Repare que esta VL representa, igualmente, a *probabilidade* de o processamento de uma requisição ultrapassar o tempo máximo permitido.

4. *Tamanho da fila de requisições (TFR)*: esta VL tem como responsabilidade fornecer mecanismos para que o PROFUSE evite um aumento de abandonos no futuro próximo. Ela se divide em quatro regiões: (i) pequena, indicando que o PROFUSE pode, dependendo das outras VL, reduzir a quantidade de recursos; (ii) normal, quando o PROFUSE não necessita tomar nenhuma ação; (iii) grande, indicando que o PROFUSE deve avaliar outras métricas para decidir sobre a alocação de novos recursos; e (iv) imensa, indicando que o PROFUSE deve alocar novos recursos imediatamente, independentemente das outras métricas.

5. *Utilização do sistema (US)*: esta VL serve basicamente como um parâmetro consultivo para o PROFUSE decidir pela (des)alocação de recursos, sendo normalmente analisado em conjunto com a VL TFR. Ela se divide em cinco regiões: (i) ocioso, indicando que o sistema está abaixo da sua capacidade, hipótese em que o PROFUSE pode, então, dependendo das outras VL, diminuir a quantidade de recursos do sistema; (ii) normal, indicando que o sistema está dentro da normalidade, não sendo necessária nenhuma ação imediata do PROFUSE; (iii) ocupado, quando o sistema está operando em seu limite; (iv) muitoOcupado, mostrando que o sistema está atuando acima de sua capacidade e, neste caso, o PROFUSE decide pela alocação de novos (poucos) recursos; e (v) proibitivo, quando necessita de uma intervenção imediata do PROFUSE através do aumento da quantidade de servidores.

6. *Número de Núcleos (NN)*: esta é a VL de saída, que indica a ação que deve ser tomada. Possui quatro regiões: (i) encolher, representando que o PROFUSE verificou que o sistema estava abaixo de sua capacidade e decidiu pela redução do número de servidores, como forma de aumentar a utilização; (ii) manter, representando que o PROFUSE verificou que o sistema apresenta taxa de quebra de contratos e utilização

aceitáveis e, por isso, decidiu pela manutenção do número de servidores; (iii) expansão, indicando que o PROFUSE detectou que o sistema está ou estará em uma situação de leve desajuste, com um aumento na taxa de quebra de contratos, e, por isso, decidiu por um aumento baixo do número de servidores; e (iv) expansão rápida, mostrando que o PROFUSE detectou alguma situação de grande desajuste, presente ou futura, que certamente acarretará em quebra de *deadlines* em grande número e, por isso, decidiu por um aumento expressivo do número de servidores. Repare que a saída da MIN é expressa pelo número de núcleos, que é a unidade computacional mínima. Nesse ponto, ainda não se decidiu pela configuração das instâncias que serão alocadas, sendo essa decisão uma responsabilidade do Módulo de Aquisição de Servidores.

A seção 4.6. descreve como chegamos a estas variáveis linguísticas, como suas regiões nebulosas foram definidas e como o conjunto inicial de regras foi determinado.

Passo 3: Refino do Descritor da Máquina de Inferência Nebulosa

O arquivo gerado na etapa anterior possui as definições iniciais da VL de entrada e da VL de saída, com suas respectivas regiões nebulosas e o conjunto inicial de regras. Uma das vantagens da utilização da Lógica Nebulosa como meio de automatizar a aquisição de novas instâncias em nuvens IaaS é a possibilidade de refino, por parte de um especialista, das componentes da MIN, uma vez que seu descritor é um arquivo texto. Assim, tanto as regiões nebulosas como o conjunto de regras podem ser adaptados para refletir os ANS vigentes e evitar a sobrecarga do sistema e abandono de cliente, com a consequente perda financeira.

Passo 4: Execução

Finalmente, o WOM inicia um *daemon*, ou seja, um processo que é executado separadamente do sistema principal. Esta componente, chamada de Coletor de Medidas, periodicamente analisa o estado do sistema e a carga de trabalho corrente, extrai os valores correspondentes das Vls, e os insere na MIN. Esta, por sua vez, realiza a inferência com base nas regras descritas anteriormente e gera um valor de saída (Δ), informando ao *Módulo de Aquisição de Servidores* qual ação deve ser tomada. Por fim, este módulo se comunica com o

provedor IaaS para realizar alocações ou liberações de instâncias. A figura 16 mostra o funcionamento completo do WOM.

4.5. Módulo de Aquisição de Servidores

Este módulo é chamado de *SAM* (do inglês *Servers Acquisition Module*) e tem como principal objetivo realizar as operações junto ao provedor IaaS para alugar ou liberar servidores, de acordo com a saída Δ após a análise do WOM.

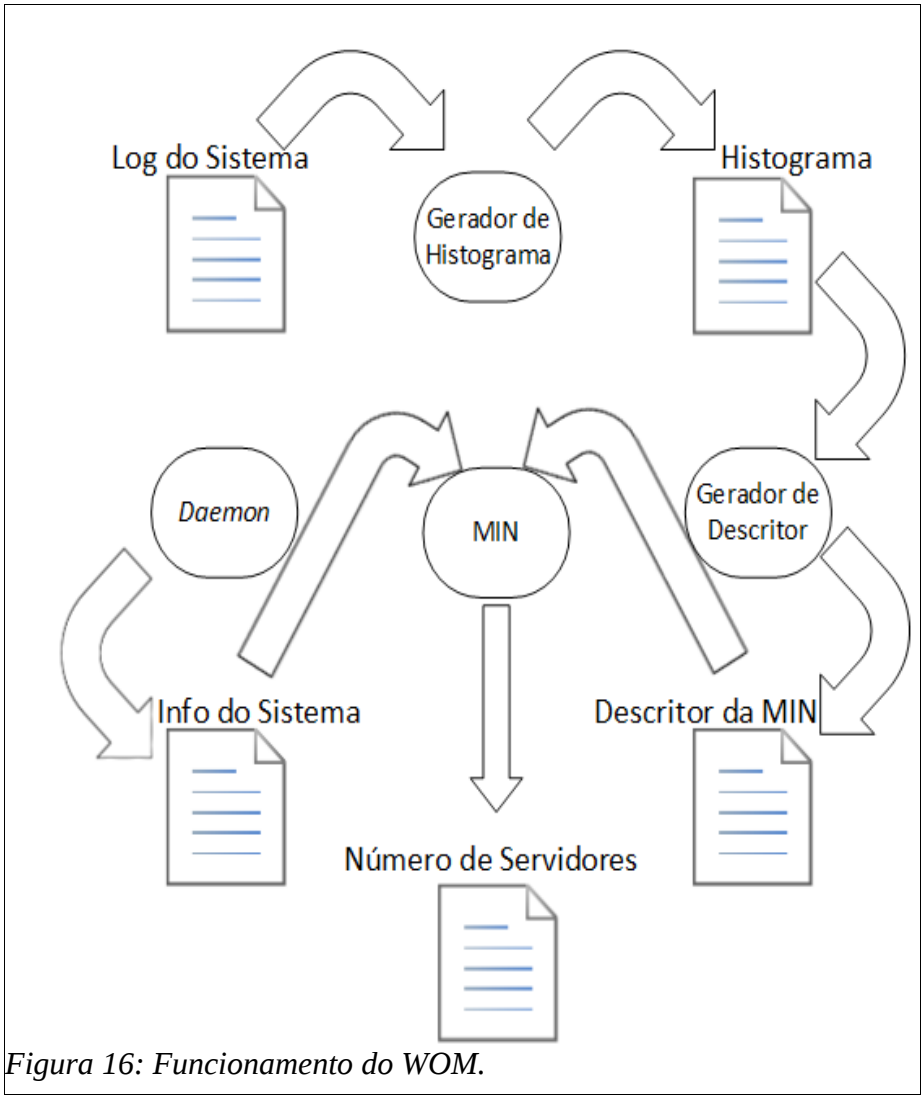


Figura 16: Funcionamento do WOM.

Heterogeneidade de Instâncias

Um provedor IaaS pode limitar o número total de instâncias que um cliente pode alugar ao mesmo tempo. No serviço Amazon EC2, por exemplo, um cliente pode ter, no máximo, 20 instâncias concorrentes, independentemente de suas configurações. Dessa forma, para continuar aumentando sua capacidade computacional, o cliente deve adquirir instâncias mais caras que, todavia, oferecem maior potência de processamento. Por exemplo, na Amazon EC2 o cliente pode escolher entre os seguintes tipos de instâncias: *pequena*, *média*, *grande e extra-grande*, onde cada uma oferece o dobro de processamento da configuração imediatamente anterior (uma instância média, por exemplo, possui o dobro da capacidade computacional de uma instância pequena). Convém notar que configurações distintas possuem preços diferentes e o relacionamento entre estas duas grandezas, ou seja, o custo-benefício, pode não ser linear. Como exemplo, considere duas instâncias de configurações diferentes: i_0 *e* i_1. O custo-benefício não é linear quando o valor de uma unidade computacional (no caso desta tese, um núcleo de processamento) é diferente em i_0 e i_1. Assim, se i_1 possuir o dobro de núcleos de i_0, o custo de i_1 é diferente do custo de duas instâncias i_0.

Premissas para Alocação de Instâncias

De uma perspectiva de aumento da receita de um serviço ou sistema hospedado em uma nuvem IaaS, é preferível adquirir uma instância mais cara e poderosa somente no momento em que a taxa de quebra de *deadlines* atinja um patamar no qual a soma das perdas financeiras associadas ultrapasse a diferença do custo entre as instâncias. Neste caso, a fim de avaliar o impacto das perdas, devemos adicionar à equação 1 o custo de executar a requisição em diferentes configurações de servidores.

Assim, considere *c(x)* como a função que determina o custo para executar uma requisição em uma instância de configuração *x*. Considere *ih* como o custo de alugar a instância de configuração *x* por uma hora, isto é, o custo da instância-hora de *x*. Considere, ainda, que a configuração *x* é capaz de processar *tr* requisições em um segundo. Dessa forma, podemos determinar *c(x)* em função de *ih* e *tr* como sendo:

$$c(x) = \frac{ih}{tr \times 3600} \tag{8}$$

A equação 8 divide o custo da instância-hora de *x* pelo número total de requisições que *x* pode processar em uma hora. Assim, podemos reescrever a equação 2, que determina o valor agregado de uma requisição, levando em consideração o custo de sua execução em um servidor de configuração *i:*

$$\bar{v} = \overline{rb} \times \overline{TCC} \times \frac{1}{\bar{n}} \times q - c(i) \tag{9}$$

A equação 9 agora calcula o valor agregado de processar uma requisição em uma instância de configuração *i* com probabilidade *q* de o cliente não abandonar o sistema.

Finalmente, de modo a determinar se é mais interessante executar a requisição na instância i_1 ou na instância i_0 (i_1 é mais cara e duas vezes mais poderosa que i_0), devemos avaliar se $\bar{v}_{i1} \geq \bar{v}_{io}$, onde *q* é a única variável. Assim, quando:

$$q \geq \frac{2 \times [c_{i1} - c_{i0}]}{(\overline{rb} \times \overline{TCC} \times (\frac{1}{\bar{n}}))} \tag{10}$$

vale a pena trocar i_0 por i_1. Note que consideramos que i_1 possui o dobro da capacidade computacional de i_0, de onde resulta a constante "2×" na equação 10.

Definindo a Estratégia para Alocação de Instâncias

A saída Δ do WOM serve como entrada para o SAM. O papel desse módulo é adquirir e/ou liberar instâncias de acordo com o valor de Δ. Note que é responsabilidade do SAM determinar quantas instâncias devem ser adquiridas e quais as suas configurações ou, no caso de diminuição da capacidade computacional, quais instâncias devem ser liberadas.

Mantendo o foco na diminuição da perda financeira do provedor do sistema, a estratégia para alocação de servidores desenvolvida neste trabalho reduz o seu custo relacionado, funcionando da seguinte maneira: considere que o provedor IaaS permita que

cada provedor de sistema alugue, no máximo, MAX instâncias. Considere, ainda, que cada instância pode possuir uma dentre *c* configurações distintas, que, por sua vez, possuem custos distintos, de maneira que o relacionamento entre capacidade computacional e custo pode ser ou não linear. Assim, cada instância do *array* pode ser definida como i_{nk}, onde *n* representa a posição da instância no conjunto ($1 \leq n \leq$ MAX) e *k*, a sua configuração ($1 \leq k \leq c$). Igualmente, o *array* inicial de instâncias do provedor de sistemas pode ser representado por

$$a = \{i_{11}, i_{21}, \ldots, i_{n1}\}$$

Como nosso objetivo é a redução do custo com a alocação de servidores e a manutenção da QoS ao mesmo tempo, o conjunto inicial de instâncias possui apenas servidores com a configuração mais barata. Para cada Δ positivo calculado pelo WOM, o SAM continua a adquirir instâncias com $k = 1$, até que o tamanho do *array* atinja o valor de MAX – 1 (número máximo de instâncias que um cliente pode alugar menos 1). Note que, se o provedor IaaS não estabelecer um limite para o número de instâncias, apenas servidores com configuração $k = 1$ serão alugados. Esta última "vaga" no conjunto de instâncias será utilizada para permitir a troca de servidores (*swapping*) e a consequente manutenção do aumento da

Método I: Expansão_PROFUSE
1. Inicie com um conjunto de instâncias de menor configuração ($k = 1$). 2. Cada aumento no *array* deve ser feito através da aquisição de novas instâncias com configuração $k = 1$, até que o número de instâncias no conjunto atinja MAX – 1. 3. A partir desse momento, utilizar a equação 10 para determinar se é economicamente interessante a aquisição de uma nova instância. Se for o caso, utilizar a última vaga no *array* para: 3.1. Iniciar uma instância com uma configuração imediatamente superior. 3.2. Liberar uma instância que possua a configuração atual utilizando o Algoritmo I. 4. Se o *array* contiver somente instâncias com configuração $k = c$, iniciar uma instância de configuração *c* na última vaga.

capacidade computacional total. O *swapping* consiste na aquisição de uma instância de configuração superior àquelas contidas no *array* e a imediata liberação de um servidor do conjunto, como forma de manter sempre uma vaga disponível.

Algoritmo I: Liberação_Instâncias_PROFUSE Entrada: tipo da instância a ser liberada (*k*) Saída: -
1. Para cada instância *i* contida no *array*: 1.1. se *i* for do tipo *k:* 1.1.1.Calcular o tempo restante *t* até que *i* aumente sua instância-hora. 2. Liberar a instância com o menor *t.*

Método II: Retração_PROFUSE
1. Se o tamanho do *array* for igual a MAX, todas as instâncias são iguais e possuem configuração *c.* Libere uma instância e mantenha o seu lugar vazio. 2. Caso contrário, determine a configuração *t* mais cara dentre os servidores contidos no *array* (maior *k*). 3. Se *k* = 0, libere uma instância. 4. Caso contrário, utilize a vaga no *array* para: 3.1. Iniciar uma instância com configuração imediatamente inferior. 3.2. .Liberar uma instância de tipo *t.*

Cabe ressaltar que a troca se torna economicamente interessante para o provedor de sistema quando a taxa de perda de *deadlines* atinge o limite estabelecido na equação 10. Sempre que uma troca é feita, uma vaga é mantida disponível para ser utilizada em uma próxima troca. Se o *array* contiver somente instâncias da configuração mais poderosa (*k* = c), este último lugar é preenchido por outra instância da configuração *k* = *c*. O Método I detalha como a aquisição de novas instâncias é feita pelo SAM.

De forma semelhante, o Algoritmo I descreve a política para liberação de instâncias. Ele leva em consideração a *oportunidade de liberação*, isto é, o servidor que estiver mais próximo de aumentar seu custo (medido em instância-hora) deverá ser devolvido ao provedor IaaS.

Ao contrário do método de expansão, a retração do *array* de instâncias deve liberar antes uma instância de configuração mais cara (sempre utilizando o Algoritmo I) e alugar um servidor que contenha uma configuração mais barata, caso seja necessário. O Método II descreve como a retração ocorre no SAM. Note que ambos os métodos (expansão e retração) atingem a configuração mais barata para evitar a quebra excessiva de contratos, uma vez que instâncias mais caras somente são alugadas quando não há mais espaço para servidores de configurações mais modestas devido ao limite do número máximo de instâncias imposto pelo provedor IaaS. Da mesma forma, a retração ocorre de forma suave, sempre trocando instâncias caras por outras mais baratas.

Tratando Ruídos na Carga de Trabalho

O PROFUSE é capaz de prever alterações na carga de trabalho baseando-se em comportamentos históricos do sistema. Contudo, não é possível garantir que a carga de trabalho corrente apresente a mesma curva que a utilizada para a construção do histograma de acessos. Ainda assim, a MIN é capaz de reagir a pequenas distorções entre a CdT contida no histograma e a real, ajustando a configuração do *array* de instâncias corretamente. Entretanto, existem situações nas quais a diferença entre a carga de trabalho real e a histórica é demasiadamente grande, podendo ocasionar perdas significativas para o provedor do sistema. Como exemplo, considere uma promoção rápida (ou seja, com duração não superior a alguns minutos ou mesmo poucas horas), com ampla divulgação em redes sociais como Twitter e/ou Facebook. O resultado esperado é um aumento significativo e rápido na carga de trabalho em comparação com aquela utilizada como base para a criação da MIN.

Assim, como uma maneira de habilitar o PROFUSE a detectar ruídos na carga de trabalho e ajustar o *array* de instâncias, foi incluído o conceito de *ganchos* (*hooks*, em inglês), que são componentes de monitoramento de unidades críticas do sistema, como tamanho da fila de espera e utilização geral do sistema. Basicamente, um gancho observa uma métrica do sistema e dispara um alarme sempre que esta apresentar um comportamento incomum. Note que ruídos são, em sua essência, eventos raros e imprevisíveis que afetam severamente a carga de trabalho. Contudo, uma vez que tenham passado, a carga de trabalho volta ao seu estado normal. Devido à sua imprevisibilidade, os ruídos devem ser tratados na parte *reativa*

do PROFUSE.

No PROFUSE, os ganchos ajudam a monitorar o tamanho da fila de requisições, taxa de quebra de contratos e utilização do sistema. Como o efeito imediato de um ruído é o aumento de ao menos uma dessas métricas, os ganchos colocam o PROFUSE em um “modo de contingência”, invocando o procedimento de expansão de forma isolada. O Método III descreve como são utilizados os ganchos.

A definição dos patamares (ou valores máximos) para as medidas observadas pelos ganchos ocorre da seguinte maneira: calcula-se a média das medições daquela hora (*m*) e seu desvio padrão (*stdev*) ao longo do tempo. Sempre que a medida atingir um valor superior a *m+stdev*, o gancho aciona a rotina de expansão. Este valor (*m+stdev*) indica que a medida observada está apresentando um comportamento diferente do histórico, caracterizando um ruído.

Método III: Ganchos_PROFUSE
1. Para cada requisição *r* que for recebida no sistema: 1.1. Verificar as medidas US, PPD e TFR. Caso ao menos uma dessas medidas esteja com valor acima do patamar, invoca-se o procedimento de expansão.

4.6. Construção da Máquina de Inferência Nebulosa

Neste trabalho utilizamos como mecanismo de Teoria de Controle a Lógica Nebulosa, mais precisamente a criação de uma Máquina de Inferência Nebulosa (MIN). Em nossa abordagem, a MIN é responsável por calcular a capacidade computacional necessária para manter a qualidade de serviço, solicitando a alocação/liberação de novos servidores.

Nesta seção, apresentamos como determinamos as variáveis linguísticas utilizadas pela MIN, como suas regiões foram definidas e como foi o processo utilizado para criar as regras SE-ENTÃO.

Definição das Variáveis Linguísticas

Um dos desafios encontrados na hora da construção da MIN foi a definição das

variáveis linguísticas. O processo observado para chegarmos ao conjunto final das VLs, utilizadas na MIN, foi o seguinte: após a criação do histograma de acessos e a construção da ferramenta Cloud-Simulator (Apêndice I), foram realizadas diversas rodadas de experimentos. Repare que neste momento foi simulado o sistema sem a utilização de qualquer mecanismo capaz de gerenciar a elasticidade necessária, ou seja, o sistema possuía um número fixo de servidores. Assim, a cada rodada de experimentos, o número de servidores (todos homogêneos, contendo a configuração mais básica) foi sendo aumentado.

Durante os experimentos, foram coletadas periodicamente as seguintes medidas: *taxa de chegada de requisições, taxa de saída, tempo médio de espera na fila, tamanho da fila, número de servidores, utilização do sistema e taxa de quebras de contrato.* Tais rodadas de experimentos geraram um arquivo de *log*, que foi utilizado como entrada para uma análise de seleção/redução do número de atributos. O processo foi feito utilizando a ferramenta Weka[5] e teve como finalidade identificar quais dos atributos iniciais eram redundantes, de modo que pudessem ser combinados com o objetivo de reduzir o conjunto final de variáveis.

Após esta análise, restaram apenas três atributos: *utilização do sistema, taxa de quebra de contratos e tamanho da fila.* Como forma de adicionar a capacidade preditiva à MIN, foram inseridos mais dois atributos a esse conjunto: *tempo restante até o próximo intervalo* e *diferença de carga de trabalho.* Essas duas variáveis são decorrentes da utilização de um histograma como forma de representar a carga de trabalho.

Finalmente, a variável de saída, *número de núcleos*, foi decidida como forma de informar a quantidade necessária de capacidade computacional para manter a QoS com a configuração corrente do sistema e, ao mesmo tempo, levar em consideração a situação em um futuro próximo.

5 http://www.cs.waikato.ac.nz/ml/weka

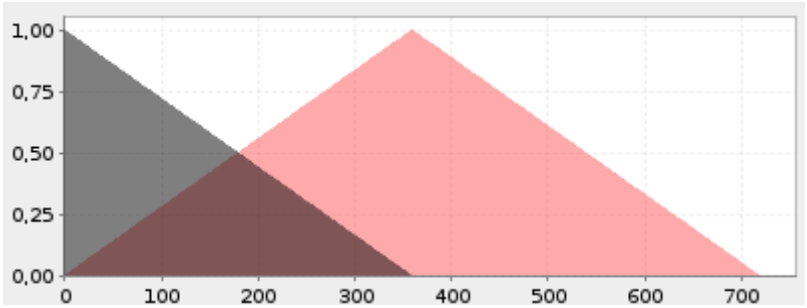


Figura 17: Regiões nebulosas da VL TPPC

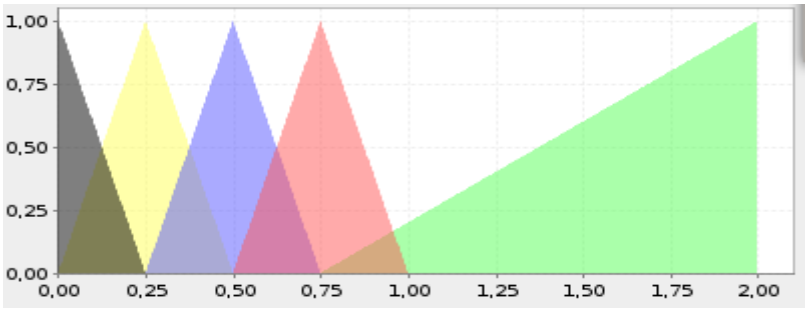


Figura 18: Regiões nebulosas da VL US

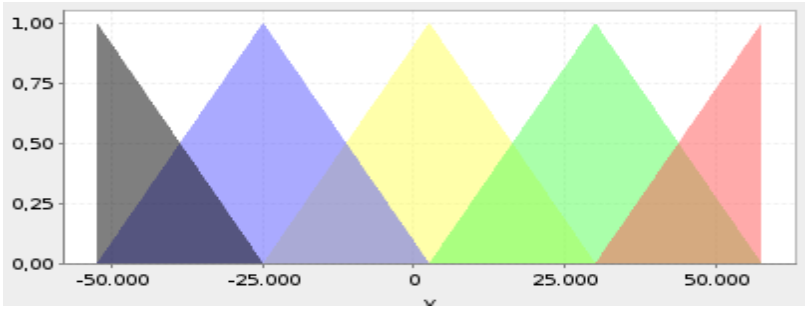


Figura 19: Regiões nebulosas da VL DCdT

As regiões nebulosas de cada VL foram definidas da seguinte forma: utilizando o mesmo *software* Weka, foi realizada uma Análise de *Clusters* através do algoritmo *k-means* (Kanungo *et al.*, 2002), como forma de identificar como cada variável se dividia. *Clusters* próximos foram combinados, de modo que cada VL ficasse com até cinco regiões. Este processo de combinação de *clusters* próximos foi realizado pelo Google em (Sharma *et al.*, 2011), onde a intenção era caracterizar a carga de trabalho para definir estratégias de alocação de tarefas nos servidores de modo eficiente. Contudo, a combinação de *clusters* e a

consequente redução do espaço de cada VL, nesta tese, tem como objetivo limitar o número de regras geradas. As regiões nebulosas finais de cada VL são exibidas nas figuras 17 a 22.

Criação das Regras

O processo para criação das regras SE-ENTÃO foi conduzido da seguinte maneira: após terem sido definidas as regiões nebulosas para cada VL e o tempo máximo que cada requisição pode demorar (valor do *deadline*), o *log* criado na etapa anterior foi reanalisado. Para cada entrada, as métricas correntes (estado do sistema no momento da medição) foram

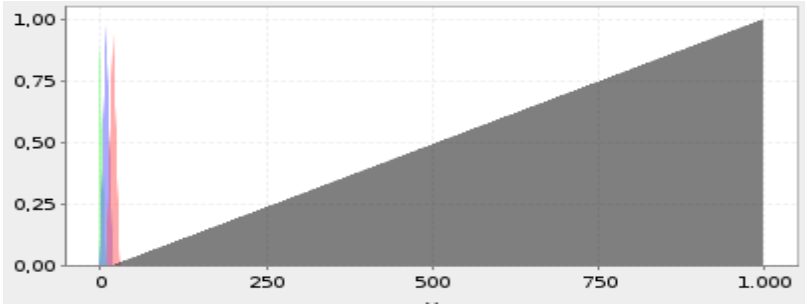


Figura 20: Regiões nebulosas da VL TFR.

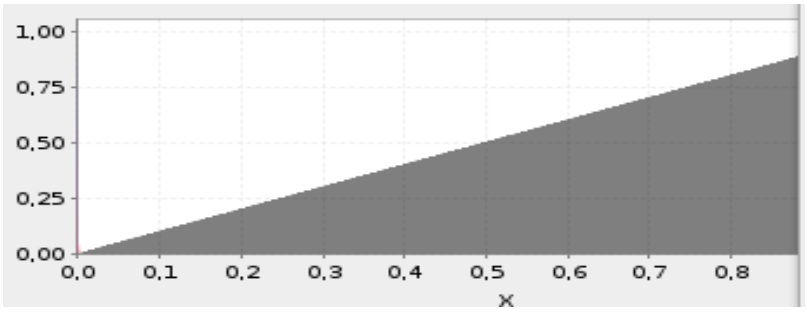


Figura 21: Regiões nebulosas da VL PPD

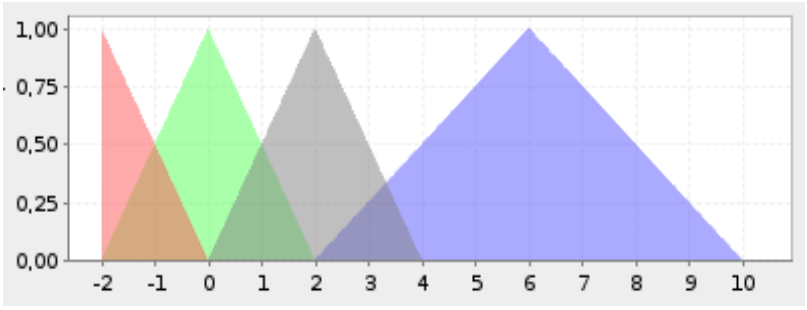


Figura 22: Regiões nebulosas da VL de saída NS

fuzzyficadas, isto é, foram convertidas nas regiões nebulosas que apresentaram o maior grau de pertinência. Em seguida, foi analisado como o sistema se comportou na medição *seguinte*.

Dessa forma, foi possível verificar como o sistema se comportou de uma medição para outra, viabilizando definir as ações a serem tomadas para o aumento do número de servidores alugados. Por exemplo, considere que o *log* contivesse as entradas apresentadas na tabela 6. Repare que, na segunda linha, o percentual de perda de *deadlines* aumentou, caracterizando que uma ação (aluguel de novas instâncias) deveria ser tomada entre as duas medições.

Tabela 6: Exemplo de *log* com as medidas *fuzzyficadas*.

#Linha	TPPC	US	DCdT	TFR	PPD
1	Longo	Alta	Baixa	Grande	Baixo
2	Longo	Proibitivo	Baixa	Imensa	**Alto**
3	…	…	…	…	...

Assim, foi possível criar o conjunto inicial de regras. Regras que possuíam a mesma ação (parte após o “ENTÃO”) foram combinadas com o conectivo “OU” como forma de reduzir o conjunto final. Por exemplo, as regras:

```
SE US é PROIBITIVA ENTÃO NN é EXPANDIR
SE TFR é IMENSA ENTÃO NN é EXPANDIR
```

seriam combinadas em

```
SE US é PROIBITIVA ou TFR é IMENSA ENTÃO NN é EXPANDIR
```

Por fim, alguns ajustes manuais foram feitos nas regras como forma de eliminar redundâncias. Cabe ressaltar que tal ajuste manual foi possível devido ao formato legível do arquivo descritor da MIN, sendo esta uma das vantagens da utilização de Lógica Nebulosa. O conjunto final das regras é constituído por cinco regras, das quais três delas são de natureza reativa, enquanto as duas restantes possuem caráter preditivo.

Regras Preditivas

As regras preditivas utilizam as variáveis TPPC e DCdT para avaliar se a carga de trabalho está próxima de sofrer alterações significativas ou não. Se for o caso, outras variáveis são analisadas (US e PPD) para decidir a quantidade de capacidade computacional necessária para manter a qualidade de serviço. As regras preditivas criadas foram:

```
REGRA 1:
      se dcdt é grandePositivo
      e tppc é curto
      e (us é muitoOcupado
      ou us é proibitiva)
      então ns é expandirRápido;

REGRA 2:
      se dcdt é positivo
      e tppc é curto
      e (ppd não é baixo
      ou tfr não é pequena)
      então ns é expandir;
```

Repare que não há obrigações de retração do *array* de instâncias nas regras preditivas da MIN. Como o objetivo do PROFUSE é manter a taxa de quebra de *deadlines* a mais baixa possível (e, consequentemente, a QoS alta), foi utilizada uma abordagem mais conservadora, onde a principal preocupação está em encontrar aumentos da carga de trabalho em potencial, que poderiam afetar o tempo de processamento das requisições. Desta forma, o PROFUSE somente libera as instâncias em sua parte reativa.

Regras Reativas

As regras reativas são responsáveis por monitorar o comportamento do sistema periodicamente e adaptar o conjunto de instâncias sempre que uma mudança na carga de trabalho for detectada. Estas regras utilizam somente as VLs PPD, US e TFR. As regras criadas foram:

```
REGRA 3:
      se (ppd é muitoAlto
      ou us é proibitiva
```

```
      ou tfr é imensa)
      então ns é expandirRápido;

REGRA 4:
      se (ppd é alto
      ou (tfr é longo
      e us é muitoOcupado))
      então nos é expandir;

REGRA 5:
      se us é ocioso ou us é normal
      então ns é encolher;
```

4.7. Discussão

Este capítulo apresentou o PROFUSE, um Modelo de Elasticidade completo. Ele é dividido em dois módulos: o primeiro, chamado WOM, é responsável por analisar a carga de trabalho e definir se é necessário alugar mais servidores com o objetivo de manter a Qualidade de Serviço. Igualmente, o WOM determina quando é seguro, do ponto de vista de QoS, reduzir o tamanho do *array* de instâncias.

Para realizar esses cálculos, o WOM utiliza uma Máquina de Inferência Nebulosa, cujas etapas para criação foram explicadas. Devido às suas regras “SE-ENTÃO”, a utilização de uma MIN permite ajuste fino por parte de especialistas sem a necessidade de novas etapas de aprendizado, comuns em outras técnicas de Teoria de Controle. Ainda mais, como o relacionamento entre as variáveis envolvidas na definição do tempo de resposta das requisições é de difícil representação matemática, a utilização da MIN facilita essa tarefa – uma vez que tem como sua finalidade a representação de dados e relacionamentos imprecisos.

O outro módulo do PROFUSE, chamado SAM, é responsável por se comunicar com o provedor IaaS e realizar as aquisições e/ou liberações de instâncias, seguindo a política de redução de custos relacionados ao aluguel de servidores. O SAM utiliza a saída do WOM como entrada para determinar quantos servidores devem ser adquiridos ou liberados junto ao provedor IaaS, através de um método incremental.

Por fim, o PROFUSE utiliza ganchos para tratar ruídos nas cargas de trabalho. Ruídos são alterações efêmeras e bruscas na CdT, não sendo detectados pela MIN. Dessa forma, os ganchos são componentes que monitoram algumas medidas essenciais, como a utilização do sistema, e acionam a rotina de expansão sempre que tais medidas ultrapassem um valor máximo, garantindo a manutenção da QoS.

Cabe ressaltar que as estratégias aqui apresentadas não são de uso exclusivo de uma MIN. É possível utilizar outras técnicas de TCR para realizar o ajuste da capacidade computacional do sistema, inclusive de forma preditiva. Exemplos de outras técnicas incluem Árvores de Decisão, em uma abordagem semelhante à apresentada em (Dutta *et al.*, 2012), e Redes Neurais.

O próximo capítulo descreve e analisa os resultados dos experimentos realizados para avaliar o PROFUSE.

Capítulo 5 - Avaliação Experimental

Como forma de validar os algoritmos que compõem o PROFUSE, foram realizados um conjunto de experimentos utilizando o Cloud-Simulator (Apêndice I). A seguir, descrevemos as cargas de trabalho utilizadas nos experimentos, as suas implicações na definição da MIN e os valores utilizados em cada parâmetro do simulador.

5.1. Cargas de Trabalho Utilizadas

O trabalho (Mao; Humphrey, 2011) caracteriza as cargas de trabalho típicas de um sistema em nuvens IaaS como:

1. *Estável:* esta CdT apresenta um comportamento quase uniforme, com variações suaves e sem a presença de picos. A figura 23a exibe um exemplo de CdT estável.

2. *Picos*: esta CdT apresenta um comportamento diferente da estável, possuindo variações, bruscas ou não, ao longo do tempo. A figura 23b exibe um exemplo de CdT com picos. Esta CdT é a mais provável de acontecer em sistemas de comércio eletrônico, pois os picos e as depressões representam momentos de maior e menor acessos.

3. *Crescente*: representa a CdT de um sistema cujo número de acessos não pára de crescer ao longo do tempo. Representa os “ruídos” descritos na seção 4.5.. A figura 23c exibe o gráfico que representa esta CdT.

4. *Liga e Desliga*: esta CdT representa a chegada e execução de lotes de tarefas, alternando bruscamente entre duas CdTs estáveis. Ocorre com frequência em tarefas que executam em segundo plano, como compactação e análise de logs, arquivamento de mensagens em um servidor de e-mails, etc. A figura 23d exibe graficamente como esta CdT se comporta.

Note que estas cargas de trabalho podem ser combinadas e representar diferentes períodos do mesmo sistema ao longo do tempo. Por exemplo, durante uma hora o sistema está estável e em seguida há um aumento vertiginoso no número de acessos. Após 10 minutos crescendo, a CdT passa a apresentar um comportamento padrão, com picos de chegadas de requisições. Para representar a CdT com Picos foi utilizada a CdT de um sistema real, cujo nome não pode ser revelado devido a um acordo de confidencialidade.

Todas as CdTs foram implementadas no Cloud-Simulator. Contudo, os formatos Estável, por apresentar poucas e baixas variações, e Liga e Desliga, por ocorrer em cenários diferentes daquele em que esta tese se baseou, não foram considerados nos experimentos.

As figuras 24 e 25 mostram graficamente as CdTs com Picos e Crescente utilizadas nos experimentos.

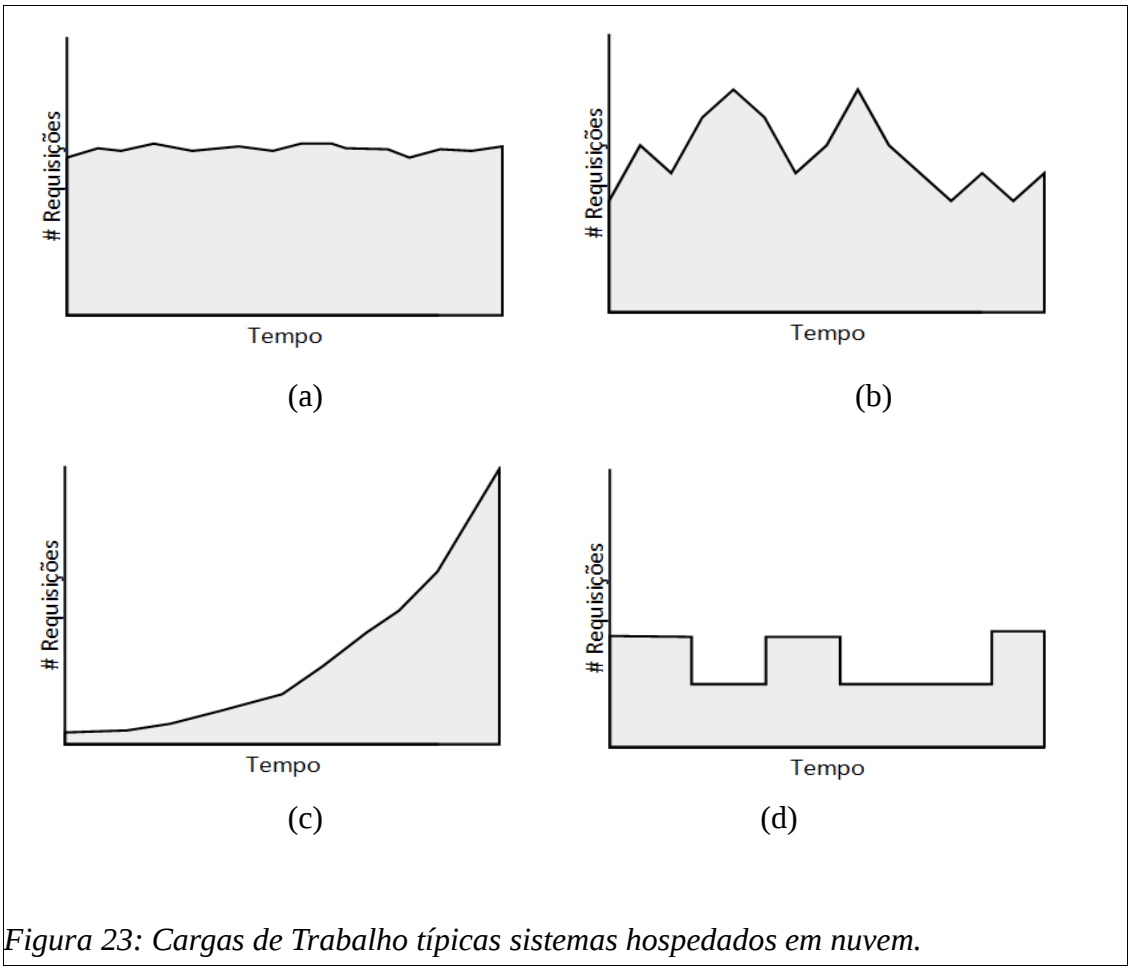


Figura 23: Cargas de Trabalho típicas sistemas hospedados em nuvem.

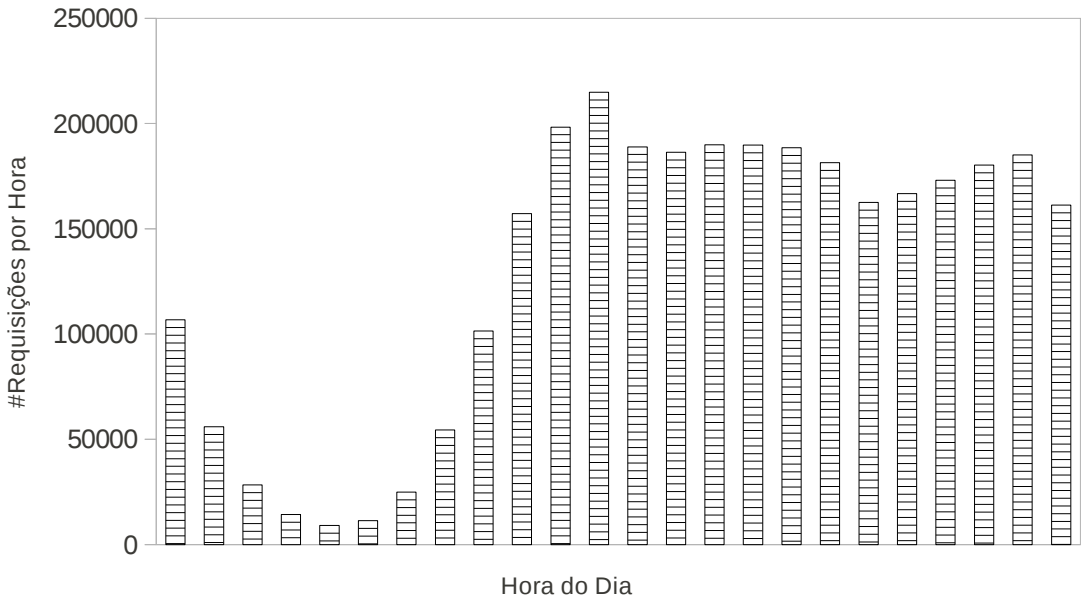


Figura 24: CdT com Picos, extraída do sistema real.

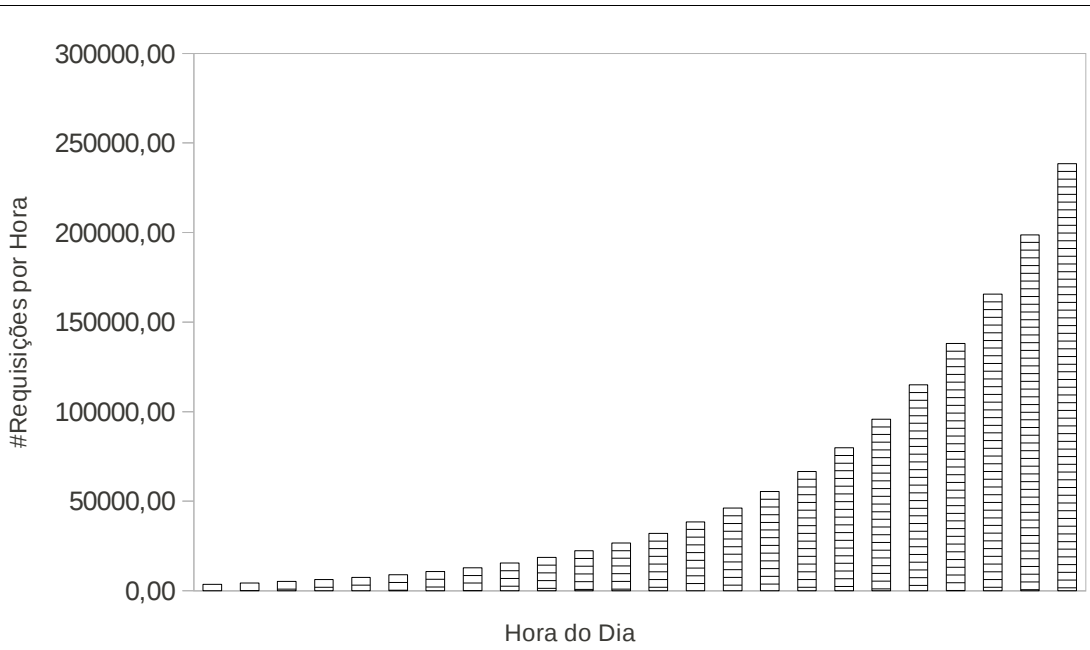


Figura 25: CdT Crescente.

5.2. Parâmetros

Para o ajuste de diversos parâmetros, como tempo de iniciação de um servidor, tempo de desligamento, etc., foram utilizados os valores apresentados no trabalho (Mao;

Humphrey, 2012). Na tabela 7 os parâmetros do SED e os valores utilizados nos experimentos encontram-se listados (valores separados por vírgula indicam que mais de um valor foi utilizado.

Tabela 7: Parâmetros iniciais utilizados nos experimentos.

Parâmetro	Valor(es)
Tamanho inicial do *array* de instâncias	5
Tempo médio para aquisição de instância	97s
Tempo médio para liberação de uma instância	8s
Número máximo de instâncias	∞, 20
Custo por hora de um servidor *single-core*	$0.02
Custo por hora de um servidor *dual-core*	$0.34
Custo por hora de um servidor *quad-core*	$2.00
Tempo de resposta máximo (*deadline*)	4s
Taxa de Conversão de Clientes	0.01, 1
Valor de uma requisição	$100, $0.001
Probabilidade de a requisição gerar receita	1/20, 1
Intervalo entre as coletas de medidas do WOM	120s

5.3. Configurações

As seguintes configurações foram utilizadas:

A) Número de servidores ilimitados, CdT sem ruídos e ganchos desativados.
B) Número de servidores limitados em 20, CdT sem ruídos e ganchos desativados.
C) Número de servidores limitados em 20, CdT com ruídos e ganchos desativados.
D) Número de servidores limitados em 20, CdT com ruídos e ganchos ativados.

Realizamos diversos experimentos para averiguar o comportamento do PROFUSE e, em alguns momentos, o comparamos com um método que procura manter a utilização do sistema constante. Este método possui o nome de Utilização Fixa (UF) e, periodicamente, verifica se houve alterações na taxa de chegada e ajusta o *array* de instâncias de modo que a

utilização do sistema permaneça sempre em 0.7. Repare que na configuração A o custo-benefício é linear, pois cada unidade de processamento (*core* de uma instância) possui o mesmo custo.

Finalmente, todos os resultados apresentados representam uma média de 10 rodadas de simulação.

5.4. Resultados

Como o PROFUSE se trata de um Modelo de Elasticidade para auxiliar sistemas comerciais em nuvens IaaS, a principal métrica dos experimentos foi a receita total ao final do dia, excluídos os custos com alocação de servidores. Outras métricas utilizadas foram o percentual de *deadlines* perdidos e a receita perdida em decorrência dos abandonos.

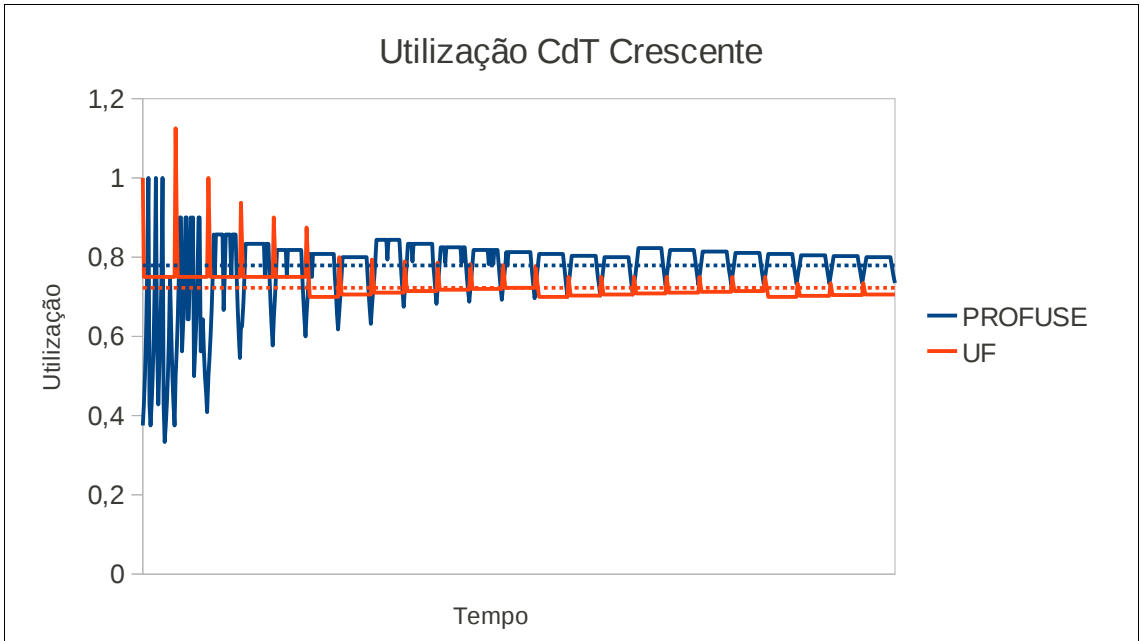


Figura 26: Utilização do sistema ao longo de um dia com a CdT Crescente.

As figuras 26 e 27 mostram a Utilização do Sistema (US) utilizando as cargas de trabalho Crescente e com Picos utilizando os Modelos de Elasticidade PROFUSE e Utilização Fixa, que foi fixada em 0.7, com a configuração A. Em todos os casos o PROFUSE apresentou uma média de US maior sem, no entanto, aumentar o número de *deadlines* perdidos, conforme pode ser conferido nas tabelas 8, 9 e 10.

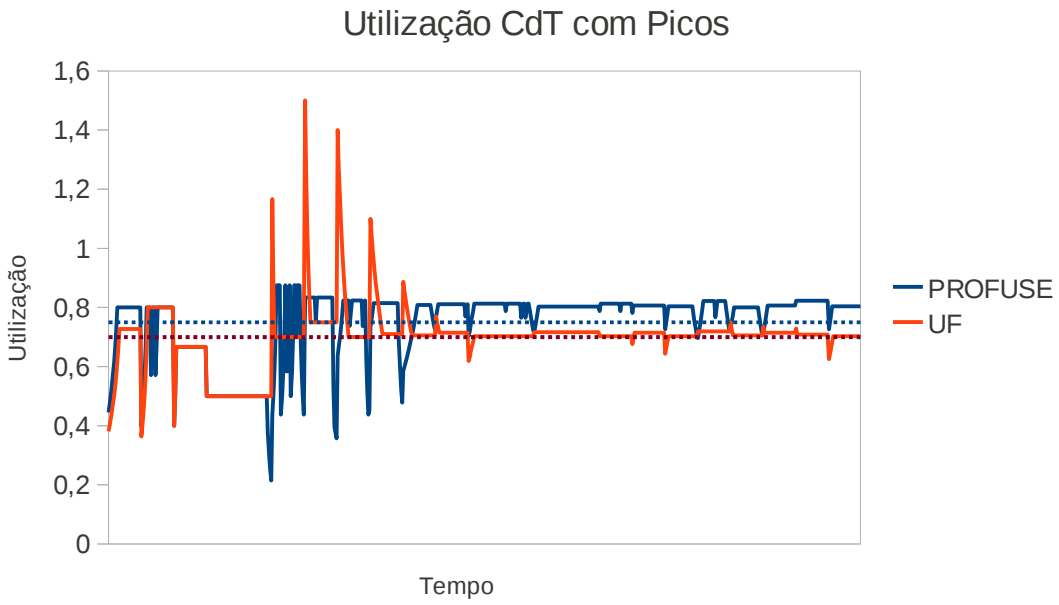


Figura 27: Utilização do sistema ao longo de um dia com a CdT com Picos.

Na figura 26, onde foi utilizada a CdT Crescente, é possível perceber que o PROFUSE leva um tempo até estabilizar completamente, possuindo fortes oscilações no início, quando a taxa de chegadas ainda é baixa. A explicação para este fato está justamente no baixo número de servidores contidos no *array* de instâncias. Por exemplo, se o sistema possui 3 servidores (com a configuração mais simples possível) alocados e a MIN decide por devolver uma instância ao provedor IaaS, estará diminuindo em 1/3 a capacidade computacional total do sistema, causando um impacto maior e forçando o sistema a readquirir novas instâncias. O mesmo não ocorre quando o sistema possui um número maior de instâncias em seu *array*, onde a liberação de um servidor não teria um impacto tão grande.

Analisando a figura 27, podemos perceber os momentos em que o PROFUSE prevê o aumento da CdT, adquirindo novas instâncias previamente e causando as quedas na US que, em seguida, volta ao normal. O modelo UF, em contraste, apresenta um aumento na US, o que ocasiona perda de *deadlines*. Mesmo nesta CdT cuja variabilidade é maior, o PROFUSE apresenta uma US maior. Como a utilização do sistema neste trabalho é calculada como a razão entre as taxas de chegada e de saída, esta pode assumir um valor superior a 1, representando um sistema altamente saturado. Este caso ocorre nas figuras 26 e 27, quando o método de expansão UF é utilizado.

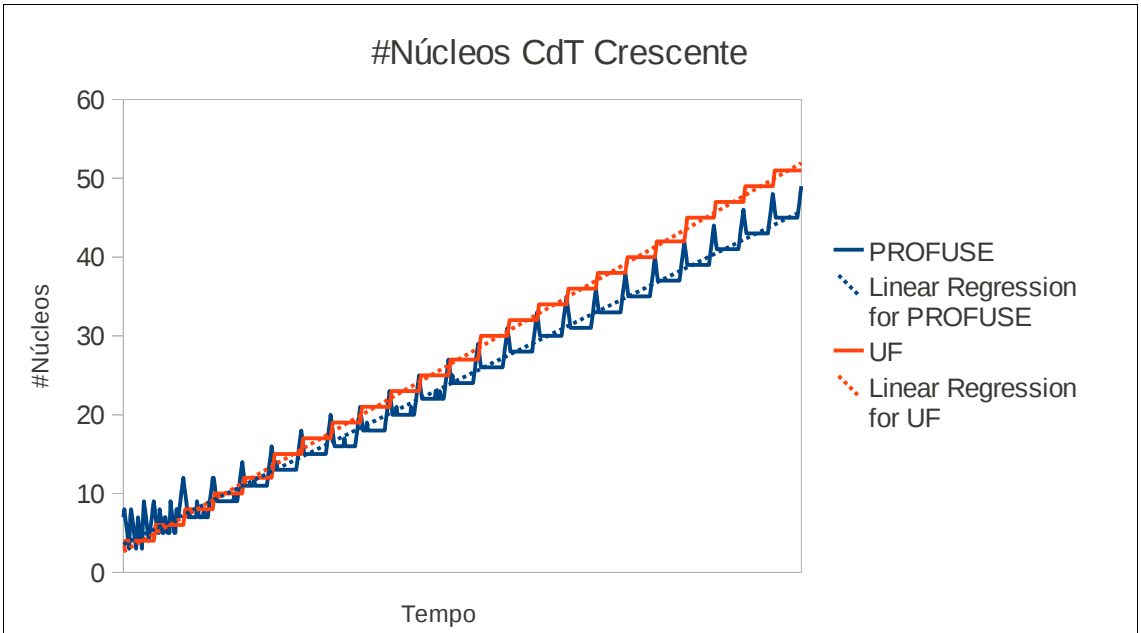


Figura 28: Variação do Poder Computacional da CdT Crescente ao longo de um dia.

As figuras 28 e 29 exibem as capacidades computacionais (sob a forma de número total de núcleos de processamento) dos sistemas, quando estes estão sujeitos às CdTs Crescente e com Picos. As linhas de tendência para cada Modelo de Elasticidade foram incluídas e mostram como ambos os modelos se comportam: com o tempo, o PROFUSE necessita de menos instâncias – e, por consequência, custa menos – para atender aos

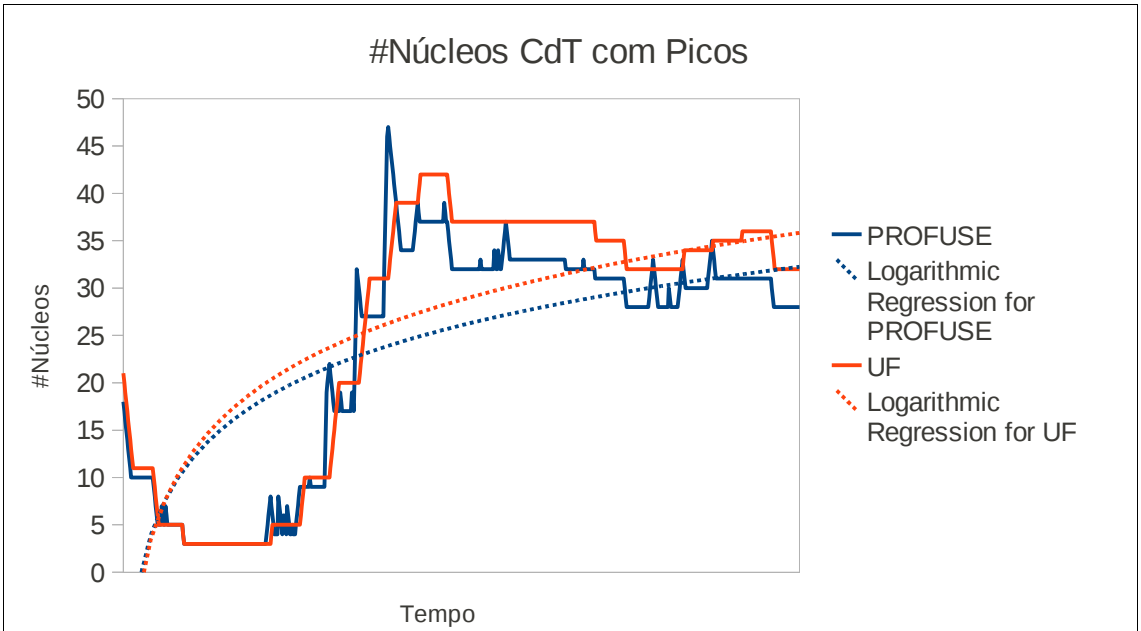


Figura 29: Variação do Poder Computacional da CdT com Picos ao longo de um dia.

requisitos de QoS.

As tabelas 8 e 9 comparam o desempenho do PROFUSE em todas as configurações quando submetido às CdTs com Picos e Crescente, respectivamente.

Tabela 8: Percentual de perdas de *deadlines* do PROFUSE com a CdT com Picos.

Configuração	%Perdas
A	0,00014%
B	0,00012%
C	8,58581%
D	0,00000%

Tabela 9: Percentual de perdas de *deadlines* do PROFUSE com a CdT Crescente.

Configuração	%Perdas
A	0,01928%
B	0,01691%
C	10,79001%
D	0,52795%
D – Conservador	0,00000%

Deve-se notar, nestes resultados, que as configurações A e B possuem desempenhos similares, com vantagem bem pequena para a configuração B. A explicação para este fato está no tempo para disponibilização das novas instâncias. Na configuração B, quando uma instância com quatro núcleos acaba de ser alugada e se tornar disponível, o sistema aumenta sua capacidade computacional em 4 unidades de uma vez. Por outro lado, quando o sistema aloca 4 instâncias de único núcleo, existem pequenas variações no tempo para disponibilização destes novos servidores.

Quando o sistema está sob a configuração C, isto é, existem ruídos na CdT e o sistema está com os ganchos desligados, o sistema apresenta uma forte queda de desempenho, perdendo um percentual alto de *deadlines*: aproximadamente 8,6% quando a CdT com Picos é utilizada e aproximadamente 10,8% quando o sistema enfrenta uma CdT Crescente. Este resultado fraco era esperado, já que os efeitos dos ruídos não são detectados pelo PROFUSE a tempo de este reagir com eficiência.

Contudo, quando os ganchos são ligados (configuração D), pode-se perceber que o PROFUSE volta a apresentar bom desempenho – com a CdT com Picos, não houve perda de *deadlines*, enquanto no caso com a CdT Crescente, o percentual caiu para, aproximadamente, 0,53% de abandonos.

Cabe ressaltar que a MIN utilizada pelo PROFUSE possui uma configuração agressiva, onde a utilização do sistema é considerada média para valores até 0,8. Assim, testamos uma configuração alternativa da MIN, com valores das regiões nebulosas para a variável linguística US mais conservadores, com média 0,6. O resultado está na última linha da tabela 9 e não apresentou nenhuma perda de deadline. Esta estratégia de utilizar configurações diferentes da MIN de acordo com a CdT é semelhante à apresentada em (Dutta *et al.*, 2012), onde a ferramenta SmartScale possui diversas Árvores de Decisão armazenadas, utilizando a mais apropriada para a CdT a qual o sistema está sendo submetido.

Para efeitos de comparação, a tabela 10 compara o desempenho do sistema quando a elasticidade é regida pelo método de Utilização Fixa, que não possui características proativas. Nesta comparação fica evidente o ganho adquirido pela previsão de alterações na CdT proporcionado pela MIN. Esta característica do PROFUSE é responsável por manter a taxa de abandonos do sistema em várias ordens de magnitude menor do que a apresentada pelo outro método na configuração A.

Quando a configuração C é utilizada, percebe-se que os ruídos possuem um impacto muito maior no desempenho do sistema. Como o método UF baseia-se somente na utilização total do sistema para adquirir/devolver instâncias ao provedor sem averiguar outras métricas, as novas aquisições são incapazes de diminuir o tamanho da fila de requisições e, consequentemente, o tempo de espera. Esta situação é corrigida na configuração D, quando os ganchos são ligados. Ainda assim, o desempenho do método UF é inferior ao do PROFUSE, mais uma vez devido ao seu caráter somente reativo.

Tabela 10: Comparação do PROFUSE com Utilização Fixa (CdT com Picos).

	Conf. A	Conf. C	Conf. D
UF	2,57830%	97,98622%	5,47310%
PROFUSE	0,00014%	8,58581%	0,00000%

Com bases nestes resultados é possível calcular, utilizando a equação 3, a receita perdida devido ao mau desempenho do sistema, tanto para um sistema de comércio eletrônico como para um sistema onde a receita é gerada pelo número de visitas que recebe, típico de *blogs*. Os resultados estão na tabela 11.

Tabela 11: Receita perdida com os abandonos (PROFUSE, CdT com Picos).

Configuração	Receita Perdida (E-Commerce)	Receita Perdida (Blog)
A	R$ 0,21	R$ 0,00
B	R$ 0,18	R$ 0,00
C	R$ 18.546,68	R$ 370,93
D	R$ 0,00	R$ 0,00

Para calcular a receita perdida no sistema de comércio eletrônico, utilizamos os seguintes parâmetros: a taxa de conversão de clientes, isto é, o número de visitas que se transformam em vendas, foi 1%, conforme sugerido em (Felipini, 2003). Não foram encontradas referências que indicassem a média de requisições enviadas quando uma venda é realizada. Contudo, fizemos alguns testes em sistemas de vendas pela internet (CompraFácil[6], Americanas.com[7] e Amazon[8]) e verificamos que, de um modo geral, são necessários 7 “cliques” para a conclusão de uma compra. Os passos para este “caminho-feliz” são:

1. Entrada no sistema;
2. Procura pelo produto;
3. Adiciona produto no carrinho;

6 http://www.comprafacil.com.br
7 http://www.americanas.com.br
8 http://www.amazon.com.br

4. Inicia o *check-out*.
5. Cadastra-se no site ou, caso já seja cadastrado, efetua a autenticação;
6. Insere os dados de pagamento e de entrega;e
7. Finaliza o pedido.

Contudo, este é o número mínimo de interações com o sistema para o fechamento de um pedido. De um modo geral, um usuário envia outras requisições (para pesquisar produtos similares, ler comentários de outros clientes, etc.). Assim, utilizamos uma média de 20 requisições por venda, de modo que a probabilidade de uma requisição ser a responsável pelo fechamento do pedido é de 1/20 ou 5%. Por fim, assumimos que as vendas possuem valor médio de R$100,00.

Para o cenário do blog, a receita perdida pode ser calculada de forma mais simples, pois não há taxa de conversão de clientes nem fechamento de pedidos, ou seja, toda requisição enviada para o sistema gera uma receita. Contudo, o valor pago para cada visualização de página é baixo (média de R$0,001). Estes resultados corroboram a robustez do PROFUSE e da utilidade dos ganchos para tratar ruídos.

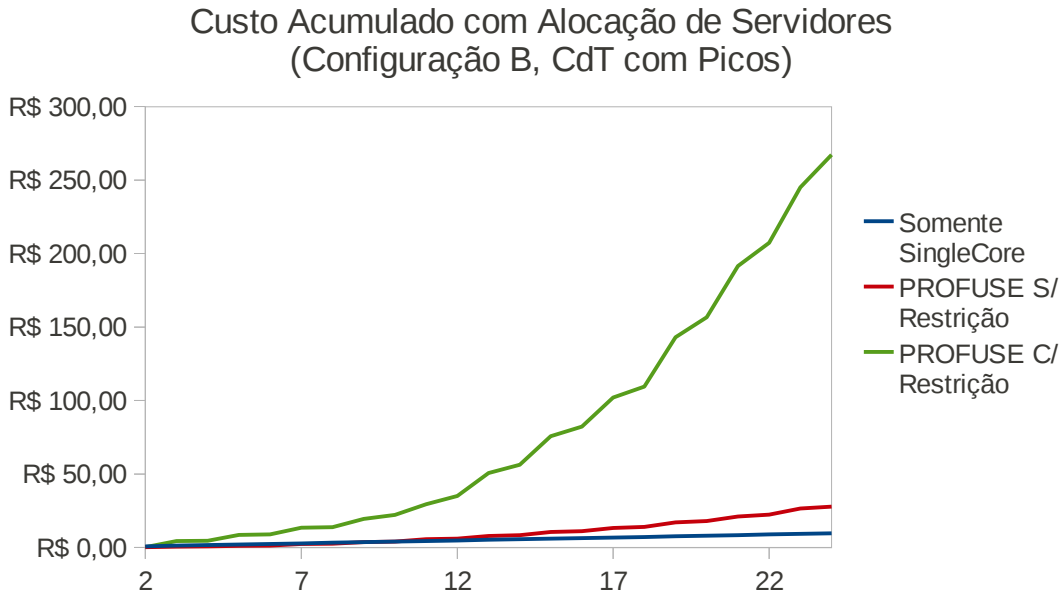


Figura 30: Comparação do custo com alocação de instâncias (limite inferior).

Para verificar a eficácia do método de alocação e liberação de instâncias utilizado

pelo SAM, comparamos o valor gasto pelo PROFUSE com o caso mais barato, onde há 20 instâncias fixas de único núcleo, e com o caso mais caro, onde há 20 instâncias fixas de quatro núcleos. Desta forma pode-se visualizar como a política de alocação de instâncias do PROFUSE se comporta tendo limites inferior e superior. Para facilitar a leitura, as comparações foram feitas em gráficos separados e os resultados são mostrados respectivamente na figuras 30 e 31, exibindo o custo acumulado em cada uma das situações. Finalmente, comparamos o PROFUSE com custo-benefício linear, sem restrição quanto ao limite de servidores, e com custo-benefício não linear, restringindo a 20 o número máximo de instâncias em poder do cliente.

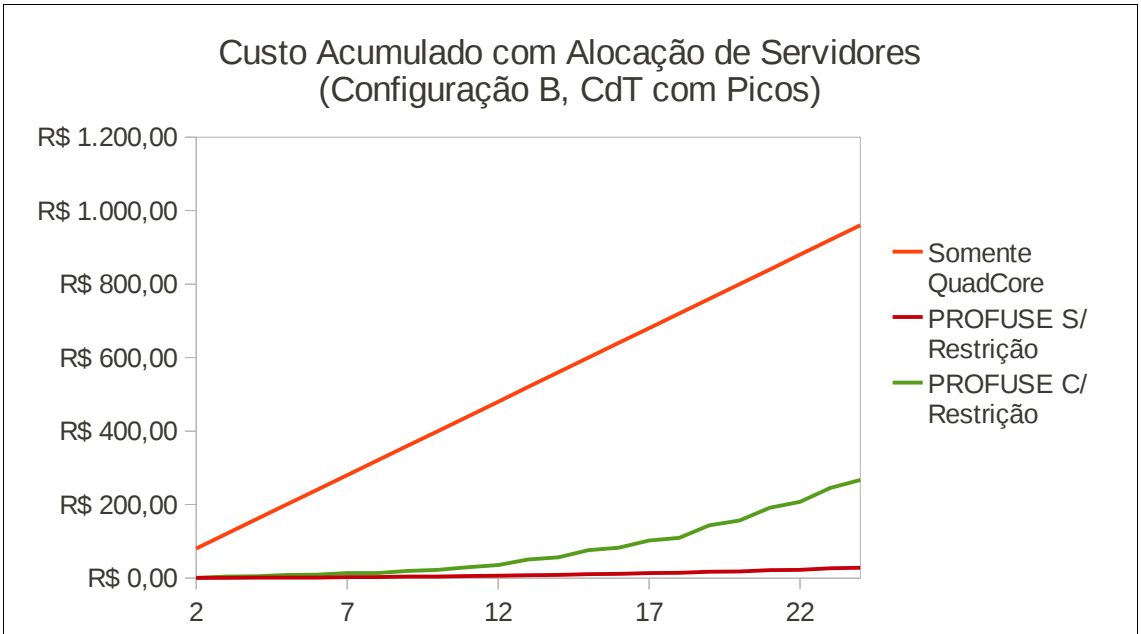


Figura 31: Comparação do custo com alocação de instâncias (limite superior).

Quando comparado com o limite inferior, isto é, somente instâncias de 1 núcleo, o PROFUSE sem restrições apresenta um custo final bem próximo daquele limite – na verdade, apresenta um custo *abaixo* do limite nas primeiras horas, pois utiliza menos instâncias no momento de menor carga do sistema. Paralelo ao aumento do número de acessos, o PROFUSE incrementa a capacidade computacional e ultrapassa o limite inferior na hora 9. Este aumento de custo com alocação de servidores tem como justificativa a manutenção da QoS do sistema, evitando o abandono de usuários e a perda de receita consequente. O PROFUSE com restrições apresenta um custo maior, devido à não linearidade do custo-benefício, contudo aproximadamente 4 vezes abaixo do limite superior (figura 31).

A tabela 12 mostra o custo total de cada estratégia, somando a receita perdida com o valor gasto com alocação de servidores. É possível notar nesta tabela que, a partir da hora 10, a estratégia com 20 instâncias de 1 núcleo se torna insuficiente para tratar a CdT corrente, causando uma perda de receita várias ordens de magnitude maior do que a apresentada pelas outras estratégias.

É possível notar, pelas figuras 30 e 31 e pela tabela 12, que o PROFUSE é uma alternativa melhor do que o superdimensionamento da capacidade computacional total do sistema, apresentando um custo total inferior ao caso onde preenche-se o *array* integralmente com instâncias da maior configuração.

Tabela 12: Custo acumulado de cada estratégia para sistemas de comércio eletrônico.

Tempo	SingleCore	QuadCore	PROFUSE S/ Restrição	PROFUSE C/ Restrição
2	R$ 0,80	R$ 80,00	R$ 0,16	R$ 0,28
3	R$ 2,00	R$ 120,00	R$ 0,58	R$ 4,28
4	R$ 3,60	R$ 160,00	R$ 0,74	R$ 4,56
5	R$ 5,60	R$ 200,00	R$ 1,16	R$ 8,56
6	R$ 8,00	R$ 240,00	R$ 1,32	R$ 8,84
7	R$ 10,80	R$ 280,00	R$ 2,34	R$ 13,38
8	R$ 14,00	R$ 320,00	R$ 2,54	R$ 13,76
9	R$ 17,60	R$ 360,00	R$ 3,68	R$ 19,46
10	R$ 20.397,31	R$ 400,00	R$ 3,98	R$ 22,18
11	R$ 40.940,86	R$ 440,00	R$ 5,58	R$ 29,42
12	R$ 61.533,23	R$ 480,00	R$ 6,00	R$ 34,94
13	R$ 82.097,43	R$ 520,00	R$ 7,90	R$ 50,66
14	R$ 102.678,31	R$ 560,00	R$ 8,42	R$ 56,34
15	R$ 123.291,74	R$ 600,00	R$ 10,60	R$ 75,76
16	R$ 143.894,43	R$ 640,00	R$ 11,12	R$ 82,28
17	R$ 164.443,80	R$ 680,00	R$ 13,36	R$ 102,10
18	R$ 185.064,57	R$ 720,00	R$ 14,06	R$ 109,44
19	R$ 205.663,74	R$ 760,00	R$ 17,08	R$ 143,06
20	R$ 226.253,46	R$ 800,00	R$ 18,04	R$ 156,60
21	R$ 246.817,00	R$ 840,00	R$ 21,12	R$ 191,56
22	R$ 267.323,94	R$ 880,00	R$ 22,38	R$ 207,22
23	R$ 287.877,57	R$ 920,00	R$ 26,50	R$ 245,06
24	R$ 308.412,46	R$ 960,00	R$ 27,82	R$ 267,12

Analisando o Impacto dos Ruídos no PROFUSE e a Eficiência dos Ganchos

Os ruídos, conforme descrito na seção 4.5., são rajadas imprevisíveis de requisições, que não constam no Histograma de Acessos. Caso o aumento na taxa de chegadas causado pelo ruído for pequeno, este pode ser tratado normalmente pelo PROFUSE em seu modo

reativo. Contudo, quando a variação é alta, o ruído pode ocasionar uma condição de saturação no sistema, diminuindo a QoS e aumentando a taxa de abandonos, sendo necessária a utilização de um mecanismo adicional, os ganchos, capaz de detectar variações bruscas e inesperadas e invocar a rotina de expansão a tempo de evitar o êxodo de clientes.

Inicialmente, avaliamos o real impacto dos ruídos no PROFUSE sem a utilização dos

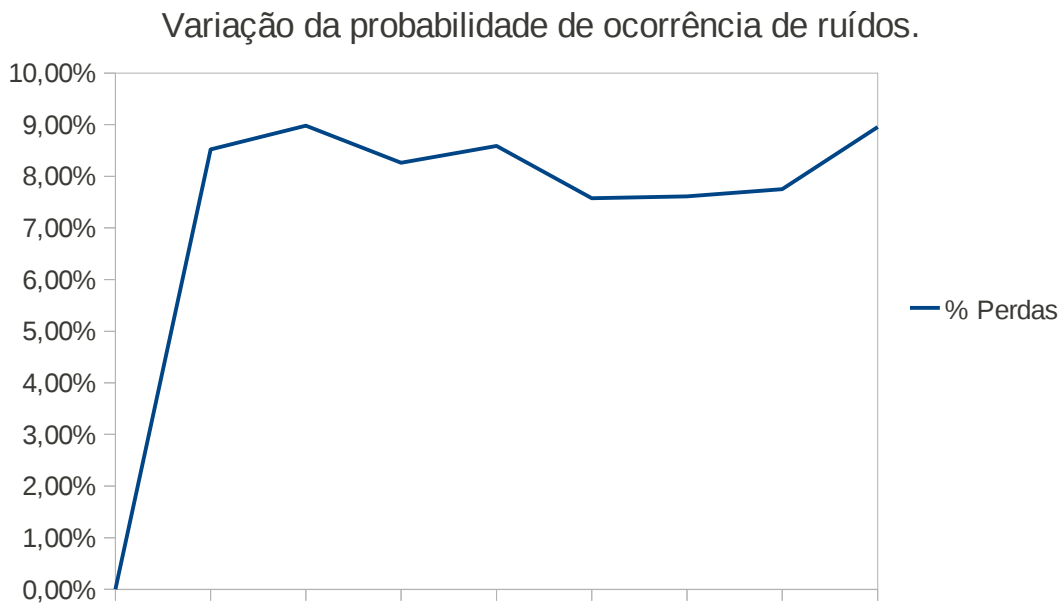


Figura 32: Comportamento do PROFUSE com diferentes probabilidades de ocorrência de ruídos.

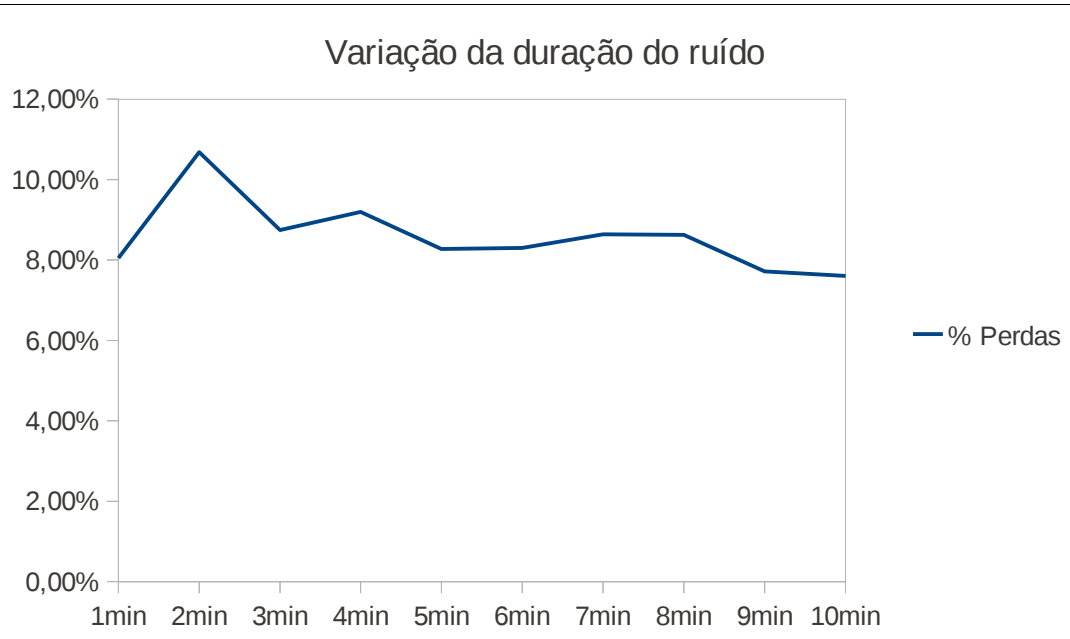


Figura 33: Comportamento do PROFUSE com ruídos de diferentes durações.

ganchos, verificando se estes são realmente necessários. Nesta tese, os ruídos possuem essencialmente 3 variáveis (valores padrão em parênteses): sua frequência (20%), sua duração (10min) e sua intensidade (aumento de 30% na CdT).

As figuras 32 e 33 mostram como o PROFUSE se comporta ao longo de valores nas duas primeiras variáveis: a figura 32 mostra a perda de *deadlines* quando os ruídos tornam-se mais frequentes, enquanto a figura 33 mostra a mesma métrica para diferentes durações de ruídos. Nestes experimentos utilizamos a configuração C.

É interessante notar no primeiro gráfico que a maior perda de desempenho do sistema ocorre quando os ruídos ocupam 10% da CdT. Mesmo para cenários onde a incidência de ruídos é mais frequente, o desempenho do sistema se mostra estabilizado, com uma taxa de perda de *deadlines* entre 7% e 9%., o que nos leva a deduzir que a partir de 10%, os ruídos passam a ser vistos pelo PROFUSE como parte integrante da CdT e a MIN é capaz de evitar percentuais de perdas de *deadlines* ainda maiores.

Na figura 33 nota-se que ruídos demorados não possuem o mesmo impacto que os rápidos. Este resultado era esperado, já que o intervalo entre as medições é 120 segundos, ou seja, 2 minutos. Desta forma, se um ruído for mais curto do que este intervalo, ele pode iniciar e terminar sem que o PROFUSE sequer seja acionado e tente corrigir o *array* de instâncias.

Com estes resultados, configuramos o simulador para que os ruídos tivessem uma duração média de 2 minutos e frequência igual a 10% do total da CdT, como forma de avaliar o impacto dos ruídos no PROFUSE para diferentes intensidades, aqui representadas pelo percentual de aumento na CdT por eles causado. No mesmo experimento, avaliamos a eficácia e a eficiência dos ganchos. Os resultados estão exibidos na figura 34. Foram utilizadas as configurações C e D, para as CdTs Crescente e com Picos.

Este gráfico exibe com clareza que ruídos de baixa intensidade possuem baixo impacto no sistema, com o PROFUSE sendo capaz de tratá-los e garantir a QoS. Contudo, para intensidades maiores (a partir de 10% para a CdT Crescente e de 15% para a CdT com Picos), o desempenho fica prejudicado. Repare os “saltos” apresentados no gráfico na configuração C, isto é, com os ganchos desligados. Os ruídos interferiram no tempo de processamento das requisições, aumentando a perda de *deadlines*, diminuindo a receita final. Contudo, quando o uso de ganchos foi acionado, a média da perda de contratos foi igual a 0 para a CdT com Picos e próxima a 0 para a CdT Crescente. Assim, sempre que alguma

medida do sistema apresentasse um valor considerado fora do normal (por exemplo, fila de requisições com um valor acima da média + desvio padrão), o gancho acionava a MIN, aumentando a capacidade computacional e garantindo a QoS.

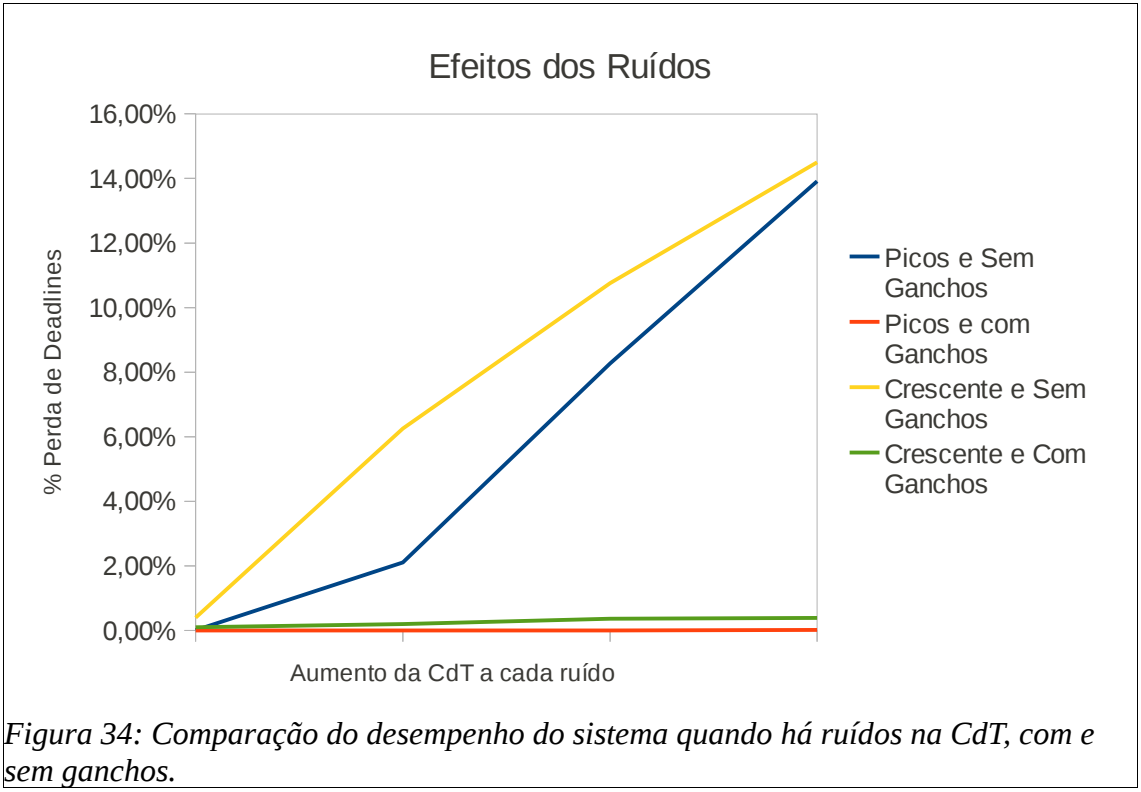


Figura 34: Comparação do desempenho do sistema quando há ruídos na CdT, com e sem ganchos.

Latência de Medição

Por fim, a figura 35 exibe o desempenho do sistema para diferentes latências, isto é, intervalos de medição do WOM: 1, 2, 5 e 10 minutos. Nestes experimentos foi utilizada a configuração D e a CdT com Picos. Intuitivamente, quanto menor o intervalo entre as medições, menor seria o número de perda de *deadlines* pois o sistema detectaria anomalias – e as contornaria – mais rapidamente. Contudo, o gráfico nos mostra que para intervalos pequenos, no caso 1 minuto, o desempenho não foi o melhor: com 2 minutos entre as medições o número de quebras de contratos chegou a 0, contra 1000 *deadlines* perdidos no cenário anterior. Este fenômeno pode ser explicado pelo fato de que o tempo de reação do sistema é, no mínimo, igual ao tempo de aquisição de uma nova instância – no caso, 94 segundos. Quando o intervalo é menor do que este tempo de reação, duas medidas são

coletadas sem que a correção da primeira tenha sido concretizada, levando o WOM, em alguns casos, a acionar uma segunda vez o mecanismo de expansão. Este fenômeno pode acontecer inclusive para a retração do *array* de instâncias, o que faria com que este ficasse menor do que o necessário e, como consequência, uma maior saturação do sistema, levando a, em algumas oscilações, a perder *deadlines*.

Conforme o intervalo de entre as medições aumenta, apesar de o WOM não realizar aquisições concorrentes, o sistema volta a apresentar quebras de contrato. Estas taxas aumentam pois quando alguma anomalia (que não chegue a se caracterizar em um ruído) aparece na carga de trabalho, o PROFUSE demora a detectá-la e a reagir a ela.

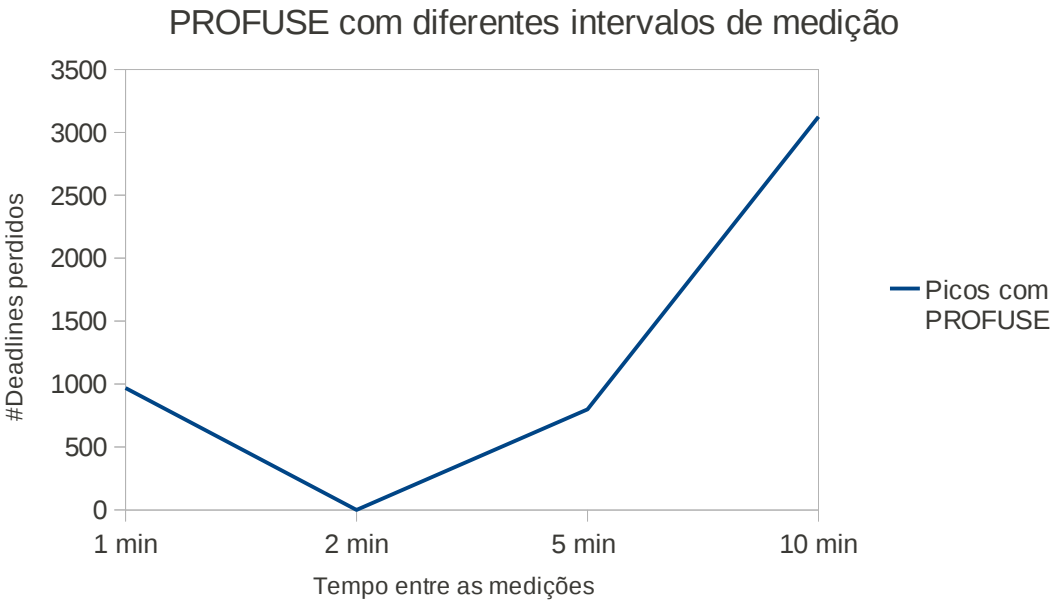


Figura 35: Comparação do desempenho do PROFUSE com diferentes intervalos de medição.

5.5. Discussão

Este capítulo apresentou os resultados dos experimentos que foram utilizados para comprovar a eficácia e a eficiência do PROFUSE como Modelo de Elasticidade para sistemas hospedados em nuvens IaaS. Foram descritas cargas de trabalho utilizadas nos experimentos (Crescente e com Picos) e definidas 4 configurações, começando em um cenário básico e adicionando dificuldades e as soluções propostas uma a uma. Todos os experimentos foram executados no Cloud-Simulator (Apêndice I).

Os experimentos iniciais mostraram como o PROFUSE se comportou em sua forma mais crua, em um cenário sem a presença de ruídos e podendo alugar um número ilimitado de instâncias do provedor IaaS. Em todos os casos, o PROFUSE se mostrou superior ao método da Utilização Fixa, reduzindo custos com alocação de instâncias e mantendo o número de quebras de contrato próximo a 0.

Em seguida, foram mostrados os resultados obtidos quando o provedor IaaS limita o número de instâncias e a relação custo-benefício das configurações não é linear. Assim, o PROFUSE conseguiu manter o balanço ao final do dia quase constante para todos os casos – a exceção foi a CdT Crescente. Esta teve seu desempenho comprometido pela configuração agressiva inicial da Máquina de Inferência Nebulosa do PROFUSE, que tentava manter a Utilização do Sistema o mais alta possível. Após um ajuste nos valores da Região Nebulosa da Variável Linguística US, o PROFUSE foi capaz de executar o mesmo experimento sem quebrar nenhum contrato.

O resultado exibido em seguida mostrou o impacto dos ruídos e como eles foram contornados neste trabalho com a utilização dos ganchos, que monitoraram o algumas componentes do sistema em busca de anomalias, ativando a rotina de expansão de forma isolada sempre que fosse necessário.

Finalmente, o último experimento mostrou como o PROFUSE se comporta com diferentes intervalos entre as medições, e ficou evidente que nem sempre tempos menores implicam em resultados melhores.

Capítulo 6 - Conclusões E Trabalhos Futuros

Empresas de comércio eletrônico têm uma particular dificuldade na hora de definir a quantidade e as configurações dos servidores que irão hospedar seus sistemas. Tal dificuldade tem como uma de suas causas principais a imprevisibilidade da demanda, uma vez que a internet traz consigo meios de divulgação cada vez mais rápidos e poderosos, como as redes sociais e os e-mails.

É necessário levar em consideração, ainda, a impaciência dos consumidores virtuais, fenômeno que se deve à crescente concorrência entre as soluções de comércio eletrônico. Para obter a satisfação dos consumidores é necessário manter uma Qualidade do Serviço mínima e garantir que as requisições enviadas serão processadas dentro de um tempo máximo. Assim, impede-se a saída de clientes para sistemas concorrentes e a consequente perda financeira.

Tendo em vista todo este cenário, torna-se de grande relevância a escolha cuidadosa da configuração do *array* de instâncias que hospedará o sistema, uma vez que um subdimensionamento pode levar a uma situação de saturação, com desempenho ruim e alta taxa de abandono como consequência.

Uma solução conservadora consiste em fazer um superdimensionamento da capacidade computacional necessária, sendo esta utilizada em sua totalidade somente em raros momentos de pico. O grande problema dessa abordagem está na possibilidade de a maioria dos servidores permanecer desligada, causando desperdício de recursos da empresa.

O paradigma da Computação em Nuvem, contudo, trouxe consigo uma nova abordagem para este cenário: hospedar os sistemas em provedores de nuvem, pagando pela quantidade de instâncias-hora consumidas, e tirando proveito de uma de suas características principais, conhecida como elasticidade. Assim, os administradores dos sistemas poderiam alugar novos servidores em situações de pico, aliviando a carga total e garantindo a QoS prestada aos clientes.

O novo desafio que chegou com esta solução foi a automatização deste processo de aluguel e devolução de instâncias, combinada com garantias de alta utilização total, para evitar o custo desnecessário com recursos ociosos, e de QoS, para reduzir as perdas financeiras decorrentes dos abandonos. Essa situação gera um impasse, pois para aumentar a QoS é necessário aumentar o número de instâncias no *array* e vice-versa.

Esta tese apresentou um conjunto de técnicas as quais foi dado o nome de PROFUSE. Utilizando uma carga de trabalho histórica, sob a forma de um histograma com a quantidade de requisições enviadas por hora, o PROFUSE cria uma Máquina de Inferência Nebulosa para resolver o problema de determinar a diferença de poder computacional que é necessária para resolver o impasse citado anteriormente. Em nossa PROPOSTA, a MIN tem o papel de determinar se a capacidade computacional do *array* de instâncias deve ser aumentada, mantida ou diminuída, sendo eficaz tanto em reagir a como em prever alterações na carga de trabalho.

Ao mesmo tempo, o PROFUSE aluga e libera instâncias de um modo incremental, levando em consideração o custo-benefício das unidades computacionais mínimas (que, nesta tese, são os núcleos computacionais) entre as diversas configurações, conseguindo uma redução considerável (em nossos experimentos, de até uma ordem de magnitude) no custo do aluguel de instâncias quando comparado com a solução conservadora de superdimensionamento. A técnica adotada no PROFUSE funciona tanto para provedores que limitem a quantidade de servidores que a empresa pode alugar como para os provedores que não estabelecem um número máximo. Finalmente, nosso algoritmo é capaz de lidar com não-linearidade do custo-benefício das instâncias, tomando o cuidado de somente incrementar a configuração do *array* se esta tiver um custo inferior ao causado pelas quebras de contrato.

Por fim, o PROFUSE possui um mecanismo, chamado de gancho, que detecta ruídos na carga de trabalho e aciona a rotina de expansão isoladamente, evitando que picos inesperados de acesso causem impacto na qualidade de serviço.

Os experimentos foram realizados com o Cloud-Simulator, um Simulador de Eventos Discretos. Os resultados mostraram a robustez do PROFUSE, que é capaz tanto de reduzir o custo final com o aluguel de instâncias como manter a QoS total, com o percentual de quebras de *deadlines* igual a 0 em diversos cenários.

Nos experimentos foram utilizadas duas cargas de trabalho típicas de sistemas em nuvem, sendo uma delas retirada de um sistema real. Para a CdT Crescente, na qual o número de acessos cresce exponencialmente ao longo do tempo, foi necessário realizar um ajuste na MIN de forma que esta alugasse mais servidores para diminuir a quebra de contratos. Abaixando a média da utilização do sistema de 0,8 para 0,6, o PROFUSE foi capaz de evitar qualquer quebra de contrato.

Por fim, os experimentos mostraram que a frequência das medições influencia no desempenho final. Quando o intervalo entre as medições é muito pequeno, duas extrações de valores são tomadas sem que a resposta da primeira tenha sido consolidada. Em contraste, se os intervalos forem grandes demais, o sistema demora para detectar mudanças na carga de trabalho e acaba não adaptando o *array* de instâncias a tempo de evitar a queda na qualidade de serviço.

6.1. Trabalhos Futuros

Dentre as pesquisas futuras que podem usar esta tese como base, pode-se destacar:

1. *Método para identificação dos padrões de Cargas de Trabalho.* Nesta tese, utilizamos três cargas de trabalho típicas de sistemas hospedados em nuvens: Constante, Crescente e com Picos. Contudo, tais CdTs podem ocorrer individualmente ou em grupo. Pensando neste último caso, pode-se criar um método para identificar quando a CdT mudou de um formato para outro e automatizar a adaptação da configuração da MIN.

2. *Método para identificação automática da latência de medição ideal.* O último experimento apresentado mostrou que a latência de medição ideal depende da velocidade com que os servidores se tornam disponíveis para uso. Um estudo interessante é a criação de um método que seja capaz de se adaptar a diferentes latências.

3. *Método incremental.* Para criar a MIN, foram utilizadas informações de acesso históricas sob a forma de um histograma que representa o número de requisições enviadas ao sistema por hora. Contudo, pretendemos investigar como adaptar o PROFUSE para ser utilizado em sistemas que não disponibilizam tais informações, criando um método adaptativo e incremental que possa adicionar novas regras à base de conhecimento e atualizar as regiões nebulosas de cada variável linguística em tempo de execução.

Referências Bibliográficas

Abramson, D., Enticott, C., Altinas, I., "Nimrod/K: towards massively parallel dynamic grid workflows". **International Conference for High Performance Computing, Networking, Storage and Analysis**, p. 1-11, Austin, Texas, USA, 2008.

Anglano, C., Canonico, M., Guazzone, M., Botta, M., Rabellino, S., Arena, S., Girardi, G., "Peer-to-Peer Desktop Grids in the Real World: The ShareGrid Project". **8th IEEE International Symposium on Cluster Computing and the Grid**, p. 609-614, 2008.

Ardagna, C. A. et al., “Scalability Patterns for Platform-as-a-Service”, **IEEE 5th International Conference on Cloud Computing**, 2012.

Armbrust, M. et al., **A view of cloud computing**, ACM, v. 53, n. 4, p. 50–58, abr. 2010.

Banks, J., Carson, J. S., **Discrete event system simulation**, Englewood Cliffs, NJ, Prentice Hall, 1984.

Berkan; R. C., Trubatch, S. L., **Fuzzy systems design principles: building Fuzzy IF-THEN rule bases**, IEEE Press, 1997.

Bezerra, E., **Princípios de análise e projeto de sistemas com UML 2.** Rio de Janeiro, Elsevier;Campus, 2007.

Bocharov, P. P. et al., **Queueing Theory**, Ed. Walter de Gruyter, 2004.

Boeres, C., Sardiña, I., Drummond, L., "An efficient weighted bi-objective scheduling algorithm for heterogeneous systems", **Parallel Computing**, v. 37, n. 8 (ago.), p. 349-364.

Boloor, K. et al, “Dynamic Request Allocation and Scheduling for Context Aware Applications Subject to a Percentile Response Time SLA in a Distributed Cloud”, **IEEE Second International Conference on Cloud Computing Technology and Science** Anais. 2010.

Buyya et al. “Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility”. **Future Generation Computer Systems**, v. 25, n. 6, p. 599–616, jun. 2009.

Chang, F., Ren, J., Viswanathan, R., "Optimal Resource Allocation in Clouds". **IEEE 3rd International Conference on Cloud Computing**, p. 418-425, 2010.

Ching-Hsien Hsu, Tai-Lung Chen, "Adaptive Scheduling Based on Quality of Service in Heterogeneous Environments". **4th International Conference on Multimedia and Ubiquitous Engineering** , 2010.

Chen et al., **Analysis and Lessons from a Publicly Available Google Cluster Trace**, University of California at Berkeley, Berkeley, California: EECS Department, University of California, 14 jun. 2010.

Crovella, M. “Performance Evaluation with Heavy Tailed Distributions”. **Job Scheduling Strategies for Parallel Processing**, Lecture Notes in Computer Science. Springer Berlin Heidelberg, p. 1–10, 2001.

Curino et al., **Relational Cloud: A Database-as-a-Service for the Cloud,** MIT web domain, jan. 2011.

Dan, A. et al., “Web services on demand: WSLA-driven automated management”. **IBM Systems Journal**, v. 43, n. 1, p. 136–158, 2004.

Doyle, J. C., Francis, B. A.; Tannenbaum, A., **Feedback control theory**, Macmillan Publishing Company New York, 1992. v. 1

Dutta, S. et al. “SmartScale: Automatic Application Scaling in Enterprise Clouds”, **IEEE 5th International Conference on Cloud Computing**. jun. 2012

Dyachuk, D., Mazzucco, M., “On allocation policies for power and performance”, **2010 11th IEEE/ACM International Conference on Grid Computing** , 2010.

El-Khamra, Y., Kim, H., Jha, S., Parashar, M., "Exploring the Performance Fluctuations of HPC Workloads on Clouds". **2010 IEEE Second International Conference on Cloud Computing Technology and Science**, p. 383–387, Washington, DC, USA., 2010.

Fahringer, T., Prodan, R., Rubing Duan, Nerieri, F., Podlipnig, S., Jun Qin, Siddiqui, M., Hong-Linh Truong, Villazon, A., et al., "ASKALON: a Grid application development and computing environment". **6th IEEE/ACM International Workshop on Grid Computing**, p. 122-131, Seattle, Washington, USA, 2005.

Felipini, (2003). Plano de Negócios para empresas da InternetSEBRAE, , 27 jun. 2003. Disponível em: <http://www.biblioteca.sebrae.com.br/bds/bds.nsf/316586D5BF12E3B703256D520059A4A. Acesso em: 22 jun. 2013

Freitas; A. L., Parlavantzas, N.; Pazat, J.-L., “An Integrated Approach for Specifying and Enforcing SLAs for Cloud Services”. **IEEE 5th International Conference on Cloud Computing**, 2012.

Goiri et al. “Supporting CPU-based guarantees in cloud SLAs via resource-level QoS metrics”. **Future Generation Computer Systems**, v. 28, n. 8, p. 1295–1302, out. 2012.

Gorder, P. F., "Coming Soon: Research in a Cloud", **Computing in Science and Engg**., v. 10, n. 6, p. 6-10, 2008.

Greenberg et al., “The cost of a cloud: research problems in data center networks.” **SIGCOMM Comput. Commun. Rev.**, v. 39, n. 1, p. 68–73, dez. 2008.

Gurney, K., **An Introduction to Neural Networks**, CRC Press, 1997.

He, Q., Zhou, S., Kobler, B., Duffy, D., McGlynn, T., "Case study for running HPC applications in public clouds". **19th ACM International Symposium on High Performance Distributed Computing**, p. 395–401, New York, NY, USA, 2010.

Howe, B., Vo, H., Silva, C., Freire, J., "Query-driven visualization in the cloud with mapreduce". **Fourth Annual Workshop on Ultrascale VisualizationFourth Annual Workshop on Ultrascale Visualization**, Portland, Oregon, USA, 2008.

Hu; Sundara, S.; Srinivasan, J., “Supporting time-constrained SQL queries in Oracle” **VLDB Endowment**, Vienna, Austria, 2007.

Islam, M. A. ; Vrbsky, S. V.; Hoque, M. A., “Performance analysis of a tree-based consistency approach for cloud databases”. **International Conference on Computing, Networking and Communications**, 2012.

Jackson, K. R., Ramakrishnan, L., Muriki, K., Canon, S., Cholia, S., Shalf, J., Wasserman, H. J., Wright, N. J., "Performance Analysis of High Performance Computing Applications on the Amazon Web Services Cloud". **IEEE Second International Conference on Cloud Computing Technology and Science**, p. 159–168, Washington, DC, USA, 2010.

Kanungo et al., “An efficient k-means clustering algorithm: analysis and implementation”. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, v. 24, n. 7, p. 881–892, 2002.

Kazemian, H. B., “Two neuro-fuzzy control schemes for a traffic-regulating buffer in wireless technology”. **Information Sciences**, v. 181, n. 16, p. 3476–3490, 15 ago. 2011.

Kelly; D. L., Smith, C., **Bayesian Inference for Probabilistic Risk Assessment: A Practitioner’s Guidebook**, Springer, 2011.

Konstanteli et al, “Admission Control for Elastic Cloud Services” **IEEE 5th International Conference on Cloud Computing**, 2012.

Leitner et al., “Cost-Efficient and Application SLA-Aware Client Side Request Scheduling in an Infrastructure-as-a-Service” **IEEE 5th International Conference on Cloud Computing** , 2012.

Manuel Serra da Cruz, S., Tese de D.Sc., **UMA ESTRATÉGIA DE APOIO À GERÊNCIA DE DADOS DE PROVENIÊNCIA EM EXPERIMENTOS CIENTÍFICOS**”. PESC/COPPE/UFRJ, Rio de Janeiro, RJ, Brasil, 2011.

Mao, M., Humphrey, M., “Auto-scaling to minimize cost and meet application deadlines in cloud workflows”, **International Conference for High Performance Computing, Networking, Storage and Analysis**. nov. 2011

Mao, M.; Humphrey, M. A , “Performance Study on the VM Startup Time in the Cloud”, **IEEE 5th International Conference on Cloud Computing**, Jun. 2012.

Marinos, A., Briscoe, G., "Community Cloud Computing". **1st International Conference on Cloud Computing**, p. 472-484, Beijing, China., 2009.

Mazzucco; M., Dumas, M. (2011), “Reserved or On-Demand Instances? A Revenue Maximization Model for Cloud Providers”, **IEEE International Conference on. Cloud Computing**, 2011.

Mazzucco; M, Dyachuk, D.; Dikaiakos, M., “Profit-Aware Server Allocation for Green Internet Services”. **18th Annual IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems**, pp 277-284, 2010.

Mishra et al., “Towards characterizing cloud backend workloads: insights from Google compute clusters”. **SIGMETRICS Perform. Eval. Rev.**, v. 37, n. 4, p. 34–41, mar. 2010.

Patiño-Martinez, M., Jimenez-Peris, R., Kemme, B., et al., “Middle-R:Consistent database replication at the middleware level”. **ACM Trans. Comput.Syst**, pp. 375-423, 2005.

Pedone, F., Wiesmann M., Schiper, A., et al., “Understanding replication in databases and distributed systems”. **International Conference on Distributed Computing Systems**, pp. 464-474, 2000.

Ogasawara, E., Tese de D.Sc., **UMA ABORDAGEM ALGÉBRICA PARA WORKFLOWS CIENTÍFICOS COM DADOS EM LARGA ESCALA**, PESC/COPPE/UFRJ, Rio de Janeiro, RJ, Brasil, 2012.

Oliveira, D., Tese de D.Sc., **Uma Abordagem de Apoio à Execução Paralela de Workflows Científicos em Nuvens de Computadores**, PESC/COPPE/UFRJ, Rio de Janeiro, RJ, Brasil, 2012.

Orleans, L. F.; Furtado, P. N., “Fair Load-Balancing on Parallel Systems for QoS”, **2007 International Conference on Parallel Processing**. 2007.

Orleans, L. F; Oliveira, C. de; Furtado, P., “Task Assignment on Parallel QoS Systems.” **Web Information Systems Engineering,** p. 543–552, 2007.

Orleans, L. F.; Zimbrão, G.; Furtado, P., “Controlling the Behaviour of Database Servers with 2PAC and DiffServ”. **Database and Expert Systems Applications**. p. 779–790, 2008.

Ozsoyoglu et al., “Time-constrained query processing in CASE-DB”. **In: IEEE Transactions on Knowledge and Data Engineering**, v. 7, n. 6, p. 865–884, 1995.

Rajavel ; Mala, T., “Achieving Service Level Agreement in Cloud Environment Using Job Prioritization in Hierarchical Scheduling”. In: Satapathy; Avadhani, P.; Abraham, A. (Eds.). **Proceedings of the International Conference on Information Systems Design and Intelligent Applications,** Visakhapatnam, India, v. 132p. 547–554. January 2012.

Rokach, L., **Data Mining with Decision Trees: Theory and Applications**, World Scientific, 2007.

Sakr; Liu, A., “SLA-Based and Consumer-centric Dynamic Provisioning for Cloud Databases”. **IEEE 5th International Conference on Cloud Computing**, 2012.

Schroeder et al. (2006), “How to Determine a Good Multi-Programming Level for External Scheduling” **22nd International Conference on Data Engineering**, 2006.

Selfish, mean, impatient customers: New Thinking: Gerry McGovern. Disponível em: <http://www.gerrymcgovern.com/nt/2008/nt-2008-07-14-selfish.htm>. Acesso em: 17 ago. 2012.

Shah, Darshana, e Swapnali Mahadik., “QoS oriented failure rate-cost and time algorithm for compute grid”. **International Conference on Advances in Computing, Communication and Control**, New York, NY, USA: ACM, p. 264–267, 2009.

Sharma et al., “Modeling and synthesizing task placement constraints in Google compute clusters”. **2nd ACM Symposium on Cloud Computing.** New York, NY, USA: ACM, 2011.

Startup company. Disponível em: <http://en.wikipedia.org/w/index.php?title=Startup_company&oldid=559349275>. Acesso em: 14 jun. 2013.

Suleiman et al. “On understanding the economics and elasticity challenges of deploying business applications on public cloud infrastructure”. **Journal of Internet Services and Applications**, p. 1–21, 2012.

Wang. L., “Fuzzy systems are universal approximators” **IEEE International Conference on Fuzzy Systems**, p.p. 1163-1170, 1992.

Wang, L.; Mendel, J. M., “Generating fuzzy rules by learning from examples”. **IEEE Transactions on Systems, Man and Cybernetics**, v. 22, n. 6, p. 1414–1427, 1992.

Yung-Han Chen; Chung-Ju Chang; Ching Yao Huang, “Fuzzy Q-Learning Admission Control for WCDMA/WLAN Heterogeneous Networks with Multimedia Traffic”, **IEEE Transactions on Mobile Computing**, v. 8, n. 11, p. 1469–1479, nov. 2009.

Zhang; Q., Cheng, L.; Boutaba, R., “Cloud computing: state-of-the-art and research

challenges”. **Journal of Internet Services and Applications**, v. 1, n. 1, p. 7–18, 2010.

APÊNDICE A – CLOUD-SIMULATOR

Como forma de testar os algoritmos contidos no PROFUSE e averiguar a eficiência e a eficácia de seus algoritmos, um Simulador de Eventos Discretos (SED), (Banks; Carson, 1984), foi desenvolvido.

Este apêndice descreve brevemente a arquitetura de um SED e com detalhes a arquitetura do simulador.

Simulação com Fila de Eventos Discretos

Um SED se baseia na modelagem de um sistema como uma sequência discreta de *eventos* que podem ocorrer em um sistema ao longo do tempo, onde cada evento altera o *estado* do sistema. Por exemplo, a chegada de uma consulta SQL em um Sistema Gerenciador de Banco de Dados pode ser considerada um evento para este sistema. Cada evento atua em uma componente do sistema, e cada componente pode possuir diversas medidas associadas a si, como *tempo de chegada*, *duração*, *restrições de QoS*, *tempo de espera na fila de requisições*, *tamanho da fila de requisições em sua chegada*, entre outras. Repare que algumas destas medidas não podem ser definidas no momento de criação do evento, devendo ser coletadas ao longo do seu ciclo de vida dentro do simulador.

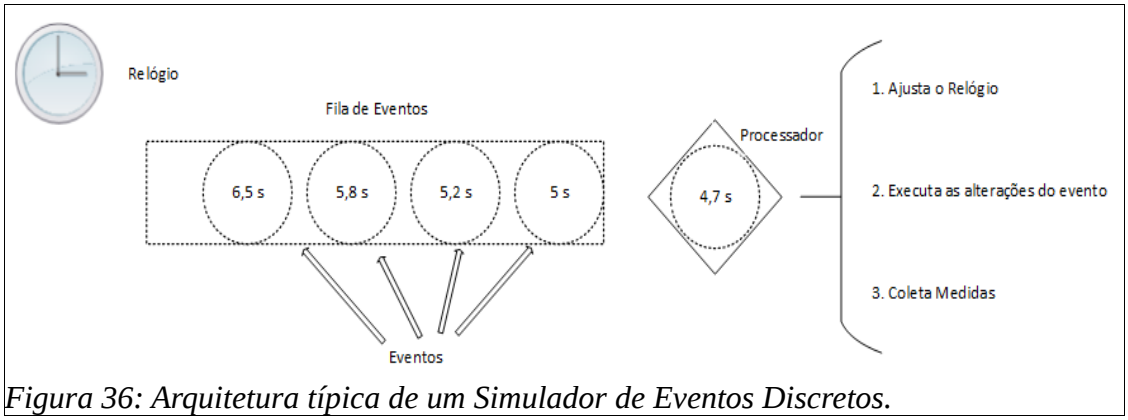


Figura 36: Arquitetura típica de um Simulador de Eventos Discretos.

Arquitetura Típica de um SED

A arquitetura básica de um SED contém as seguintes componentes:

1. *Eventos*: um evento pode ser entendido como toda e qualquer condição que altere o estado de uma ou mais componentes do sistema sendo simulado. Cada evento ocorre em um determinado tempo/hora. Entre dois eventos consecutivos nenhuma alteração no sistema é realizada, daí o tempo de simulação pode saltar diretamente de um evento para o outro.

2. *Relógio*: como forma de gerenciar o tempo de simulação, um dos componentes principais de um SED é o relógio. Sua responsabilidade é manter atualizado o tempo decorrido desde que a simulação iniciou.

3. *Fila de Eventos*: uma fila ordenada que contém os eventos ainda não processados. Os eventos devem estar ordenados cronologicamente, logo esta não é uma fila do tipo "Primeiro a Entrar, Primeiro a Sair", mas sim uma Fila de Prioridade.

4. *Processador da Fila de Eventos*: cada evento realiza uma alteração em alguma componente do sistema. Assim, quando chega o momento de sua execução, o evento é retirado da fila pelo processador e as suas alterações são executadas.

5. *Gerador de Números Aleatórios*: um SED necessita de um ou mais Geradores de Números Aleatórios (GNA), cujas função é, basicamente, associar tempos a eventos. Por exemplo, utiliza-se um GNA para gerar o tempo em que uma nova requisição chegará ao sistema, outro GNA para gerar o tempo de duração da requisição e assim por diante. Cada GNA gera um número seguindo uma distribuição estatística.

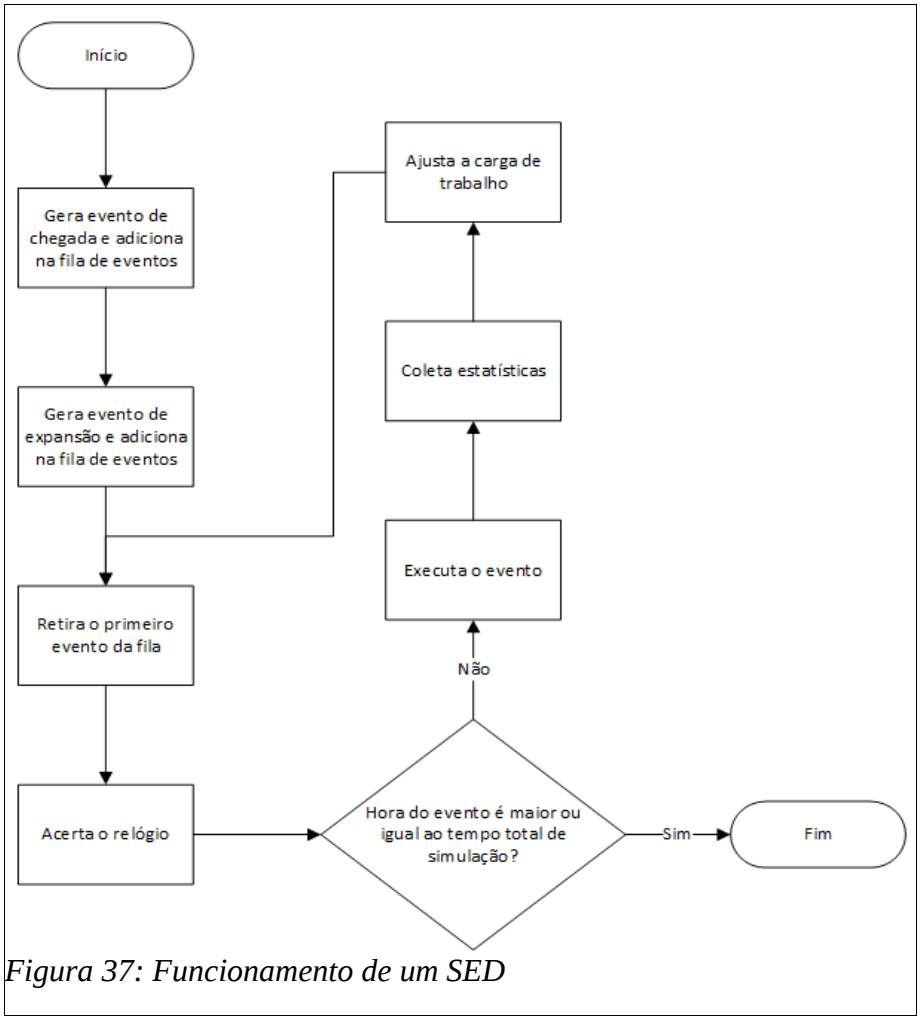


Figura 37: Funcionamento de um SED

6. *Tempo Total de Simulação*: é a condição de parada do simulador. Quando o relógio atingir um valor maior ou igual ao tempo total de simulação, o processo pára. A figura 23 exibe a arquitetura típica de um SED, com os relacionamentos entre os componentes. Repare que na Fila de Eventos, os eventos estão ordenados de acordo com o seu tempo de execução, enquanto a figura 25 exibe o funcionamento de um SED. O Cloud-Simulator segue esta arquitetura.

Eventos Típicos

Eventos comuns em um SED são:

1. *Chegada de uma nova requisição*: representa o momento em que uma nova requisição

é recebida pelo sistema. Sua componente associada é a requisição em si, que captura o momento (tempo) de sua chegada. De um modo geral, quando um evento de chegada é processado, sua primeira instrução é a geração do próximo evento de chegada (figura 38).

2. *Término de uma requisição*: este evento indica que uma requisição já foi processada e que suas estatísticas devem ser armazenadas (figura 26).

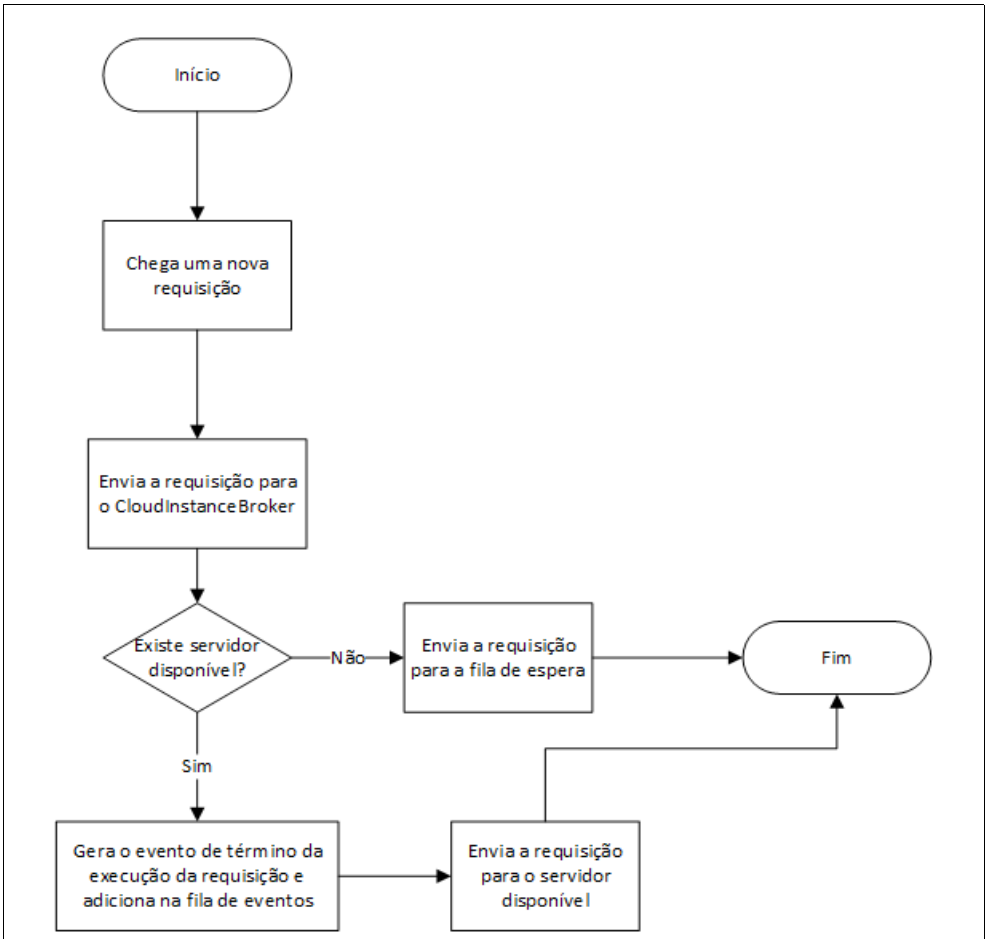


Figura 38: Tratamento do evento de chegada de uma requisição.

No Cloud-Simulator, os seguintes eventos foram detectados, além dos típicos:

1. *Expansão*: representa o tempo entre as chamadas à componente do PROFUSE que monitora o sistema e invoca a rotina de expansão (figura 27).

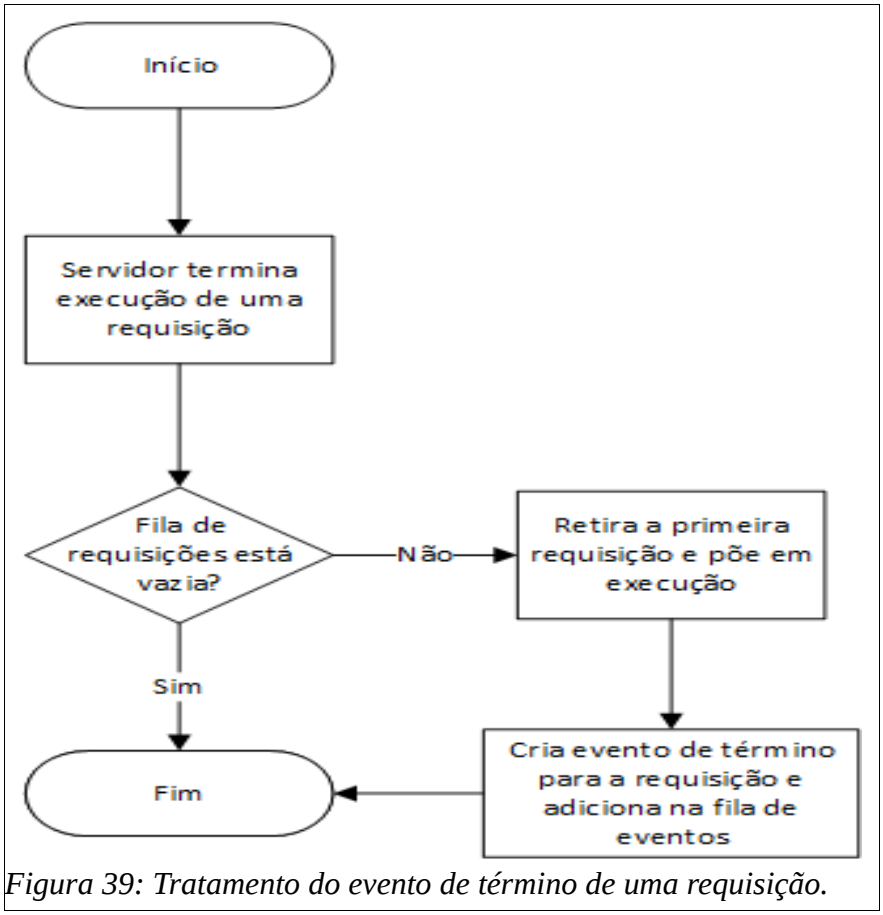


Figura 39: Tratamento do evento de término de uma requisição.

2. *Ligar um novo servidor*: quando um servidor adicional é solicitado ao provedor IaaS, este não fica disponível imediatamente. Este evento representa o tempo que um novo servidor leva para se tornar disponível, ou seja, até ser incluído efetivamente no *array* de instâncias.

3. *Desligar um Servidor*: da mesma forma que o item anterior, este evento representa o tempo que um servidor leva para ser desligado, isto é, ser devolvido ao provedor IaaS.

Arquitetura Simulada

Além das componentes básicas, descritas anteriormente, um SED possui módulos que representam objetos concretos do sistema sendo simulado. Por exemplo, um SED utilizado para simular um Sistema Gerenciador de Banco de Dados pode possuir

representações do otimizador de consultas SQL, do processador de consultas, discos rígidos, etc.. O Cloud-Simulator possui representações das seguintes componentes:

1. *Servidores*: representam as instâncias que podem ser alugadas junto ao provedor IaaS. Possuem 3 tipos ou configurações diferentes: 1, 2 ou 4 processadores. Cada servidor possui um custo de instância-hora diferente e o número de processadores indica o nível máximo de paralelismo que pode ser atingido.

2. *Gerenciador de alocação de tarefas*: representa o módulo que escolhe, dentre os servidores contidos no *array*, qual vai executar a próxima requisição. Caso o sistema esteja ocupado, isto é, todos os processadores de todas as instâncias estão ocupados,

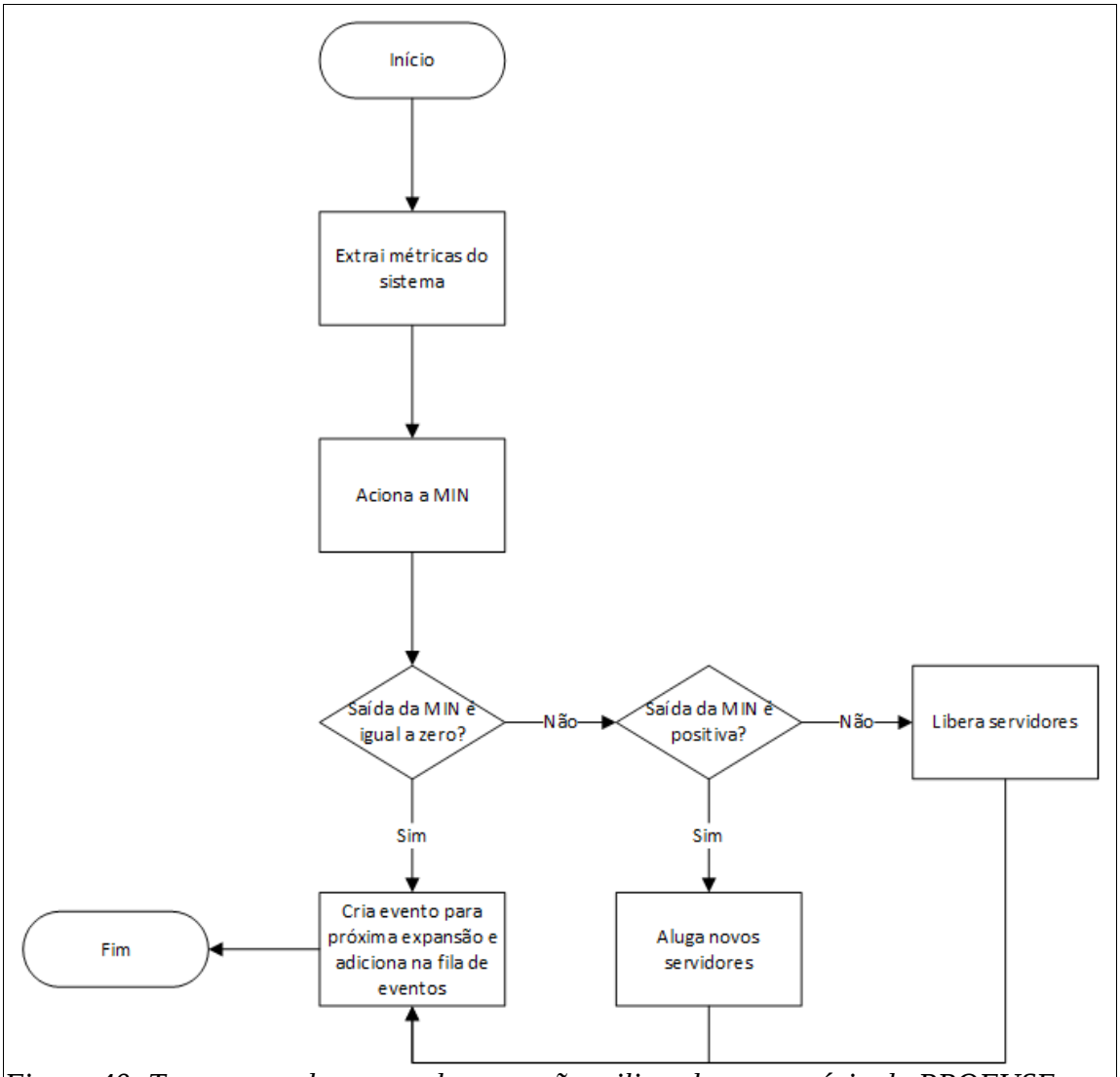


Figura 40: Tratamento do evento de expansão utilizando a estratégia do PROFUSE.

sinaliza que não há como processar a requisição.

3. Gerenciador do modelo de expansão: quando o evento de expansão é processado, um dos modelos de expansão é invocado, como forma de verificar se é necessário aumentar ou reduzir o array de instâncias. Possui 3 modelos de expansão implementados: (i) Número fixo de Instâncias, que não executa qualquer ação de expansão, simulando um sistema paralelo tradicional e mantendo o número de servidores fixo, independente da carga de trabalho; (ii) Utilização Fixa, que é um modelo de expansão reativo, onde o objetivo é manter a utilização total do sistema fixa. Com isso, a cada mudança na carga de trabalho, novas instâncias são adquiridas ou liberadas para manter a utilização constante; e (iii) PROFUSE, que representa o WOM, descrito na seção 4.4..

4. *Broker:* esta é uma componente híbrida, realizando diversos papéis. É responsável por receber a requisição, verificar se há um servidor disponível para executá-la, enviá-la para a fila de espera caso todos os servidores estejam ocupados e retirar requisições da fila, pondo-as em execução sempre que um processador se tornar disponível. Realiza, ainda, o papel do SAM (seção 4.5.), contendo os algoritmos para a aquisição e liberação de servidores. Finalmente, é a componente que contém o *array* de instâncias.

5. *Carga de Trabalho:* esta componente representa a carga de trabalho contida no Histograma de Acessos. A cada ciclo do simulador, isto é, a cada evento executado, esta componente é verificada e, caso seja necessário, a taxa de chegadas de novas requisições é alterada. Sua implementação padrão representa uma CdT de um sistema real. Outras CdTs típicas de sistemas hospedados em nuvem foram implementadas, representando comportamento estável e crescente.

A figura 41 mostra o diagrama de classes da arquitetura simulada pelo Cloud-Simulator.

Eliminação da Fase Transiente

Um dos cuidados que se deve ter ao utilizar um SED é com a *fase transiente*, isto é, o período inicial da simulação onde o sistema ainda não está estável e, por isso, seus resultados não devem ser levadas em consideração.

Existem dois motivos principais para a existência da fase transiente, onde o primeiro é o fato de o sistema simulado estar vazio, com seus servidores ociosos e fila de requisições vazia, condições raramente encontradas.

O segundo motivo está relacionado ao fato de os tempos iniciais dos eventos não serem conhecidos de antemão. Como resultado, o conjunto inicial de eventos inseridos da fila de eventos pode não possuir tempos de chegada representativos do sistema real.

Desta forma, deve-se ignorar as estatísticas oriundas dos primeiros eventos gerados, como forma de não adicionar incorreções nas métricas geradas pelo sistema. O Cloud-Simulator descarta todas as estatísticas geradas durante a primeira hora de simulação.

Analisador de Log

Esta componente tem a função de ler um arquivo que contenha um *log* em um formato específico (descrito na tabela 4, página 59) e gerar o Histograma de Acessos. Sua execução combina as requisições contidas no *log* com um outro arquivo-texto que possui a granularidade a ser apresentada no histograma.

Parâmetros Iniciais

O Cloud-Simulator ainda possui os parâmetros abaixo, com os seus devidos valores padrão:

1. *Tempo de Simulação em segundos*: 86400 segundos (1 dia).
2. *Duração de uma requisição*: 0,5 segundos (taxa de 2 requisições por segundo)
3. *Exibir log:* sim.
4. *Tempo entre as chamadas do PROFUSE*: 120 segundos (2 minutos)
5. *Tamanho inicial do array de instâncias*: 5.

6. *Tamanho máximo do array de instâncias*: 20.
7. *Custo de cada tipo de instância*: $0,02 para instâncias com um processador; $0,4 para instâncias com dois processadores e $2 para instâncias com quatro processadores.
8. *Utilizar ganchos*: sim.
9. *Tempo máximo de uma requisição*: 4 segundos.

Para implementação de um protótipo, é possível utilizar “programas de teste” (*toy programs*) para determinar o valor do item (2), ou seja, a duração de uma requisição, utilizando a mesma solução adotada em (Oliveira, 2012).

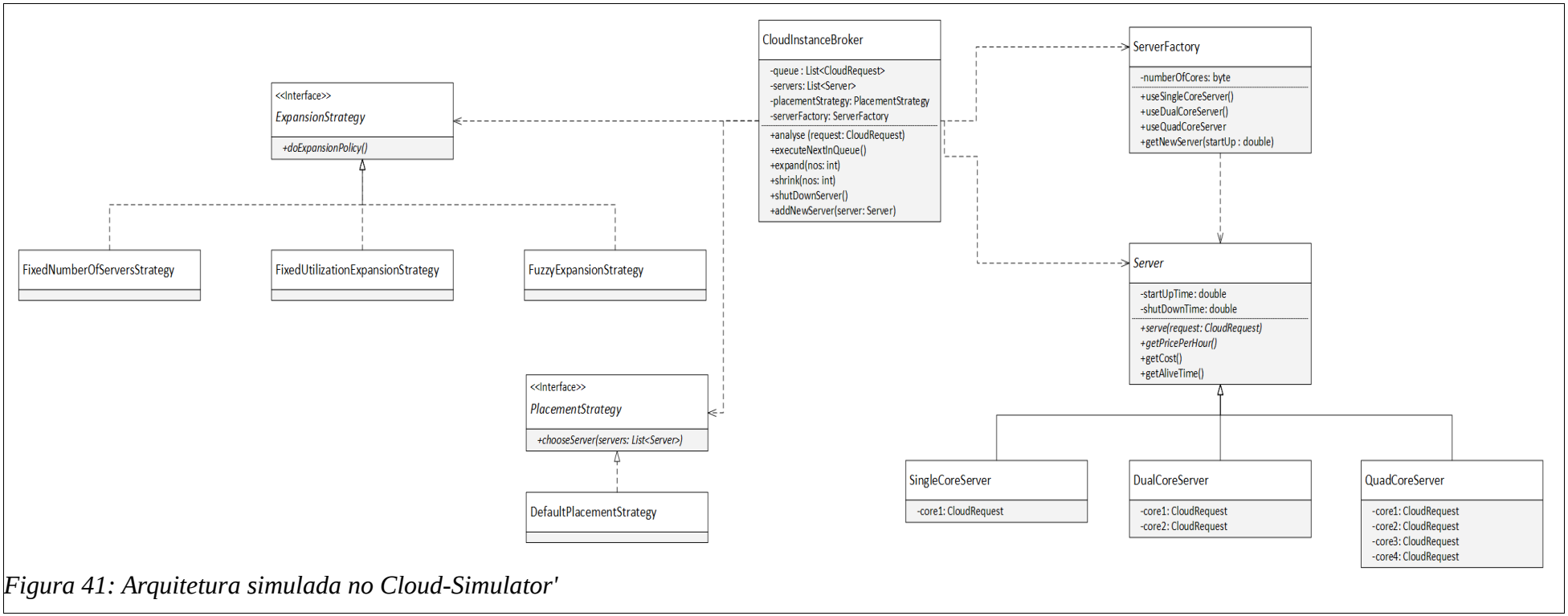


Figura 41: Arquitetura simulada no Cloud-Simulator'