Universidade Federal de Campina Grande
Centro de Engenharia Elétrica e Informática
Coordenação de Pós-Graduação em Ciência da Computação

SaaSim - Um framework para Simulação de Software as a Service

Ricardo Araújo Santos

Campina Grande, Paraíba, Brasil
Setembro - 2012

Universidade Federal de Campina Grande
Centro de Engenharia Elétrica e Informática
Coordenação de Pós-Graduação em Ciência da Computação

SaaSim - Um framework para Simulação de Software as a Service

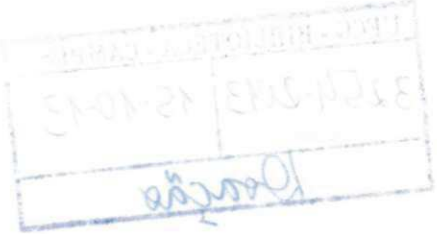
Ricardo Araújo Santos

Dissertação submetida à Coordenação do Curso de Pós-Graduação em Ciência da Computação da Universidade Federal de Campina Grande - Campus I como parte dos requisitos necessários para obtenção do grau de Mestre em Ciência da Computação.

Área de Concentração: Ciência da Computação
Linha de Pesquisa: Metodologias e Técnicas da Computação

Raquel Vigolvino Lopes
(Orientadora)

Campina Grande, Paraíba, Brasil



S237s Santos, Ricardo Araújo.
SaaSim : um framework para simulação de software As a Service / Ricardo Araújo Santos. - Campina Grande, 2012.
66 f.

Dissertação (Mestrado em Ciência da Computação) - Universidade Federal de Campina Grande, Centro de Engenharia Elétrica e Informática, 2012.
"Orientação : Profª. Drª. Raquel Vigolvino Lopes".
Referências.

1. Simulação. 2. Computação na Nuvem. 3. Framework. 4. Aplicação Web. 5. Dissertação - Ciência da Computação. I. Lopes, Raquel Vigolvino. II. Universidade Federal de Campina Grande - Campina Grande (PB). III. Título

CDU 004.65(043)

"SAASIM - UM FRAMEWORK PARA SIMULAÇÃO DE SOFTWARE AS A SERVICE"

RICARDO ARAUJO SANTOS

DISSERTAÇÃO APROVADA EM 10/09/2012

RAQUEL VIGOLVINO LOPES, D.Sc
Orientador(a)

LÍVIA MARIA RODRIGUES SAMPAIO CAMPOS , D.Sc
Examinador(a)

AYLA DÉBORA DANTAS DE SOUZA REBOUÇAS, D.Sc
Examinador(a)

CAMPINA GRANDE - PB

Resumo

Com a difusão do paradigma de computação na nuvem, aplicações Web passaram a ser ofertadas como um serviço no modelo de Software as a Service (SaaS). Nesse modelo o contratante da aplicação paga uma assinatura para usá-la enquanto o provedor continua a arcar com os custos de manutenção da infraestrutura que a executa. Com isso vários dos problemas relacionados à gerência de aplicações Web tem sido revisitados à luz dos novos modelos de mercado de infraestrutura e aplicações. No entanto, percebe-se a dificuldade dos pesquisadores na realização de experimentos de longa duração, e que considerem o modelo de negócio das aplicações SaaS, para avaliar soluções propostas para esses problemas, levando-os a partir para o uso de ambientes de simulação. Essa dissertação investiga um simulador que possa ser usado para avaliar, em cenários de longa duração, mecanismos de gerência de recursos usados para executar aplicações Web horizontalmente escaláveis ofertadas no modelo de SaaS, evitando o alto custo de experimentos reais e oferecendo uma metodologia que permita validar os modelos incorporados ao simulador atingindo um nível de confiança esperado dos resultados obtidos com experimentos de simulação. Nesse contexto propomos o SaaSim, um framework para simulação de aplicações Web horizontalmente escaláveis incorporando o modelo de negócio próprio do mercado de SaaS. A implementação é flexível e aplica a metodologia de validação proposta para o modelo conceitual usado. Por fim apresentamos uma extensão simples do framework para avaliação de um algoritmo de provisionamento dinâmico, num cenário de duração de um ano, para uma aplicação de uma camada com modelo de negócio similar ao BigCommerce.

Abstract

With the popularisation of cloud computing paradigm, Web applications started to being delivered as a service in a model known as Software as a Service (SaaS). In such model, contractors pay for a subscription while the provider keeps responsible for execution costs and infrastructure management. As a consequence, several problems related to Web application management have been revisited according to new market models of infrastructure and application in the cloud. However, it is notable that researchers have difficulties in running long duration measurement experiments in order to evaluate new solutions considering SaaS business model, leading them to fall back on simulation experiments. This work designs a simulator that can be used to evaluate, in long duration scenarios, resource management techniques used to run horizontally scalable Web applications delivered as SaaS, avoiding costs associated with real measurement experiments while offering a methodology to allow the validation of models implemented with such simulator. We present SaaSim, a framework for horizontally scalable Web application simulation which focus on SaaS business model. This implementation is extensible and applies the validation methodology proposed to the single-tier application model implemented. We still present a simple framework extension for the evaluation of a dynamic provisioning algorithm for managing a single-tier Web application with a business model inspired by BigCommerce, in a long duration scenario of one year.

Agradecimentos

Aos meus pais pelo incentivo diário e pelas lições que carrego comigo. À minha orientadora Raquel pelas diversas discussões proveitosas e pelo profissionalismo na construção desse trabalho. Ao meu irmão de orientação David Candeia pela parceria nessa jornada, e que ela continue nos trazendo bons frutos. Aos amigões da sala Carvalheira, em especial a Marcus, Paulo e Lília, pelo suporte e empolgação nos longos brainstorms e cervejas pela madrugada; e a Renato e Degas que estiveram presentes no início do trabalho contribuindo com discussões proveitosas. Aos vários colegas e professores do Laboratório de Sistemas Distribuídos com os quais pude contar ao longo desses quase 6 anos de cooperação. Aos amigos que compreenderam a importância desse trabalho para mim e acreditaram que eu fosse capaz. E aos que não estão mais aqui, pelas boas memórias que enchem o coração.

Sumário

Lista de Siglas e Abreviaturas

CRM *Customer Relationship Management*

FCFS *First Come, First Served*

HTTP *Hypertext Transfer Protocol*

IAAS *Infrastructure as a Service*

JVM *Java Virtual Machine*

PAAS *Platform as a Service*

QoS *Quality of Service*

SAAS *Software as a Service*

SOA *Service Oriented Architecture*

TDD *Test-driven Development*

TI Tecnologia da Informação

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

1.1 Motivação e Relevância

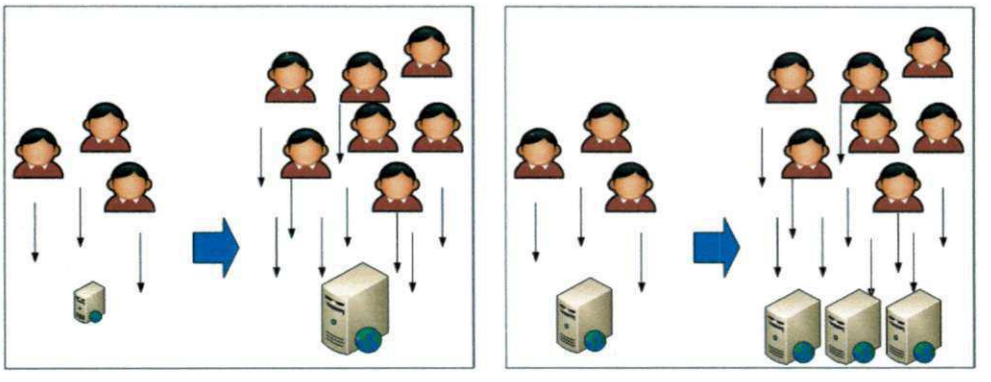
Ao longo dos anos, a crescente importância das aplicações Web para empresas e usuários tem sido motivação para pesquisas nas áreas de gerência [1; 2; 3] e planejamento de capacidade [4; 5]. Com isso, pesquisadores e desenvolvedores tem empenhado esforços na criação e implementação de novas soluções a fim de utilizar eficientemente os recursos disponíveis atendendo adequadamente os usuários e provendo qualidade de serviço (do inglês *Quality of Service - QoS*) desejada.

A qualidade de serviço da aplicação Web é estabelecida através de um contrato chamado de Acordo de Nível de Serviço (do inglês *Service Level Agreement - SLA*) que é composto, dentre outras informações, por indicadores de serviço (e.g. tempo de resposta, disponibilidade média no mês e tempo médio para restabelecimento do serviço), o nível mínimo de cada indicador esperado pelo contratante e o valor da multa paga pelo provedor caso o serviço viole os limiares de QoS estabelecidos no SLA. A infração do SLA implica por um lado em déficit para a empresa e por outro usuários insatisfeitos com a aplicação/serviço.

Quando a quantidade de requisições submetidas à aplicação é maior do que o que a infraestrutura disponível é capaz de processar, entra em cena o gerente da aplicação [1] que pode, entre outras ações, aumentar a quantidade de recursos disponíveis para a aplicação para evitar a violação do SLA. O modo de aumentar os recursos da aplicação

[1] explicar antes...

define o tipo de escalabilidade que ela implementa, como visto na figura 1.1: (a) verticalmente, quando o aumento na demanda pela aplicação é resolvido promovendo um aumento na capacidade do hardware do servidor (memória, disco, etc.) executando a aplicação; e (b) horizontalmente, quando mais servidores são adicionados à infraestrutura com o objetivo de dividir o atendimento da demanda entre si. Nem todas as aplicações Web são horizontalmente escaláveis, mas nessa dissertação estaremos concentrados em aplicações distribuídas que são horizontalmente escaláveis.



(a) Escalabilidade Vertical (b) Escalabilidade Horizontal

Figura 1.1: Modelos de escalabilidade

Com a difusão do paradigma de computação na nuvem, vários dos problemas estudados tem sido revisitados à luz dos novos modelos de mercado de infraestrutura e aplicações [6; 7; 8]. Aplicações Web passaram a ser ofertadas como um serviço no modelo de Software as a Service (SaaS). Nesse modelo o contratante paga uma assinatura para usar a aplicação enquanto o provedor continua a arcar com os custos de manutenção da infraestrutura que a executa. Com isso, novas soluções começaram a ser desenvolvidas para tirar proveito das características do mercado de SaaS. Dois problemas são tipicamente enfrentados por pesquisadores que se propõem a estudar tais soluções: i) a falta de cargas de trabalho reais que possam ser usadas em experimentos para avaliar a nova solução proposta; e ii) os experimentos de longa duração são caros e complexos, necessitando de tempo, infraestrutura adequada e aplicações. O problema da falta de cargas de trabalho coletadas de sistemas reais não é o foco deste trabalho. Esse problema é fortemente dependente do tipo de aplicação estudada. Consideramos aqui que existem pesquisas que caracterizaram requisições de aplicações específicas,

como Menascé et al.[9] e Arlitt et al.[10], que estudaram serviços de comércio eletrônico, e essas invariantes podem ser usadas para a geração de cargas sintéticas.

O segundo problema é o alto custo de experimentos reais de longa duração demandando tempo, infraestrutura adequada e aplicações reais que possam ser usadas nos experimentos de medição. Para lidar com esse problema, os pesquisadores realizam experimentos menores quando possível e a análise é complementada com experimentos de simulação. No entanto, a falta de um ambiente de simulação validado e verificado tem estimulado cada pesquisador a desenvolver seu próprio simulador [11; 12; 13; 7; 14; 15], diminuindo o reuso dos ambientes de simulação e, por consequência, colocando em risco a confiança nos resultados obtidos.

Entendemos que um simulador facilmente extensível e associado a uma metodologia de validação dos resultados é importante no contexto dessas pesquisas por permitir que, uma vez que se crie uma cultura de reutilização entre os pesquisadores, possamos ter uma maior confiança nos resultados apresentados. Outro benefício direto é a possibilidade de redução de custos na reprodução de experimentos de outros pesquisadores com o intuito de comparar técnicas que se proponham a resolver problemas similares. Além disso, existe a redução do trabalho de implementação, sendo necessária somente a implementação de componentes particulares de novas soluções de gerência de recursos estudadas.

1.2 Problema Investigado

Essa dissertação investiga um simulador que possa ser usado para avaliar, em cenários de longa duração, mecanismos de gerência de recursos usados para executar aplicações Web horizontalmente escaláveis ofertadas no modelo de SaaS, evitando o alto custo de experimentos reais e oferecendo uma metodologia que permita validar os modelos incorporados ao simulador atingindo um nível de confiança esperado dos resultados obtidos com experimentos de simulação.

O design do simulador proposto neste trabalho tem três objetivos principais:

1. capturar elementos fundamentais e significativos do ambiente real encontrando o balanceamento entre um modelo geral e acurácia dos resultados. Modelos

acurados demais acabam restringindo sua utilidade a um grupo muito específico de cargas de trabalho [16];

2. associar o design a uma metodologia de validação para garantir confiança nos resultados obtidos;

3. permitir que novos modelos, cenários e tipos de aplicações sejam investigados com menor esforço de implementação.

O SaaSim é uma implementação inicial do design proposto. Sua arquitetura permite que vários componentes possam ser estendidos e adequados a novas soluções de gerência de capacidade incluindo algoritmos para provisionamento dinâmico, planejamento de capacidade, balanceamento de carga e controle de acesso. O modelo de aplicação Web de uma camada horizontalmente escalável implementado no SaaSim possui um modelo de valoração típico de aplicações SaaS atualmente disponíveis no mercado.

Por fim, uma metodologia de validação foi seguida e os resultados produzidos pelo simulador puderam ser avaliados em comparação com experimentos reais de medição num ambiente controlado. Essa metodologia pode ser seguida para implantar novos modelos de aplicação Web no simulador.

1.3 Contribuições

Neste contexto, esse trabalho de dissertação apresenta as seguintes contribuições:

1. o design de um simulador de aplicações de longa duração horizontalmente escaláveis que pode ser usado na avaliação de soluções para problemas comuns na área de gerência de capacidade;

2. implementação e teste de um framework de simulação seguindo o design levantado;

3. modelagem de um fator de correção para o modelo aplicação Web implementado no simulador;

4. validação operacional do simulador com a comparação de cenários simulados e experimentos controlados de medição;

5. descrição de uma extensão do simulador para análise de um algoritmo de gerência de capacidade num cenário de longa duração.

1.4 Organização do Documento

Após uma revisão da literatura apresentada no Capítulo 2, apresentamos os diferentes aspectos da gerência de recursos para aplicações SaaS que desejamos capturar no simulador no Capítulo 3. O capítulo 4 apresenta o design do simulador enquanto no Capítulo 5 são apresentados detalhes específicos e limitações da implementação realizada. No capítulo 6, serão apresentados a metodologia de verificação e validação e uma extensão do simulador é apresentada no Capítulo 7 E por fim, conclusões e trabalhos futuros são discutidos no capítulo 8.

Capítulo 2

Fundamentação Teórica

2.1 Gerência Automática de Aplicações Web

Hoje em dia, as aplicações Web variam desde páginas simples de conteúdo dinâmico até soluções mais complexas de edição de texto, planilhas e imagens, sites de comércio eletrônico e soluções para *Business Intelligence*, por exemplo. Uma de suas principais vantagens está na facilidade de manutenção e atualização, uma vez que quem provê a aplicação detêm a responsabilidade por toda a gerência e o acesso à aplicação é realizado de forma muito simples através de um navegador universal. Aplicações Web costumam ser divididas em três camadas [17] (Figura 2.1):

- a camada de apresentação, na qual os elementos visuais são processados para exibição nos diversos dispositivos usados pelos clientes. Separar esse processamento da lógica de negócio implica em aplicações mais robustas e fáceis de se adaptar a novos dispositivos de exibição;

- a camada de aplicação, responsável pela lógica de negócio. O trabalho realizado pelas máquinas nessa camada envolve o processamento de requisições recebidas da camada de apresentação e geração de resultados a partir de consultas aos dados dos servidores transacionais. O Apache Tomcat[1] é um exemplo de servidor para aplicações desenvolvidas usando as tecnologias *Java Servlets* e *JavaServer*

[1] `http://tomcat.apache.org`

Pages enquanto o JBoss[2] é um exemplo de servidor para aplicações desenvolvidas para *Java Platform Enterprise Edition*;

- a camada de dados, implementada de maneira mais trivial com o uso de servidores transacionais e bancos de dados. Das três camadas essa é mais difícil de se prover escalabilidade horizontal, pois implicaria em replicação e manutenção da consistência dos dados persistidos.

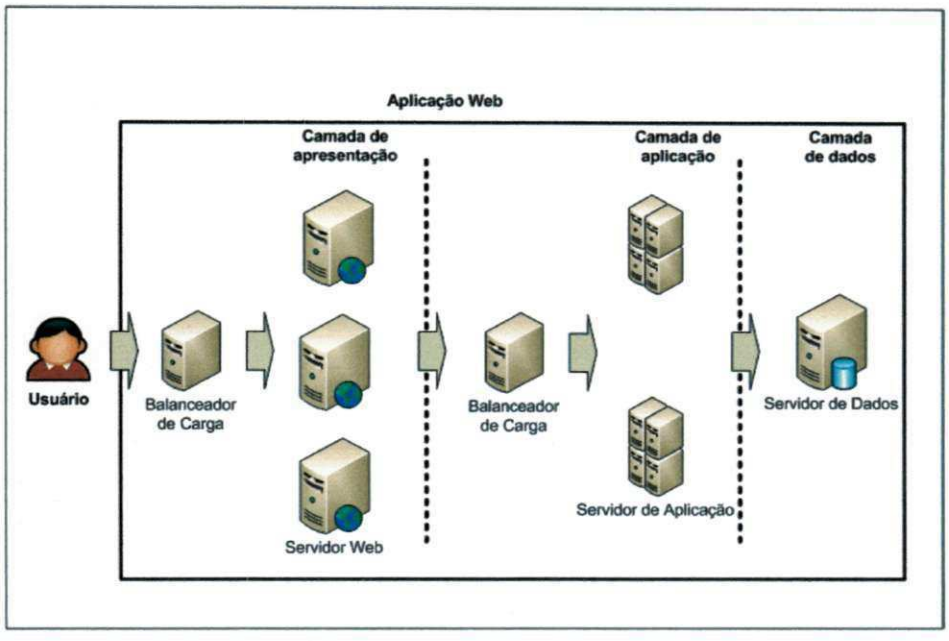


Figura 2.1: Aplicações Web em três camadas

Uma vantagem direta desse modelo de arquitetura é a facilidade de prover escalabilidade horizontal. Servidores podem ser adicionados e removidos facilmente das camadas de apresentação e aplicação realizando balanceamento da carga de trabalho. Além disso, mais servidores por camada implica no aumento da disponibilidade e na vazão atendida pela aplicação.

Essas aplicações apresentam algumas características peculiares. Primeiramente, elas são serviços que executam indefinidamente à espera de solicitações vindas de usuários. Em segundo lugar, a qualidade de serviço dessas aplicações, em termos de disponibilidade e tempo de resposta, desempenham um papel muito importante

[2] `http://www.jboss.org`

dado o tipo de interação recorrente dos usuários com a aplicação. Finalmente, a carga a que essas aplicações são submetidas é altamente variável ao longo do tempo [1; 18] e exibe padrões de acordo com a hora do dia ou dia da semana [19], que facilitam o uso de mecanismos de predição.

Há alguns anos, a importância das aplicações Web no meio empresarial vem crescendo bastante. O uso de tais aplicações abrange desde a gerência de processos internos da empresa até a oferta de serviços para usuários finais desempenhando papel fundamental na geração de lucros. No cenário de comércio eletrônico, por exemplo, falhas no serviço como indisponibilidade e baixo desempenho da aplicação Web, ou até o mal planejamento da infraestrutura que a executa, podem ser responsáveis por uma queda na venda de produtos e consequentemente nos lucros da empresa. Dessa forma, há uma grande preocupação em gerenciar a execução de aplicações lidando com situações como aumento ou diminuição repentina na quantidade de requisições fazendo uso mais eficiente da infraestrutura disponível.

De acordo com a taxonomia proposta por Guitart et al.[20], as soluções para gerência automática de aplicações Web podem ser divididas de maneira não excludente, com relação ao foco de atuação, em:

- Escalonamento de requisições: quando requisições concorrentes são ordenadas segundo algum critério para obter um melhor aproveitamento dos recursos utilizados na execução;

- Diferenciação de serviço: permite tratar classes de usuários diferentemente em caso de sobrecarga, atribuindo níveis de qualidade de serviço diferentes;

- Controle de admissão: caso extremo da diferenciação de serviço, no qual o serviço é negado a uma parcela menos importante de clientes;

- Provisionamento dinâmico de recursos: diz respeito a formas de aumentar os recursos utilizados pela aplicação em caso de sobrecarga e de diminuir ou transferir para outras aplicações no caso de subutilização;

- Degradação de serviço: para evitar negar serviço a algum cliente durante a sobrecarga, as soluções nessa categoria baixam a qualidade do serviço oferecido

temporariamente.

Com relação ao tipo de avaliação usada para análise das soluções propostas, podemos agrupar trabalhos relacionados entre os que: (i) realizaram experimentos de medição em ambientes controlados; (ii) usaram pacotes de software conhecidos pela comunidade para realizar experimentos de simulação; e (iii) desenvolveram seus próprios simuladores sem uso de nenhum pacote ou framework de simulação.

Dentre os trabalhos relevantes na categoria de provisionamento dinâmico de recursos, alguns utilizam experimentos reais geralmente de pequena escala como o modelo de provisionamento dinâmico com algoritmo reativo proposto por Menascé et al. [21], que foi avaliado num ambiente de cluster com carga de trabalho gerada usando o TPC-W [3] e experimentos reais com amostras de 30 minutos de duração. Pradhan et al. [22], por sua vez, propôs um método reativo para configurar automaticamente sistemas de provisionamento dinâmico e o avaliou utilizando experimentos reais com 100 minutos de duração para analisar o comportamento do algoritmo submetido a uma carga sintética. Em outro exemplo, cargas sintéticas foram submetidas às aplicações Rubis [4] e Rubbos [5] numa infraestrutura real de cluster com virtualização para avaliar: i) um modelo analítico para provisionamento dinâmico [23] em experimentos de duração de 30 minutos; e ii) um algoritmo de provisionamento combinado a um controle de admissão baseado em histórico em cenários com no máximo 60 minutos de duração [24].

Na segunda categoria de trabalhos, destacam-se: i) Ranjan et al. [1], que propõe um algoritmo de provisionamento baseado em utilização de máquina que foi avaliado usando um simulador baseado no YACSIM [6] alimentado com cargas reais não disponíveis publicamente de um sistema de comércio eletrônico e um sistema de busca; e ii) Chandra et al. [2], que propôs um modelo que combina predição usando séries temporais e caracterização da carga de trabalho para fazer alocação dinâmica de recursos de um cluster compartilhado entre aplicações Web. Nesse caso, a avaliação foi realizada através de um simulador implementado usando os pacotes NetSim [7] e DASSF [8].

[3] http://www.tpc.org/tpcw

[4] http://rubis.ow2.org

[5] http://jmob.ow2.org/rubbos.html

[6] http://softlib.rice.edu/rppt.html

[7] http://www-net.cs.umass.edu/fluidsim/archive.html

[8] http://users.cis.fiu.edu/~liux/research/projects/dassf/index.html

Por fim, na terceira categoria, Villela et al. [11] propõe três soluções baseadas num modelo de redes filas $M/G/1/PS$ para maximizar o lucro obtido na execução de aplicações de comércio eletrônico. As soluções foram avaliadas usando um simulador próprio e cargas de trabalho sintéticas com taxas de chegadas modeladas com distribuição de Poisson. Assim como Wang et al. [12] que também teve seu modelo avaliado por um simulador próprio com cargas de trabalho sintéticas (seguindo distribuição de Poisson) e cargas reais coletadas da aplicação Web da Copa do Mundo de 1998 [18].

2.2 Computação na Nuvem

Nos últimos anos, a computação na nuvem tem modificado vários setores da indústria e academia. Com isso, a ideia de oferecer computação como um serviço utilitário, como energia elétrica, vem se tornando uma realidade. Serviços como processamento, armazenamento, infraestrutura ou mesmo o acesso a um software através da Internet têm se tornado mais comum tanto para o departamento de Tecnologia da Informação (TI) das empresas como para os usuários finais.

Os serviços oferecidos como computação utilitária abrangem vários níveis. Apesar de não haver um consenso sobre a nomenclatura, a mais comumente empregada [25] classifica os serviços em: (i) Infraestrutura como Serviço (do inglês *Infrastructure as a Service - IAAS*), na qual estão serviços como processamento e armazenamento, tendo como exemplos mais conhecidos a Amazon Elastic Compute Cloud [26] e GoGrid [27]; (ii) Plataforma como Serviço (do inglês *Platform as a Service - PAAS*), englobando ferramentas para auxiliar desde o desenvolvimento de aplicações até a manutenção do ciclo de vida delas, representada aqui pelo Google App Engine [28]; e (iii) Software como Serviço (do inglês *Software as a Service - SAAS*), ofertando aplicações prontas para o usuário final, tendo como exemplos mais conhecidos aplicações para *Customer Relationship Management (CRM)* e editores de texto, planilhas e apresentações.

Um problema interessante e que merece uma atenção especial por impactar diretamente nos lucros do provedor de SaaS diz respeito a como a aplicação será mantida. A Figura 2.2 mostra como os diferentes papéis interagem nesse processo. Empresas contratantes usam os serviços disponibilizados pelo provedor de *SaaS* que, por sua vez,

adquire recursos dos provedores de *PaaS* ou diretamente de provedores de *IaaS*. Tanto a forma que esses recursos são adquiridos como a forma como são utilizados impactam diretamente no custo de manter os serviços e, consequentemente, no lucro do provedor SaaS.

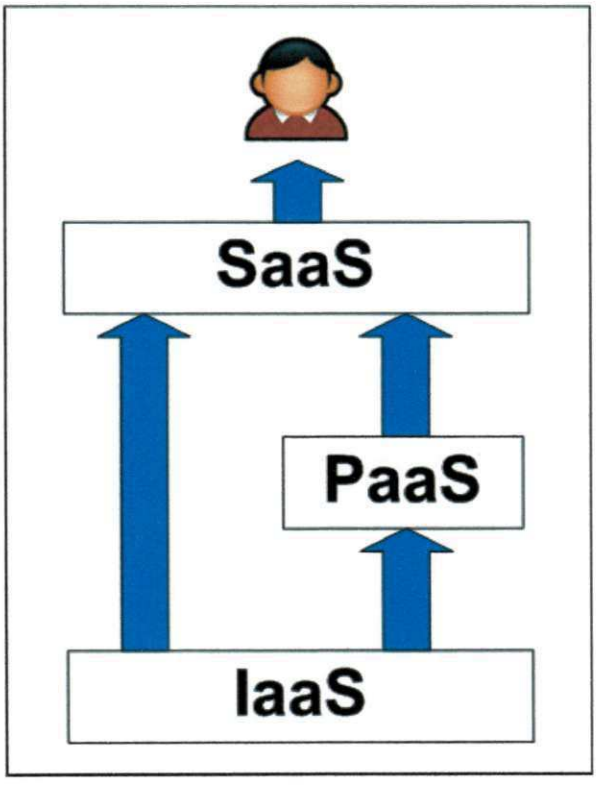


Figura 2.2: Interação entre clientes e provedores na nuvem

Atualmente, o provedor de *IaaS* mais conhecido é o Amazon EC2 [26], que oferece máquinas com diversas configurações a preços diferentes. O contratante paga por hora de máquina usada e pela quantidade de dados transferida e armazenada nos servidores. Atualmente a Amazon EC2 disponibiliza servidores que podem ser adquiridos em três mercados diferentes [29]:

Mercado sob demanda: As máquinas são compradas sem a necessidade de um contrato ou reserva. Dessa forma o usuário pode facilmente aumentar e diminuir a capacidade da aplicação. Em momentos de sobrecarga, o pedido de aquisição de instâncias sob demanda pode ser negado;

Mercado de reserva: Nessa modalidade as máquinas são reservadas pelo intervalo de

1 a 3 anos a um preço de uso mais baixo que o do mercado sob demanda e diante do pagamento de uma taxa de reserva. Dessa forma, quando o usuário precisar usar uma máquina reservada, a quantia paga pela hora utilizada é inferior ao do mercado sob demanda. Existe aqui a garantia de que as máquinas reservadas vão estar disponíveis quando solicitadas para uso.

Mercado baseado em lances de oferta: Essa modalidade de mercado é adequada para quem pode se beneficiar de máquinas num grão de tempo menor que o do mercado sob demanda. O usuário especifica um preço limite para a máquina-hora. Como o preço da instância flutua baseado na oferta e demanda por máquinas de um determinado tipo, quando o preço da instância está abaixo do limite especificado pelo usuário, a máquina é vendida. Se o preço da instância ultrapassar o limite do usuário, a máquina é encerrada.

Como consequência, um provedor de SaaS pode, por exemplo, realizar um planejamento de capacidade para o ano e implantá-lo adquirindo servidores no mercado de reserva. Correções no planejamento podem ser implementadas atendendo picos de demanda usando servidores comprados do mercado sob demanda [30]. Para casos em que a aplicação pode se beneficiar de um cenário *best-effort*, custos podem ser reduzidos comprando máquinas no mercado baseado em lances.

2.2.1 Mercado de Software as a Service

A ideia de prover acesso a aplicações por meio de *SaaS* não é nova. Brereton e Budgen [31] discutiram o desenvolvimento de software em componentes independentes que juntos entregariam um serviço de mais alto nível ao usuário. Já a ideia de separar os papéis de dono e usuário do software é defendida por Turner et al. [32]. Logo, podemos pensar num serviço composto por outros mais específicos que é vendido a uma empresa (contratante) por meio de uma assinatura. Os contratantes, por sua vez, disponibilizam a aplicação para os usuários finais. Por exemplo, consideremos um provedor de SaaS que disponibiliza sites de comércio eletrônico. Uma empresa contratante, personaliza a aplicação (com seus produtos, preços e logomarca) e divulga o site para sua rede de clientes que serão os usuários finais da aplicação provida.

Algumas outras características marcantes de *SaaS* são [33]:

Multi-locação: [9] Que significa prover acesso a uma mesma aplicação para diferentes contratantes, impedindo que um acesse informações do outro. Nesse caso, as aplicações devem ser genéricas o suficiente para permitir uma adaptação e customização do serviço (como no exemplo de comércio eletrônico no paragrafo anterior) oferecido a diferentes usuários. É necessário também uma gerência mais específica dos recursos utilizados para executar as diferentes instâncias do serviço impedindo que o alto processamento solicitado numa instância possa resultar num atraso no tempo de resposta de outra instância;

Cobrança em um grão pequeno: As aplicações *on-premise* [10] possuem um sistema de licença única no momento da aquisição da aplicação. No caso das aplicações *SaaS*, licenças temporárias são adquiridas com base na quantidade de usuários finais aos quais o contratante quer prover acesso e a cobrança é feita de acordo com a quantidade usada do serviço (processamento utilizado, quantidade de dados armazenados ou transferidos, por exemplo). Dessa forma, tanto a empresa contratante da aplicação acaba por economizar já que a cobrança é mais justa com relação ao tempo de uso da aplicação como o provedor da aplicação lucra provendo uma mesma aplicação para diferentes usuários;

Alta escalabilidade: A arquitetura das aplicações permite que se possa atender uma taxa maior de requisições ao aumentar a capacidade ou a quantidade dos recursos de infraestrutura usados;

Fácil composição de aplicações: É muito comum utilizar a saída de um serviço como entrada de outros serviços de forma a produzir um serviço mais especializado agregando valor a um ou mais outros. Isso tem sido possível atualmente com a utilização de tecnologias de Arquitetura Orientada a Serviços (do inglês

[9] em inglês é comum encontrar o termo *tenant* para se referir a uma instância da aplicação, e as aplicações que possuem essa característica são *multi-tenant*

[10] *on-premise* significa executar a aplicação na infraestrutura local da empresa, em oposição a *off-premise*, que significa executar numa infraestrutura remota. Aplicações *SaaS* são comumente chamadas de aplicações *off-premise*.

Service Oriented Architecture - SOA). Um exemplo pode ser uma aplicação que oferece serviço de edição de texto com correção automática provida por outra aplicação;

Facilidade de migração: Como o serviço é acessado através da Internet, mudar a infraestrutura onde a aplicação está hospedada não deve ser uma tarefa difícil. Dessa forma, uma empresa contratante pode utilizar recursos próprios de infraestrutura para executar a aplicação mantendo os dados na rede interna da empresa ou optar por executar numa infraestrutura remota;

Internacionalização: Disponibilizar a aplicação para clientes finais acessando a partir de diferentes regiões geográficas com o uso de Redes de Fornecimento de Conteúdo como o Akamai [11] visando aumentar o desempenho da aplicação.

2.2.2 Vantagens da metodologia *SaaS*

Muitas são as vantagens desse modelo de entrega de serviço, tanto para a empresa ou usuário final como para quem provê o serviço. Do lado de quem usa, as pequenas e médias empresas têm seus custos reduzidos com o uso do modelo *pay-as-you-go* pois não há mais a necessidade de adquirir e manter a aplicação muito menos infraestrutura para sua execução [34]. Grandes empresas, por sua vez, tiram proveito do fato de não precisar desenvolver uma nova aplicação. Além disso, há também a possibilidade de executar na própria infraestrutura da empresa reduzindo preocupações com segurança de dados críticos.

Na academia, iniciativas como Globus Online [35] e R-PHP [36] mostram a viabilidade na utilização desse modelo para prover serviços que auxiliem os pesquisadores durante o ciclo da pesquisa. Serviços como edição de texto, processadores de dados, classificação de artigos são de interesse de muitos pesquisadores e o uso de ferramentas que não necessitem de instalação reduz bastante o trabalho adicional e permitem um maior foco no próprio processo da pesquisa.

Quanto aos provedores, a mesma aplicação pode ser provida a vários usuários de empresas diferentes. Dessa forma, custos com desenvolvimento e manutenção da apli-

[11] `http://www.akamai.com`

cação e infraestrutura são compensados com a economia de escala já que a mesma infraestrutura pode ser utilizada para manter o mesmo serviço para milhares de clientes independentes. No entanto, vários outros problemas surgem com o emprego desse modelo, por exemplo: i) preocupações com a modelagem, design, construção e manutenção de aplicações *SaaS* [37; 38]; ii) segurança e privacidade de dados críticos de empresas; iii) como prover internacionalização; e iv) como fazer balanceamento de carga entre servidores adquiridos em vários provedores de *IaaS*.

2.3 Gerência Automática em Ambientes de Computação na Nuvem

Com a popularidade dos mercados de *IaaS* e *SaaS*, problemas como o provisionamento de aplicações voltaram a ser revisitados. Iqbal et al.[39] estudou preliminarmente como reagir a mudanças repentinas na carga submetida à aplicação com base no tempo de resposta das requisições. A solução foi avaliada com experimentação real de uma aplicação para cálculo de matrizes submetida a uma carga sintética em cenários de duração aproximada de 50 minutos. Chi et al.[14] investigou como realizar provisionamento dinâmico com um modelo mais rebuscado usando redes de filas com camadas, usando experimentação real com uma aplicação de venda de livros num ambiente controlado. Detalhes sobre a duração do experimento não foram reportados. É importante salientar que ambos os sistemas usaram o Eucalyptus[12] como provedor de infraestrutura durante a experimentação.

Algumas modelos propostos foram avaliados usando simuladores próprios. Wang et al.[12], Lee et al.[13] e Goudarzi et al.[40] implementam seus próprios simuladores dirigidos por eventos para analisar modelos para provisionamento e alocação de recursos de aplicações multicamadas, enquanto Katsalis et al.[41] avalia um modelo para diferenciação de serviço usando também um simulador próprio.

Wu et al. [42] propôs um algoritmo de alocação de recursos para aplicações com multilocação. O modelo foi avaliado usando o CloudSim[13] [43]. O CloudSim é um ambiente

[12] `http://www.eucalyptus.com`

[13] `http://www.cloudbus.org/cloudsim`

para modelagem e simulação de escalonamento de máquinas virtuais em infraestruturas de computação na nuvem, simplificando a modelagem de estratégias de alocação de recursos virtuais e detalhes como a conectividade de rede entre eles, além de oferecer suporte a avaliação de modelos de consumo de energia. O simulador apresentado nesse trabalho de dissertação diferencia-se por ter como foco a aplicação SaaS propriamente dita e seu modelo de valoração, executando em cenários de provisionamento dinâmico de recursos, em que há uma solução de escalonamento e/ou planejamento de capacidade, e que foi desenvolvido seguindo uma metodologia de validação e verificação dos componentes.

Capítulo 3

Visão Geral do Sistema de SaaS

Como já mencionado na Seção 2.1, estas aplicações apresentam tipicamente uma carga bastante variável ao longo do tempo e muitas vezes difícil de ser prevista. Tendo em mente esse problema, o simulador foi modelado visando contemplar três diferentes aspectos da gerência de aplicações SaaS: i) provisionamento dinâmico de recursos; ii) balanceamento de carga; e iii) planejamento de capacidade.

3.1 Avaliando soluções para provisão dinâmica de recursos

Modelos nessa categoria são desenvolvidos visando obter respostas para questões como:

- Quantas máquinas a aplicação precisa para atender todos os usuários no horário comercial?
- Quando devo adicionar máquinas na infraestrutura executando minha aplicação para prover qualidade de serviço aceitável para os usuários?
- Quando devo remover máquinas da minha aplicação para evitar desperdício de recursos?

Um modelo mais geral de um sistema de provisionamento dinâmico se baseia no laço de controle de realimentação mostrado na Figura 3.1. Destacam-se nele três grandes componentes: i) a aplicação sob a qual o sistema está atuando, ou no caso de um

modelo de simulação, uma representação da aplicação; ii) o monitor da aplicação, responsável por coletar informações sobre a infraestrutura e a execução da aplicação, e em alguns casos informar ao sistema de provisionamento a ocorrência de algum evento incomum durante a execução da aplicação; e iii) o sistema de provisão dinâmica (SPD) em si, no qual os dados coletados pelo monitor são processados podendo gerar um nova configuração a ser implantada na aplicação em execução.

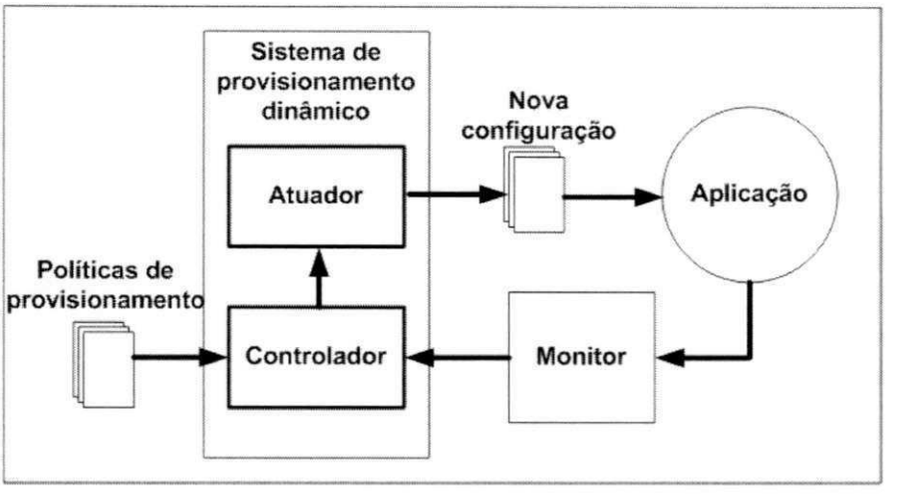


Figura 3.1: Modelo de Sistema de Provisionamento Dinâmico

O sistema de provisionamento dinâmico possui dois componentes: o controlador e o atuador. O primeiro é responsável por interpretar as informações recebidas do monitor da aplicação e, de acordo com políticas bem definidas de provisionamento fornecidas pelo administrador da aplicação, gerar uma nova configuração que será implantada na aplicação em execução pelo atuador. O atuador, por sua vez, conhece a arquitetura da aplicação e a infraestrutura que a executa e é responsável por aplicar ajustes na configuração da aplicação (soluções de controle de acesso) ou da infraestrutura (soluções de provisionamento dinâmico). Consideremos como exemplo um sistema de provisionamento implementando o algoritmo *QuID-online* proposto por Ranjan et al.[1]. O *QuID-online* observa o nível de utilização de cada uma das N máquinas que executam atualmente a aplicação Web e computa o número ideal N' de máquinas com base nos valores de utilização coletados e um valor alvo de utilização ρ fornecido pelo administrador responsável pela aplicação. Nesse caso, *QuID-online* tem como objetivo manter

o valor das utilizações das máquinas em torno do alvo ρ especificado adicionando ou removendo máquinas na infraestrutura que executa a aplicação Web de acordo com o parâmetro de saída N'.

No exemplo apresentado, o controlador implementa um algoritmo reativo. O sistema de provisão dinâmica segue um algoritmo que reajusta o provisionamento da infraestrutura quando percebe uma mudança na execução da aplicação. Algoritmos desse tipo, executam de tempos em tempos ou a partir da ocorrência de um certo evento registrado pelo monitor (e.g. o nível de utilização médio dos servidores ultrapassou um limite aceitável). Uma desvantagem dos algoritmos reativos é que eles podem demorar a tomar uma decisão e a implantação da nova configuração ser tardia, não conseguindo evitar a violação do SLA.

Por outro lado, existem algoritmos, como o proposto por Chandra et al.[2], que utilizam um modelo baseado em séries temporais para processar informações históricas sobre a aplicação para predizer parâmetros da carga de trabalho num futuro próximo e assim poder preparar e antecipar uma configuração necessária para executar a aplicação com uma probabilidade menor de violar o SLA. Tais algoritmos, classificados como proativos, contornam o problema no atraso da implantação de uma nova configuração sob o risco de desperdiçar recursos tomando decisões antecipadas demais. Geralmente ganha-se com a combinação dos dois tipos de atuação como no algoritmo proposto por Urgaonkar et al.[3], que além de atuar na reconfiguração da aplicação a partir do processamento de informações sobre a taxa de chegada de requisições em cada camada da aplicação coletadas em tempo real pelo monitor, utilizam informações de execuções passadas para estimar o comportamento da carga de trabalho da aplicação no próximo turno antecipando a implantação de uma nova configuração.

3.2 Avaliando soluções para balanceamento de carga

Uma questão tão importante quanto decidir quando ou quantas máquinas usar da infraestrutura disponível é a decisão sobre como utilizar as máquinas disponíveis na execução das requisições. Soluções de balanceamento de carga e alocação de recursos visam resolver esse tipo de problema.

Considerando o modelo simplificado apresentado na Figura 3.2, uma requisição submetida por um usuário final é recebida por um componente sentinela (S), responsável por autorizar o processamento da requisição. Sentinelas são usadas em modelos que implementam controle de admissão. Uma vez que a requisição é autorizada, o componente sentinela a encaminha para um balanceador de carga (LB) que, com base em informações coletadas sobre a execução da aplicação, decide para qual dos servidores disponíveis na camada a requisição será enviada para ser processada. O modelo da Figura 3.2 é facilmente extensível para suportar aplicações com mais camadas.

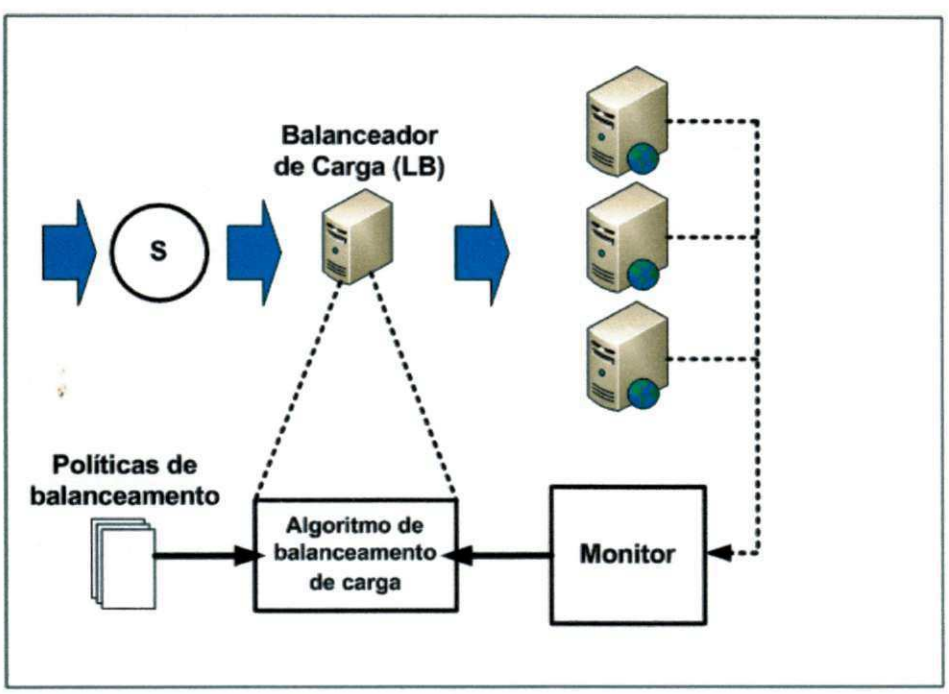


Figura 3.2: Modelo de Balanceamento de Carga da Aplicação

Os balanceadores de carga e sentinelas devem realizar o seu trabalho com o objetivo de aproveitar ao máximo os recursos que estão disponíveis no momento para a execução da aplicação. Neste sentido, existem pesquisas que propõem diferentes soluções para lidar com os problemas de balanceamento de carga e controle de admissão, que fogem do escopo dessa dissertação, mas que certamente poderiam se beneficiar do modelo de simulação aqui descrito, conforme ilustrado na Figura 3.2.

3.3 Avaliando soluções para planejamento de capacidade

Modelos de planejamento de capacidade são propostos visando obter respostas para perguntas como: "Dada uma previsão da demanda por uma aplicação num período futuro tipicamente longo (da ordem de um ou mais anos), qual a quantidade de recursos necessária para manter níveis aceitáveis de QoS para os usuários sem infringir o SLA?". Existem soluções para o planejamento de capacidade em longo prazo, que geralmente tem sua eficácia bastante atrelada à acurácia da estimativa realizada para os parâmetros da carga de trabalho esperada para o período que se deseja planejar.

Modelos de planejamento de capacidade tem como saída um plano de aquisição de recursos para a infraestrutura no decorrer do período futuro sendo planejado (ver Figura 3.3). O plano de aquisição pode mudar bastante dependendo do cenário de planejamento. Tradicionalmente esse plano levava à aquisição de recursos computacionais para executar a aplicação *in-house* com capacidade constante e superprovida. O modelo de mercado de IaaS permite que planos de aquisição sejam implantados tirando proveito dos mercados de reserva e sob demanda [44].

Decidir quando um plano de capacidade é considerado aceitável requer verificar o desempenho da aplicação na infraestrutura. Como é impraticável testar diversos planos em infraestruturas reais a fim de se decidir qual a melhor alternativa, simuladores são usados para que se tenham indícios de como a aplicação se comportaria num ambiente real segundo um plano de aquisição de recursos, como mostrado na Figura 3.3.

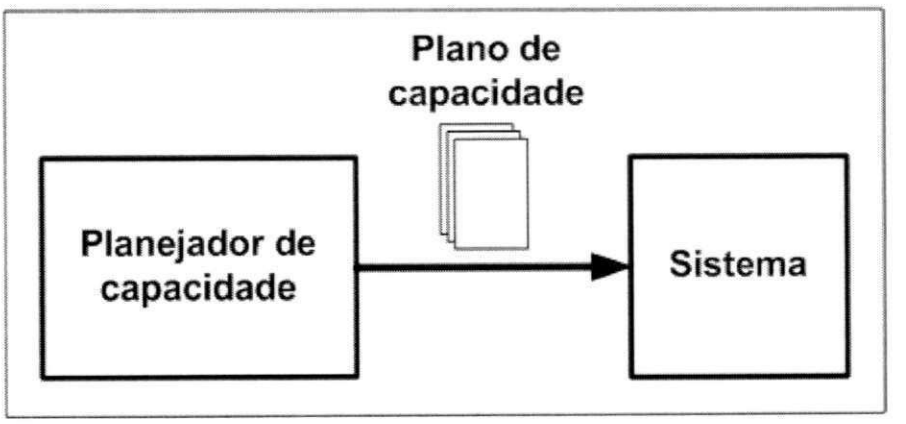


Figura 3.3: Modelo de Planejamento de Capacidade

Capítulo 4

Design do Framework

4.1 Componentes do Simulador

Visando contemplar os aspectos identificados no Capítulo 3, o design do framework foi construído segundo o modelo apresentado na Figura 4.1.

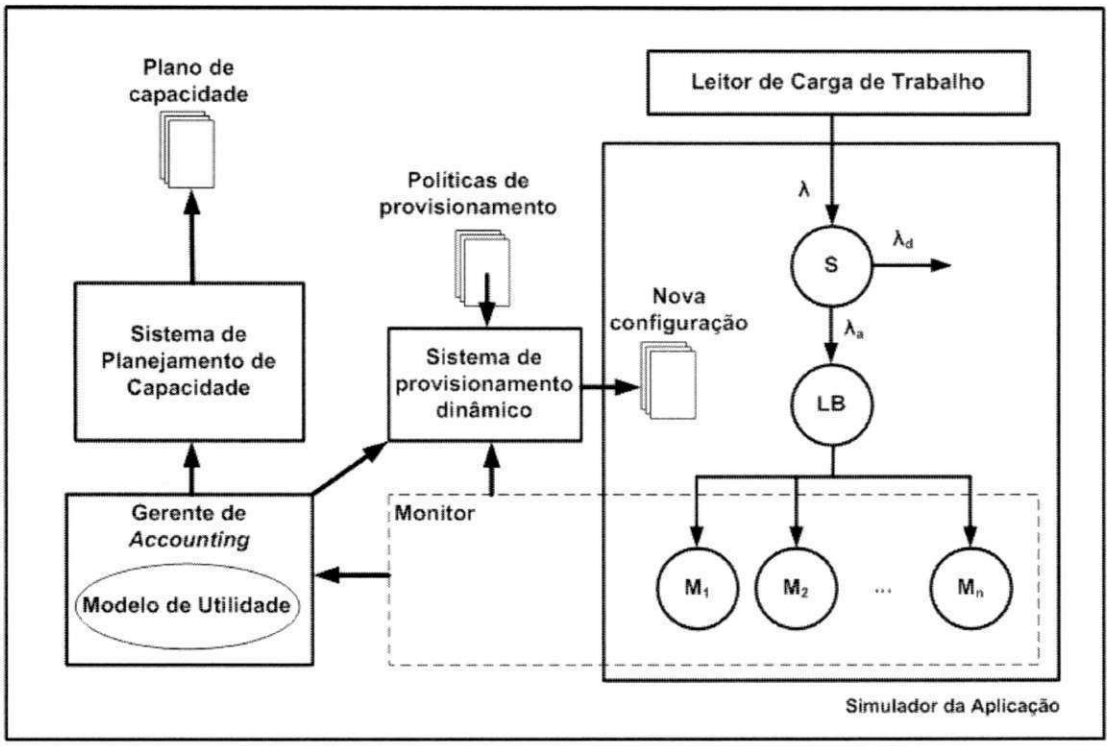


Figura 4.1: Modelo do Simulador

Um leitor de carga de trabalho é responsável por fazer o *parsing* dos rastros usados para alimentar a aplicação. Como dito anteriormente, no cenário de aplicações SaaS, uma mesma aplicação pode ser provida para vários contratantes. Esses contratantes do provedor SaaS, por sua vez, disponibilizam a aplicação para os seus usuários finais. Com isso, o simulador precisa estar preparado para lidar tanto com uma carga de trabalho agregada quanto arquivos de rastros independentes para cada contratante da aplicação SaaS.

O simulador conta com um módulo de monitoramento que tem por responsabilidade coletar métricas técnicas a respeito da aplicação e do uso dos recursos da infraestrutura como o número de requisições recebidas, a porcentagem delas que não foi atendida, o tempo de processamento das requisições atendidas, quantos servidores estão ativos na aplicação, entre outras. Novas métrias podem ser incorporadas através da extensão desse módulo. As informações coletadas pelo monitor são encaminhadas para: i) o gerente de *accounting*[1]; e ii) o sistema de provisionamento dinâmico.

O gerente de *accounting* é o componente que implementa o modelo de cobrança típico de aplicações SaaS. Nesse componente, os dados coletados pelo monitor são processados e usados para alimentar o modelo de valoração associado à aplicação SaaS. Uma implementação baseada no conceito de utilidade é descrita mais à frente na Seção 5.1.5. Uma vez processadas, essas informações são úteis para sistemas de planejamento de capacidade, que produzem um plano de capacidade associado com a execução da aplicação com a carga de trabalho usada como entrada. O módulo de provisionamento dinâmico é alimentado tanto com os dados coletados diretamente pelo monitor como pelas métricas processadas pelo gerente de *accounting*. Isso permite que o simulador avalie sistemas baseados tanto em métricas técnicas como em métricas dirigidas por negócio. Em ambos os casos, o sistema de provisionamento, ao confrontar as métricas coletadas com as políticas de provisionamento definidas, pode produzir uma nova configuração a ser aplicada na infraestrutura usada para executar a aplicação Web.

Para simplificar a implementação, modelamos apenas a camada de aplicação (ver Figura 2.1). A infraestrutura que executa essa camada é composta por três componen-

[1] A palavra *accounting* foi usada ao longo do texto por desconhecer-se uma outra em português que capturasse a função do componente.

tes: i) uma sentinela (S), responsável por implementar políticas de controle de acesso; ii) um balanceador de carga(LB), cuja responsabilidade principal é decidir qual servidor deve atender uma determinada requisição, permitindo assim implementar mecanismos de escalonamento de requisições, diferenciação e degradação de serviço; e iii) servidores de aplicação (M_i), que são responsáveis por atender as requisições. As configurações produzidas pelo sistema de provisionamento dinâmico atuam sobre a configuração desses três componentes.

Para modelar o servidor de aplicação, usamos um modelo baseado em Menascé et al. [45] (ver Figura 4.2). As requisições que chegam no servidor de aplicação entram numa fila de espera com política "primeiro a chegar, primeiro a ser atendido" (do inglês *First Come, First Served - FCFS*). Um sistema de fichas é usado para controlar quantas requisições podem ser atendidas simultaneamente. Uma vez que haja ficha disponível, uma requisição é encaminhada para uma fila de processamento com política *Round-Robin*, onde cada requisição executa no recurso durante uma fatia de tempo chamada de quantum. Se a demanda da execução for maior que o quantum, a requisição volta para o final da fila. O custo da troca de contexto é considerado desprezível. Ao finalizar a execução da demanda de uma requisição, a ficha é devolvida e uma nova requisição, se houver, é retirada da fila de espera e posta para execução.

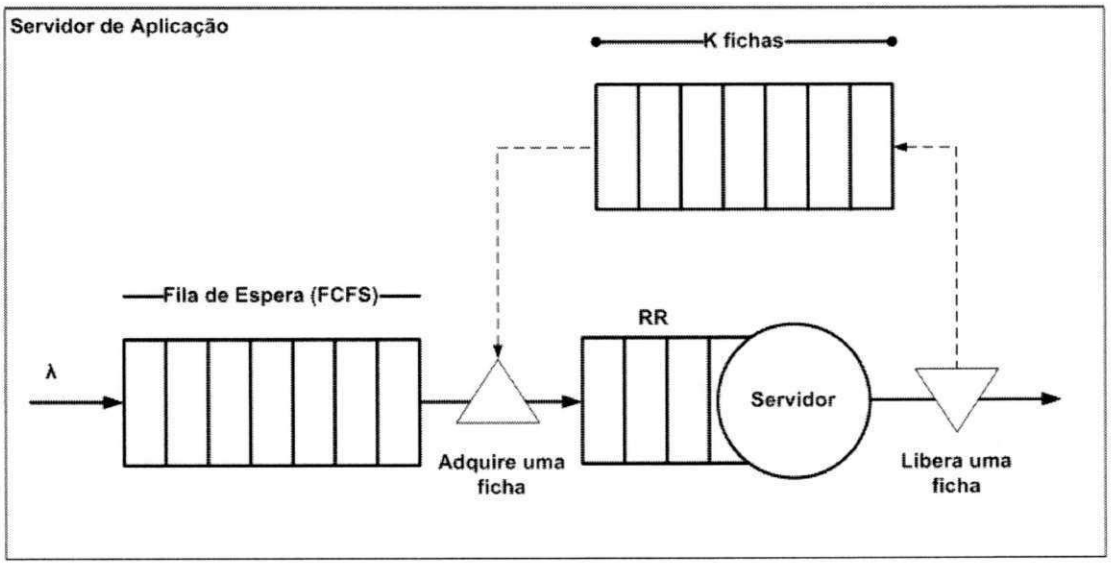


Figura 4.2: Modelo do Servidor de Aplicação

4.2 Fator de Correção

O modelo de servidor de aplicação apresentado na seção anterior, por ser bastante simples, pode não ser acurado o suficiente para simular o comportamento de uma aplicação. Nesse caso, modelamos um fator de correção que aproxima os resultados obtidos nos experimentos de simulação com medições realizadas em ambiente controlado. O fator de correção foi modelado com base na quantidade de requisições na fila de processamento com política *Round Robin* do modelo proposto por Menascé et al. [45] apresentado na Figura 4.2.

Realizamos um experimento de medição para observar o comportamento da execução das requisições em diferentes cenários de utilização. A hipótese que motivou estes experimentos é que o modelo de simulação é bastante simplificado e não considera a carga gerada pelo sistema operacional e pelo próprio servidor de aplicação quando submetido a cargas altas de utilização. Quando operações do sistema operacional tiverem que ser realizadas com bastante frequência o desempenho da aplicação pode piorar e isto não está modelado no simulador. O experimento descrito a seguir é uma investigação de como obter um fator de correção para o design apresentado na seção anterior.

A metodologia usada para determinar um fator de correção para o design proposto baseia-se nos seguintes passos: i) preprarar infraestrutura e aplicação similares às que se pretendem simular; ii) coletar tempos de resposta para as requisições em cenários significativos para a simulação; iii) extrair valores de demanda para cada requisição a partir dos tempos de resposta; iv) agrupar os valores de demanda das requisições pela quantidade de requisições existentes no sistema no momento de chegada no servidor; e v) extrair o fator de correção comparando os valores de cada grupo com uma estimativa acurada da demanda real da requisição.

Nossa implementação desse experimento real de medição consistiu num cliente disparando requisições HTTP para uma máquina servidor com o *container* web instalado e uma aplicação simples com pouco acesso ao disco, mais adequando ao modelo usado no design do framework. Como máquina cliente, foi utilizado um servidor com Intel Xeon X5550 com 8 núcleos de 2.26GHz e 20GB de memória principal. Uma máquina

virtual com processamento equivalente a um dos 8 núcleos e memória principal de 1GB foi utilizada para hospedar o servidor de aplicações. A máquina virtual executa na mesma máquina que o cliente, minimizando o impacto na taxa de chegada provocado por atrasos na transferência na rede. O *container* de aplicações utilizado foi o Apache Tomcat/7.0.27.0 com duas alterações na configuração original: (i) a quantidade de linhas de execução no sistema foi mantida com valor 200, alterando os parâmetros `minSpareThreads` e `maxSpareThreads`; e (ii) a JVM foi iniciada com o parâmetro `-server` ativado para aumentar o desempenho da máquina virtual.

Foi usada uma aplicação Web Service *RESTful* exemplo[2] desenvolvida usando a API Jersey[3]. A aplicação possui uma única operação (realiza processamento de matrizes) acessada através de uma URL específica. Todas as requisições são semelhantes com intervalo de 95% de confiança para a demanda média das requisições de 199.805 ± 0.8284. Essa medição foi realizada com a execução isolada num terminal da operação implementada no WebService. Acreditamos que, dada a baixa variação do valor coletado, essa estimativa é próxima da demanda real da operação realizada.

Para obtermos cenários significativos, focamos em experimentar a aplicação em diferentes níveis de utilização do recurso. Para isso, controlamos o tempo entre os disparos das requisições de acordo com a tabela 4.1. Durante a execução dessas configurações, a utilização da máquina foi medida usando o utilitário `sar` do pacote `sysstat`[4].Na tabela 4.2 são apresentados os intervalos de 95% de confiança da utilização média dos recursos medida com a instrumentação da aplicação. É possível ver que a aplicação foi experimentada em níveis distintos de utilização, conforme planejado inicialmente.

[2]disponível em `http://www.lsd.ufcg.edu.br/~ricardo/saasim/saaswebservice.zip`

[3]`http://http://jersey.java.net`

[4]`http://sebastien.godard.pagesperso-orange.fr`

Cenário	c_1	c_2	c_3	c_4	c_5
Tempo entre disparo das requisições (ms)	1000	800	600	400	200
Número de requisições enviadas	1800	2250	3000	4500	9000
Duração	30 minutos				
Repetições em cada cenário	30				
Demanda média da requisição (ms)	199.805 ± 0.8284				

Tabela 4.1: Parâmetros para o experimento de medição

c_1	c_2	c_3	c_4	c_5
$37.80 \pm 0.13\%$	$47.11 \pm 0.15\%$	$62.71 \pm 0.23\%$	$87.34 \pm 0.32\%$	$99.7 \pm 0.27\%$

Tabela 4.2: Utilização média do servidor no experimento de medição (IC 95%)

Para cada requisição, o tempo de chegada no servidor de aplicação e o tempo de resposta são coletados. Como esperado o tempo de execução medido é maior ou igual ao tempo de demanda real da aplicação. Isso se deve ao fato de que as requisições competem pelos recursos da máquina com o sistema operacional e outros processos entre eles o próprio gerenciador do servidor de aplicação. Para estimar o impacto dos outros processos na execução dessas requisições, precisamos observar o tempo que cada requisição utiliza do recurso. Como esperado, as requisições atendidas executam em pseudo-paralelismo competindo pelos recursos, principalmente nos cenários c_4 e c_5 nos quais a taxa de chegada é maior. Como não há uma maneira direta de coletar quanto tempo do recurso cada requisição de fato usou, estimamos esse valor dividindo o tempo decorrido na execução pelo número de requisições que estavam executando no momento (ver figura 4.3). Para determinar a demanda da requisição R_2, por exemplo, temos:

$$t_{R_2} = \frac{(t_2 - t1)}{2} + \frac{(t_3 - t_2)}{3} + \frac{(t_4 - t_3)}{2} + (t_5 - t_4) \tag{4.1}$$

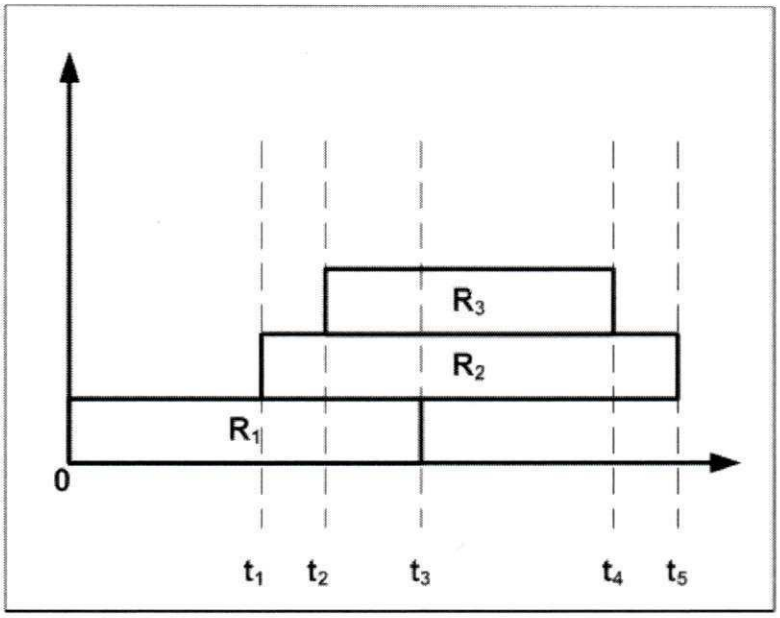


Figura 4.3: Requisições em paralelo no servidor de aplicação

Dessa forma, conseguimos extrair a partir do tempo de resposta de cada requisição uma super-estimativa de quanto tempo do recurso cada requisição tomou. Para a aplicação usada na experimentação, o conjunto de demandas das requisições extraídas usando o processo acima não seguiu uma distribuição normal mesmo considerando cenários e repetições isoladamente. Optamos por comparar a média das demandas de cada repetição, que pelo teorema do limite central seguem uma distribuição normal. A demanda média estimada, para nossa surpresa, diminui nos cenários de maior utilização média (ver figura 4.3 se aproximando do valor computado ao instrumentar a aplicação (aproximadamente $200ms$).

Observando amostras do cenário c_5, notou-se que a demanda das requisições tende a aproximar do valor estimado inicialmente (em torno de 200 milissegundos) conforme o número de requisições executando em paralelo aumenta. Isso se deve à ação de mecanismos de cache do servidor Apache Tomcat não incorporados ao modelo. Esse fenômeno também pode ser percebido observando a demanda das requisições nos cenários c_1 e c_2, por exemplo. Nesses casos como o intervalo entre as requisições é maior, a ocorrência de competição por recursos é quase inexistente, no entanto, o tempo para executar cada requisição é maior. O servidor acaba tomando o dobro do tempo estimado na

Cenário	Média da demanda das requisições
c_1	$378.4063148 \pm 0.6667984$
c_2	$377.3982963 \pm 0.6142368$
c_3	$377.0560667 \pm 0.6848832$
c_4	$350.5255556 \pm 0.6518946$
c_5	203.176959 ± 1.044786

Tabela 4.3: Intervalo de confiança da demanda média estimada

instrumentação para executar as requisições por não tirar proveito dos mecanismos de cache.

No design atual, optamos por não modelar a causa desse impacto na demanda estimada das requisições. Escolhemos associar esse impacto, nesse trabalho à quantidade de requisições que o servidor executa em paralelo. Utilizando as demandas extraídas (com um método baseado na Equação 4.1) dos rastros de tempo de resposta, obtivemos os valores mostrados na tabela 4.4. Percebeu-se na análise que acima de 5 requisições, os valores para correção flutuavam em torno de 1 com variância muito baixa.

Número de requisições paralelas	Fator de correção
0	1.7934613
1	1.0720390
2	0.9938633
3	0.9993731
4	1.0000850
5	1.0080514
6 ou mais	1

Tabela 4.4: Fator de correção para requisições paralelas

A partir desse processo, pudemos determinar valores para o fator de correção modelado em cada servidor M da Figura 4.1.

Capítulo 5

Implementação do Framework de Simulação

5.1 O SaaSim

O *Software as a Service Simulator* (SaaSim) é uma implementação do design levantado no capítulo 4 para simulação de aplicações Web distribuídas horizontalmente escaláveis. O código está escrito na linguagem de programação Java e atualmente está disponível para uso[1].

O ambiente de simulação é dirigido por eventos e, portanto, construído em cima de um núcleo de processamento de eventos. As classes no pacote `saasim.sim.core` são responsáveis pelo ordenamento na linha do tempo e envio de eventos para os tratadores. Todos os componentes realizam trocas de mensagem agendadas no *EventScheduler*. Eventos agendados para o mesmo instante de tempo são executados na mesma ordem que são agendados. Novos tratadores de eventos podem ser adicionados no sistema após o início da simulação. Avaliamos do SimJava [2] para a implementação desse módulo, no entanto, limitações como a impossibilidade de adicionar novos tratadores de evento durante a simulação, nos levaram a optar por implementar uma versão própria.

A configuração da simulação é feita pelo gerente de configuração através de um único arquivo de propriedades na forma de pares de chave e valor. O Anexo A contem

[1] Código disponível em `http://www.lsd.ufcg.edu.br/~ricardo/saasim`

[2] `http://www.dcs.ed.ac.uk/home/hase/simjava/`

exemplo de arquivo de configuração para uma aplicação com uma camada usando provisionamento estático, com uma infraestrutura composta por 10 máquinas adquiridas no mercado sob demanda de um provedor de IaaS e um único cliente SaaS contratante do plano do tipo *diamond*. Detalhamos, a seguir, a implementação de cada componente no design proposto na Seção 4.1 e como ele pode ser configurado.

5.1.1 Leitor de Cargas de Trabalho

O componente responsável pela leitura da carga de trabalho foi implementado no pacote `saasim.io`. O processo de leitura é separado em duas partes: i) gerência da leitura; e ii) parsing do arquivo da carga de trabalho. Para a gerência da leitura, o SaaSim disponibiliza uma implementação padrão baseada na leitura de um arquivo de carga de trabalho para cada contratante (*tenant*) da aplicação SaaS e uma alternativa que realiza a leitura de um único arquivo com a carga agregada do sistema. Outro diferencial da implementação é poder configurar a quantidade de requisições lidas por vez dos traços com base numa janela de tempo configurada com a propriedade `dps.workload.pagesize` no arquivo de propriedades. Quanto ao parsing dos arquivos de carga de trabalho, a implementação padrão realiza leitura do padrão de arquivo do GEIST [46], um gerador de cargas de trabalho para aplicações de comércio eletrônico, onde cada linha é composta pelas seguintes informações:

- identificador único do usuário, pode ser usado para mecanismos de escalonamento baseados em sessão, por exemplo;
- identificador único da requisição;
- tempo de chegada da requisição no sistema, em milissegundos;
- tamanho da requisição em número bytes recebidos;
- tamanho da resposta, em número de bytes enviados, que a aplicação geraria com a execução da requisição; e
- valor da demanda em milissegundos.

Novos formatos podem ser implementados extendendo a classe `saasim.io.WorkloadParser` e sinalizando no arquivo de propriedades na propriedade `dps.workload.parser` o nome da nova classe responsável pelo *parsing*.

5.1.2 Sistema de Provisionamento Dinâmico

No pacote `saasim.provisioning` estão as classes que implementam o monitor e o gerente de provisionamento baseado em laço de realimentação como na figura 3.1. São providas implementações básicas para três controladores: i) baseado em provisionamento estático; ii) o algoritmo *QuID-online* proposto por Ranjan et al. [1]; e iii) o algoritmo proposto por Urgaonkar et al. [3]. A escolha sobre qual algoritmo usar é feita através do arquivo de configuração com a opção `dps.heuristic`. Novos algoritmos podem ser avaliados com o simulador implementando uma extensão da classe *DynamicProvisioningSystem*, usando o valor `CUSTOM` para a propriedade `dps.heuristic` e indicando o nome da classe implementada na propriedade `dps.heuristicclass`.

5.1.3 Sistema de Planejamento de Capacidade

O pacote `saasim.planning` possui implementações para o sistema de planejamento de capacidade. Classes no pacote `saasim.planning.io` são responsáveis por gerenciar a leitura de carga de trabalho usada, por exemplo, com heurísticas de predição. Heurísticas são criadas implementando a interface `saasim.planning.heuristic.PlanningHeuristic` e configurando o arquivo de propriedades.

5.1.4 Aplicação, Infraestrutura e Monitoramento

A arquitetura da aplicação em si é implementada extendendo a classe `saasim.sim.DynamicConfigurable`. O SaaSim possui uma implementação padrão é para uma aplicação web de uma única camada mas novas classes podem ser implementadas e sinalizadas na configuração na propriedade `saas.application.factoryclass`.

Por outro lado, componentes da infraestrutura que executa a aplicação são implementados no pacote `saasim.sim.components` com implementações padrão para ba-

lanceador de carga e servidores de aplicação (com o fator de correção) baseados no design proposto. No pacote saasim.sim.schedulingheuristics estão presentes as classes para implementação de heurísticas de balanceamento de carga. O simulador possui uma implementação padrão do algoritmo *Round Robin*, mas novos algoritmos podem ser usados estendendo a clase `SchedulingHeuristic` e configurando a propriedade `saas.application.heuristic`.

O trabalho executado pelo componente Monitor apresentado no design (ver Figura 4.1) foi distribuído entre classes relacionadas à infraestrutura. As métricas são coletadas pelas próprias implementações do servidor de aplicação e do balanceador de carga e informadas ao componente de provisionamento dinâmico.

5.1.5 Gerente de Accounting e Modelo de utilidade

Uma vez que o simulador apresentado nesse trabalho de dissertação propõe-se a modelar uma aplicação Web no contexto de SaaS, é importante que exista um modelo de negócio associado a tal aplicação, permitindo que as soluções de planejamento de capacidade e provisão dinâmica anteriormente apresentadas sejam avaliadas não apenas segundo métricas técnicas (como disponibilidade e tempo de resposta) mas também segundo métricas de negócio [47].

Para incorporar características do modelo de negócio das aplicações SaaS, a solução modelada usa funções de utilidade [48]. Modelos utilizando esse tipo de função são usados para representar a preferência de agentes para guiar seu comportamento no sistema, permitindo que diferentes tipos de métricas sejam combinadas. No modelo de computação na nuvem, o conceito de utilidade pode ser mapeado no lucro obtido por um provedor de SaaS que oferece uma aplicação Web distribuída horizontalmente escalável para diversos clientes, executando numa infraestrutura adquirida num provedor de IaaS.

Atualmente no mercado existem diversas aplicações SaaS com diferentes modelos de negócio, por exemplo:

- Plano único: Provedores de aplicações nesse modelo disponibilizam um único plano de adesão ao serviço. São usados tipicamente para aplicações Business to

Consumer (B2C) que oferecem o serviço a clientes individuais. São exemplos de aplicações nesse modelo: Evernote[3], Flickr[4] e Microsoft CRM Dynamics[5]

- Múltiplos planos: as aplicações com esse tipo de modelo são oferecidas pelos provedores SaaS em diferentes planos de adesão, geralmente chamados Bronze, Gold, Platinum ou Home, Professional, Enterprise Edition. Os diferentes planos permitem que clientes com diferentes demandas de servico se beneficiem do uso da aplicação. Aplicações que utilizam esse modelo são: BigCommerce[6], DeskAway[7], ServiceCloud[8], SurveyMonkey[9], Volusion Ecommerce Software[10] e Linkedin[11];

- Sem plano: cuja cobrança é realizada por transação ou operação bem sucedida. O CampaignMonitor[12], por exemplo, é um sistema para gerência de *newsletters* no qual o cliente paga uma taxa fixa por campanha realizada e um adicional relacionado à quantidade de destinatários da campanha. O doattend[13] implanta um modelo de cobrança similar para um serviço de gerência de eventos, venda de tickets e cobertura online de eventos.

Dentre as categorias de modelo de negócio, as mais frequentes são: i) cobrança periódica (mensalmente, bimestralmente ou anualmente) de uma taxa fixa para utilizar a aplicação; ii) o contrato da aplicação impõe uma quantidade limitada de recursos/operações que podem ser utilizados no período; iii) cobrança extra pelos recursos consumidos além da quantidade limite permitida pelo plano no período; iv) pagamento de uma taxa inicial para *set up* do serviço; v) especificação de regras de reembolso da mensalidade mediante o descumprimento do SLA estabelecido.

Considerando as características descritas anteriormente, o modelo da solução considera um provedor de SaaS que oferece uma mesma aplicação A a um conjunto de

[3] http://www.evernote.com
[4] http://www.flickr.com
[5] http://crm.dynamics.com
[6] http://www.bigcommerce.com
[7] http://www.deskaway.com
[8] http://www.salesforce.com/br/crm/products.jsp
[9] http://www.surveymonkey.com
[10] http://www.volusion.com
[11] http://www.linkedin.com
[12] http://www.campaignmonitor.com
[13] http://doattend.com

clientes (*tenants*) $U = \{u_1, u_2, u_3, \ldots, u_{|U|}\}$. Visando atender clientes com diferentes demandas por serviço com diferentes SLAs, o provedor oferece um portfólio de múltiplos planos $P = \{p_1, p_2, \ldots, p_{|P|}\}$ tal que cada plano p define a quantidade de serviço para uso num determinado período de tempo τ mediante o pagamento de uma taxa. Dessa forma, considerando uma aplicação com intervalo de tempo de contratação definido por $D = [1, d]$, ou seja, d intervalos de tamanho τ, temos que se τ for definido como sendo um mês, a aplicação é cobrada mensalmente e um intervalo D é composto por d meses.

A receita total (ι) obtida provendo a aplicação num intervalo $D = [1, d]$ é computada somando a receita (i_n) obtida em cada período n do intervalo D como mostrado na Equação 5.1:

$$\iota(D) = \sum_{n=1}^{d} i_n \tag{5.1}$$

Por simplicidade e sem perda de generalidade, toda simulação inicia no período 1 e tem duração de d períodos. Assim o intervalo [1,d] representa a sequência de todos os períodos de simulação a serem analisados. Se a tarifa cobrada pelo provedor for mensal, então cada período tem duração de 1 mês; se for semestral, cada período tem duração de 6 meses e assim por diante. A receita no n-ésimo período (representada por i_n) é calculada através da soma da receita obtida no período provendo o serviço para cada cliente $u \in U$, como mostrado na Equação 5.2. Cada parcela $i_{n,u}$ da Equação 5.2 é composta pela taxa fixa cobrada pelo serviço, a taxa cobrada por operações excedendo o limite estabelecido e a taxa de ressarcimento ao cliente quando aplicável (infração do SLA).

$$i_n = \sum_{u \in U} i_{n,u} \tag{5.2}$$

O modelo nesse trabalho considera dois tipos de infração: i) indisponibilidade da aplicação, quando requisições dos usuários não são respondidas; e ii) infração do tempo de resposta, quando as requisições são respondidas acima de um tempo limite estabelecido.

A aplicação modelada é executada numa infraestrutura adquirida num provedor de

IaaS. O custo por manter a aplicação A executando durante um intervalo de tempo D é composto pela soma dos custos da infraesturura em cada um dos n-ésimos períodos de tamanho τ (ver Equação 5.3). Cada parcela c_n é composta pela soma do custo ca_n de instâncias compradas no mercado de reserva com o custo cv_n referente a instâncias adquiridas no mercado sob demanda (ver Equação 5.4).

$$\alpha(D) = \sum_{n=1}^{d} c_n \tag{5.3}$$

$$c_n = ca_n + cv_n \tag{5.4}$$

A partir dos componentes de receita e custo modelados, podemos calcular a utilidade (υ) da aplicação no intervalo de tempo D conforme a Equação 5.5.

$$\upsilon(D) = \iota(D) - \alpha(D) \tag{5.5}$$

As classes que implementam esse modelo estão no pacote `saasim.cloud`. Dados relativos a execução da aplicação são enviados ao *AccountingManager* que mantém informações para cada contratante da aplicação SaaS. Nesse pacote também constam as classes para implementação do contrato estabelecido e do modelo de negócio dos provedores de infraestrutura envolvidos segundo o modelo matemático apresentado acima.

Capítulo 6

Validação

No processo de design e implementação do framework, simplificações no modelo do sistema real foram inseridas a fim de reduzir o custo de implementação e execução dos experimentos de simulação. Dessa forma, precisamos (i) verificar se a implementação de fato corresponde ao modelo de conceitual capturado pelo design proposto (capítulo 4) e (ii) validar o design do framework proposto contrapondo-o a um sistema real, de forma que seja possível avaliar quão significativas são as avaliações realizadas usando a implementação. A verificação e validação do nosso modelo foi realizada tomando como base a versão simplificada do processo de modelagem [49] na figura 6.1. Tal processo foca nas interações entre o sistema real, o modelo conceitual extraído do sistema e a implementação computadorizada desse modelo. Com isso a tarefa de validação se decompõe em parcelas mais específicas. A seguir, detalhamos como lidamos com as várias partes desse processo.

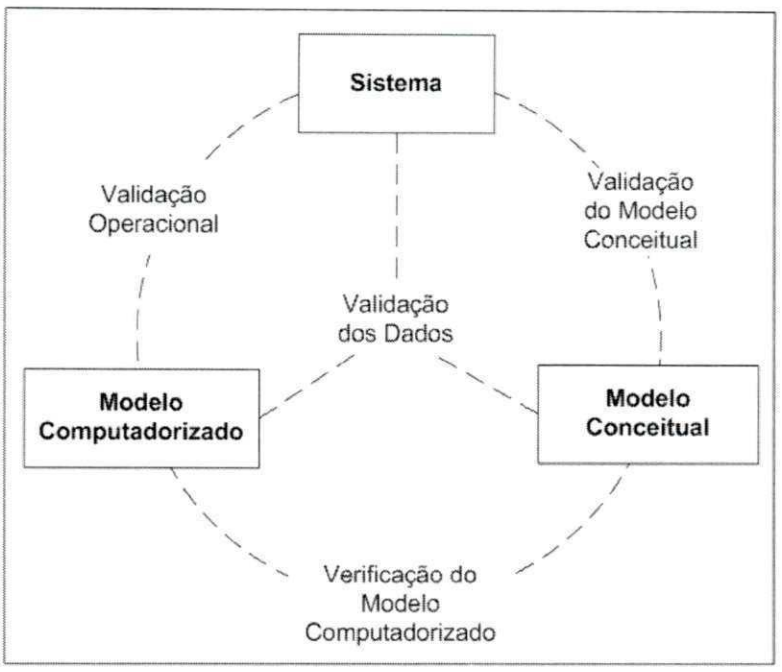


Figura 6.1: Versão Simplificada do Processo de Modelagem [49]

6.1 Validação do modelo conceitual

O modelo de servidor de aplicação utilizado captura somente requisitos de processamento de uma aplicação. Aplicações com demandas que envolvam acesso ao disco, por exemplo, podem ser modeladas e incluídas no modelo de simulação, mas fogem ao escopo dessa dissertação. Para esses casos é interessante realizar uma nova avaliação do modelo numa infraestrutura real com experimento de medição nos moldes da apresentada na Seção 4.2.

6.2 Verificação do modelo implementado

O desenvolvimento do modelo de simulação foi realizado seguindo as práticas do *Test-driven Development - TDD*. Os componentes foram avaliados independemente com testes de unidade. Em módulos como o gerente de *accounting*, o componente inteiro foi submetido a testes de aceitação com cenários muito simples para verificar a saída gerada pelo modelo de utilidade, contrastando com os valores obtidos analiticamente.

O código fonte atualmente possui cobertura de testes com média em torno de 60%.

6.3 Validação dos dados

Como dito anteriormente, existe o problema da falta de cargas de trabalho reais coletadas de sistemas em produção. Existem diversos tipos de aplicação Web e com requisitos de demanda diferentes, tornando difícil criar um modelo que seja geral para todo tipo de carga de trabalho Web. Por isso, o uso de cargas sintéticas acaba restringindo a validade dos resultados conseguidos uma vez que a modelagem de cargas com requisitos de demanda diferentes podem levar a conclusões diferentes.

Os dados usados nos cenários de refinamento do modelo e validação são restritos a aplicações horizontalmente escaláveis com requisições que representam demandas de processamento. Quando a demanda de outros recursos precisar ser modelada, uma extensão desse modelo precisa ser definida e novas avaliações e validação são necessárias.

Quanto à coleta das métricas dos experimentos de medição, o fato das máquinas cliente e servidor serem isoladas da rede local minimiza o impacto da latência de rede na taxa de chegada das requisições no servidor. Com isso é razoável considerar que os resultados apresentados são válidos para tempos de resposta medidos pelo provedor de SaaS. Entidades externas ao provedor podem perceber um aumento no tempo de resposta devido a fatores que não foram modelados nesse trabalho.

6.4 Validação Operacional

Nessa etapa da validação, confrontamos o modelo implementado com a mesma versão controlada do sistema real utilizada na etapa de refinamento do modelo em alguns cenários significativos para o domínio da aplicação. O ambiente simulado, implementado com o SaaSim, possui configurações similares ao ambiente de experimentação, com uma única camada de aplicação. As requisições são todas encaminhadas ao mesmo cliente SaaS com uma configuração estática de provisionamento com uma única máquina atuando como servidor de aplicação.

A validação operacional foi dividida em dois grupos de cenários: i) cenários com

carga constante: requisições são enviadas ao servidor segundo uma taxa fixa; e ii) cenários com carga em rajadas: as requisições são submetidas em rajadas;

6.4.1 Grupo 01 - Carga constante

Esse cenário é o mesmo usado no refinamento do modelo, com tempos de disparo das requisições no cliente obedecendo a uma taxa constante segundo os parâmetros usados na tabela 4.1.

Como forma de validar o modelo de simulação implementado para o servidor de aplicação, submetemos o simulador a uma carga composta pelos tempos de chegada das requisições no servidor coletados na experimentação e demandas iguais ao inteiro mais próximo do valor da demanda média das requisições[1] (lembrando que o intervalo de confiança de 95% para a demanda média das requisições é 199,805 -+ 0,8284).

Coletamos o tempo de resposta médio para cada repetição de cada cenário. Como esse valor, em diferentes escalas, é usado para reportar essa métrica para os sistemas de provisionamento e *accounting* das aplicações, consideramos o valor significativo para o estudo. Pelo teorema do limite central, a distribuição amostral do tempo de resposta médio segue uma distribuição normal. Como os cenários de experimentação e simulação podem ser pareados dada a mesma configuração para tempos de chegada, nossa hipótese é de que a fração entre o tempo de resposta médio de uma execução no experimento controlado e na simulação deve ser próxima de 1, ou seja:

$$Hipótese : R = \frac{\bar{tr}_{ex}}{\bar{tr}_{sim}} = 1 \tag{6.1}$$

Realizando o teste t de Student para a relação R (equação 6.1) agrupando as 30 repetições de cada um dos 5 cenários, observando os valores para intervalo de confiança da média dos tempos médios de resposta na tabela 6.1, a hipótese é rejeitada. No entanto, a razão difere num fator pequeno comparado com o tamanho da requisição estudada, chegando a 6% de uma demanda de aproximadamente 200 milissegundos. Além disso, ao processar os tempos de resposta das requisições do cenário c_5, percebeu-se a aparição de requisições com demandas esperadas de cerca de 1/4 da demanda média, o

[1] Na implementação atual do simulador, cada requisição possui um número inteiro representando sua demanda em milissegundos.

que é impossível para a aplicação em questão executando em paralelo com requisições com demanda esperada muito acima da média. Uma possível explicação é que algumas requisições ao terminar de executar são postas numa fila separada enquanto esperam para ter sua resposta despachada para o usuário. Essas requisições não estão mais concorrendo pelo processamento, logo a sua demanda esperada deveria ser menor do que o que foi extraído considerando somente o tempo de resposta (que inclui o tempo que a requisição gastou na fila de espera pela resposta).

Cenário	Intervalo de confiança para a média de R
c_1	$[1.055276, 1.058919]$
c_2	$[1.052634, 1.056035]$
c_3	$[1.051511, 1.055370]$
c_4	$[0.9803917, 0.9841936]$
c_5	$[1.085816, 1.265618]$

Tabela 6.1: Média dos tempos de resposta médio para cada cenário com carga constante

6.4.2 Grupo 02 - Carga em rajadas

Para esse grupo de cenários, a cada 20 segundos, um número fixo de requisições foi disparado simultaneamente usando o programa `ab`[2]. Todas as requisições são similares e possuem demanda média, assim como no cenário anterior, com intervalo de confiança de 95% de 199.805 ± 0.8284. Para obtermos diferentes níveis de utilização média no cenário, controlamos a quantidade de requisições enviada de acordo com a tabela 6.2. A execução dessas configurações resultou em cenários com utilização média segundo a tabela 6.3.

[2]http://httpd.apache.org/docs/2.0/programs/ab.html

Cenário	c_1	c_2	c_3	c_4
Tempo entre disparo das rajadas	20 segundos			
Número de requisições por rajada	20	40	60	80
Duração	30 minutos			
Repetições	30			
Demanda da requisição na simulação (ms)	199.805 ± 0.8284			

Tabela 6.2: Parâmetros para o Grupo 02 da Validação

Cenário	Utilização Média
c_1	22.69158 ± 0.0008839
c_2	42.23434 ± 0.00095225
c_3	61.82422 ± 0.0015377
c_4	81.17136 ± 0.001944

Tabela 6.3: Utilização do Servidor para Carga em Rajadas (IC 95%)

É importante lembrar que, como a aplicação realiza somente processamento, a utilização média reportada equivalente a todo o período de duração do cenário é composta por períodos com utilização de 100% da máquina enquanto a aplicação está executando requisições e períodos de ociosidade enquanto a próxima rajada de requisições não chega.

Realizando uma investigação similar aos cenários para carga constante, observamos pelos valores na tabela 6.4 que, assim como no outro cenário, nenhum dos intervalos contem o valor 1 para a métrica R. Logo, não temos como afirmar que as médias dos tempos de resposta do experimento controlado e do modelo simulado são iguais. Chamamos atenção também para o a magnitude da razão calculada com a equação 6.1 cujos valores para os cenários de carga em rajadas são mostrados nas tabelas 6.4. Esses valores implicam que na média, os tempos de resposta coletados no experimento de medição são maiores que os produzidos pelo modelo de simulação. Isso mostra a necessidade de prover novos refinamentos ao modelo.

Cenário	Intervalo de confiança para a média de R
c_1	$[1.119145, 1.128059]$
c_2	$[1.056735, 1.061624]$
c_3	$[1.039008, 1.044716]$
c_4	$[1.028993, 1.034005]$

Tabela 6.4: Média dos tempos de resposta médio para cada cenário com carga em rajadas

Capítulo 7

Exemplos de Uso do Framework de Simulação

O framework proposto foi usado em dois contextos diferentes: i) avaliação heurísticas propostas para planejamento de capacidade de uma aplicação SaaS de comércio eletrônico (mais detalhes em [44]); e ii) avaliação de uma implementação do algoritmo de provisionamento dinâmico proposto por Urgaonkar et al.[3] num cenário de longo prazo com aplicação de SaaS de comércio eletrônico, que será detalhado nesse capítulo.

O algoritmo de Urgaonkar et. al.[3] realiza provisionamento dinâmico baseado em métricas técnicas coletadas da aplicação. A parte preditiva do algoritmo estima um percentil alto da taxa de chegada para o próximo período de avaliação (tipicamente 1 hora) e adquire máquinas, o suficiente para atender tal taxa de chegada, no nosso caso, num provedor de IaaS com mercado sob demanda nos moldes do Amazon EC2. Uma vez reconfigurada a infraestrutura um algoritmo reativo executa de tempos em tempos verificando se a taxa de chegada de requisições ultrapassou o limite estimado, ativando um sistema de controle de acesso em caso afirmativo e demandando uma reavaliação da quantidade de máquinas atualmente usadas.

O objetivo do experimento é mostrar como o framework pode ser estendido para ser usado na avaliação, usando o modelo de utilidade apresentado na Seção 5.1.5 com valores inspirados em provedores de SaaS e IaaS reais, de um algoritmo de provisionamento baseado em métricas técnicas num cenário de longo prazo.

Nessa avaliação usamos o algoritmo de Urgaonkar disponível no framework. A classe

`saasim.sim.provisioning.UrgaonkarProvisioningSystem` é uma extensão simples da classe `DynamicProvisioningSystem` do pacote `saasim.sim.provisioning` do SaaSim similar ao algoritmo proposto [3]. A configuração do algoritmo é realizada através de propriedades específicas definidas pela própria implementação sendo possível: i) ligar ou desligar as porções preditiva e reativa do algoritmo; ii) escolher o tipo de máquina usada no provisionamento; iii) limitar a janela de tempo (em dias) usada na predição; iv) limitar o percentil da distribuição construída na predição usada para caracterizar o pico da demanda na próxima hora; v) configurar o tempo entre as execução do algoritmo reativo, cujo valor é explorado nesse experimento; e vi) estimar o tempo de resposta (em milissegundos) ideal de uma requisição na camada. Os valores usados nesse experimento de simulação para essas configurações foram:

```
dps.urgaonkar.predictive=true
dps.urgaonkar.reactive=true
dps.urgaonkar.type=m1_small
dps.urgaonkar.windowsize=30
dps.urgaonkar.percentile=95
dps.urgaonkar.responsetime=800
dps.urgaonkar.reactive.threshold= # os valores usados foram 60000ms e 300000ms
```

Para a instanciação do modelo de utilidade definido na Seção 5.1.5, os valores utilizados para receita foram inspirados no modelo de cobrança mensal do BigCommerce[1], um provedor SaaS de sites de comércio eletrônico. Clientes interessados em criar um site para vendas de seus produtos escolhem um dos 5 planos disponíveis no portfólio. Para simplificar, incluímos abaixo apenas duas categorias, Bronze e Diamond, conforme os dados na tabela 7.1 e ajustando o arquivo de propriedades da seguinte forma:

```
saas.plan.number=2
saas.plan.name=diamond
saas.plan.priority=1
saas.plan.price=300.00
saas.plan.setup=0
```

[1] www.bigcommerce.com/pricing

```
saas.plan.cpu_lim=2147483647
saas.plan.ex_cpu=0
saas.plan.transfer_lim=46080
saas.plan.ex_transfer=0|0.005
saas.plan.storage_lim=3072
saas.plan.ex_storage=0.1

saas.plan.name=bronze
saas.plan.priority=1
saas.plan.price=25.00
saas.plan.setup=0
saas.plan.cpu_lim=2147483647
saas.plan.ex_cpu=0
saas.plan.transfer_lim=2048
saas.plan.ex_transfer=0|0.005
saas.plan.storage_lim=200
saas.plan.ex_storage=0.1
```

Usamos a proporção de 1 cliente Diamond para cada 32 clientes Bronze, de forma que a receita total gerada pelo conjunto de clientes Bronze fosse igual à receita total gerada pelos clientes Diamond. Para cada cliente foram espercificados, além do tipo de contrato estabelecido, o caminho para o arquivo de carga de trabalho com o uso da propriedade `saas.user.workload`.

O modelo de custo do provedor de IaaS foi configurado usando os parâmetros inspirados nos valores praticados pelo Amazon EC2 (ver tabela 7.2). Nesse experimento consideramos somente a aquisição de instâncias de um único tipo (`m1_small`) configurando da seguinte forma:

```
iaas.number=1
iaas.provider.name=amazon
iaas.provider.types=m1_small
iaas.provider.ondemand_cpu_cost=0.085
```

Plano	Parâmetro de Configuração	Valor	
Diamond	Mensalidade ($)	300.00	
	Taxa de SetUp ($)	0	
	Limite de Transferência de Dados	até 45GB	acima de 45GB
	Custo por Transferência Extra de Dados ($)	0	0.005
	Limite de Armazenamento de Dados	3072	
	Custo de Armazenamento Extra	0.1	
Bronze	Mensalidade ($)	15.00	
	Taxa de SetUp ($)	0	
	Limite de Transferência de Dados	até 2GB	acima de 2GB
	Custo por Transferência Extra de Dados ($)	0	0.005
	Limite de Armazenamento de Dados	200	
	Custo de Armazenamento Extra	0.1	

Tabela 7.1: Parâmetros de Receita: Provedor de SaaS

```
iaas.provider.reserved_cpu_cost=0.03
iaas.provider.reservationLimit=100
iaas.provider.oneYearFee=227.5
iaas.provider.threeYearsFee=350
iaas.provider.monit=0.15
iaas.provider.transIn=0
iaas.provider.costsTransIn=0|0
iaas.provider.transOut=1|10240|51200|153600
iaas.provider.costsTransOut=0|0.12|0.09|0.07|0.05
```

Parâmetro de Configuração	Valor				
Tipos de Instância	$m1_small$				
Custo da Instância/Hora ($)	0.085				
Custo do Monitoramento ($)	0.15				
Custo de Transferência de Entrada ($)	0				
Limites de Transferência de Saída	1GB	10TB	40TB	150TB	+150TB
Custo Transferência de Saída ($/GB)	0	0.12	0.09	0.07	0.05

Tabela 7.2: Parâmetros de Custo: Provedor de IaaS

A carga de trabalho foi gerada com o GEIST [46]. Estimamos os parâmetros de geração para adequar a quantidade de requisições no mês à quantidade de transferência máxima dos planos Bronze e Diamond. A tabela 7.3 contém os valores para taxas de chegada média usados para gerar o arquivo de carga de cada cliente nos diferentes dias da semana. Dessa forma a carga exibe um padrão similar ao encontrado em Arlitt et al.[10]. Valores para demanda foram distribuídos inspirados nos valores para camada de aplicação da aplicação de comércio eletrônico usado em Ranjan et al. [1].

Plano	Bronze	Diamond
Domingo	0.058	0.65
Segunda	0.058	0.65
Terça	0.058	0.65
Quarta	0.117	1.3
Quinta	0.058	0.65
Sexta	0.029	0.325
Sábado	0.029	0.325

Tabela 7.3: Taxa de chegada para os diferentes dias da semana (requisições/segundo)

A aplicação simulada possui uma única camada com balanceamento de carga implementando `Round Robin`. O limite para o tempo de resposta das requisições nessa camada é de 1 segundo (`saas.sla.maxrt`). Uma vez compradas no provedor de infraestrutura, as máquinas demoram 150 segundos para iniciar na aplicação. A aplicação

foi configurada da seguinte forma:

```
saas.application.factoryclass=saasim.sim.util.SimpleApplicationFactory
saas.application.numberoftiers=1
saas.application.heuristic=ROUNDROBIN_U
saas.setuptime=150000
saas.sla.maxrt=3600000
```

Na simulação foram explorados dois parâmetros: i) a quantidade limite de máquinas que podem ser adquiridas simultaneamente no provedor IaaS (configurado com a propriedade `iaas.provider.ondemandLimit`; e ii) o intervalo de tempo entre as avaliações do algoritmo reativo definido pela propriedade `dps.urgaonkar.reactive.threshold` específica da implementação do algoritmo de provisionamento descrito anteriormente. O período simulado foi de 1 ano, equivalente a 12 cobranças mensais no modelo de utilidade implementado com as classes do pacote `saasim.cloud`.

Os 6 cenários foram simulados em paralelo e demoraram cerca de 20 horas numa máquina com configuração Intel Xeon X5550 com 8 núcleos de 2.26GHz e 20GB de memória principal.

Na figura 7.1 é mostrado o comportamento do modelo de utilidade com a variação da combinação dos parâmetros especificados acima. Como esperado, a quantidade limite de máquinas que podem ser adquiridas no provedor IaaS impacta na implantação das decisões tomadas pelo algoritmo de provisão dinâmica. Ao aumentar a quantidade limite de máquinas aumentamos o custo de execução da aplicação, já que mais máquinas são compradas suprindo a demanda de requisições em momentos de pico, reduzindo a penalidade paga por infração ao SLA para os usuários dos dois tipos. Por outro lado, diminuir o tempo entre as avaliações impacta na diminuição dos custos por minimizar o impacto de decisões erradas tomadas pelo algoritmo reativo.

Em relação ao SLA, os cenários com limite menor de máquinas no mercado sob demanda apresentam uma taxa de infração do SLA em torno de 10%, para ambos os conjuntos de usuários, como pode ser visto na figura 7.2. Essa porcentagem com o aumento do limite, caso em que a aplicação deixa de operar em contenção de recursos. Diminuir o tempo entre as avaliações do algoritmo reativo surpreendentemente impacta

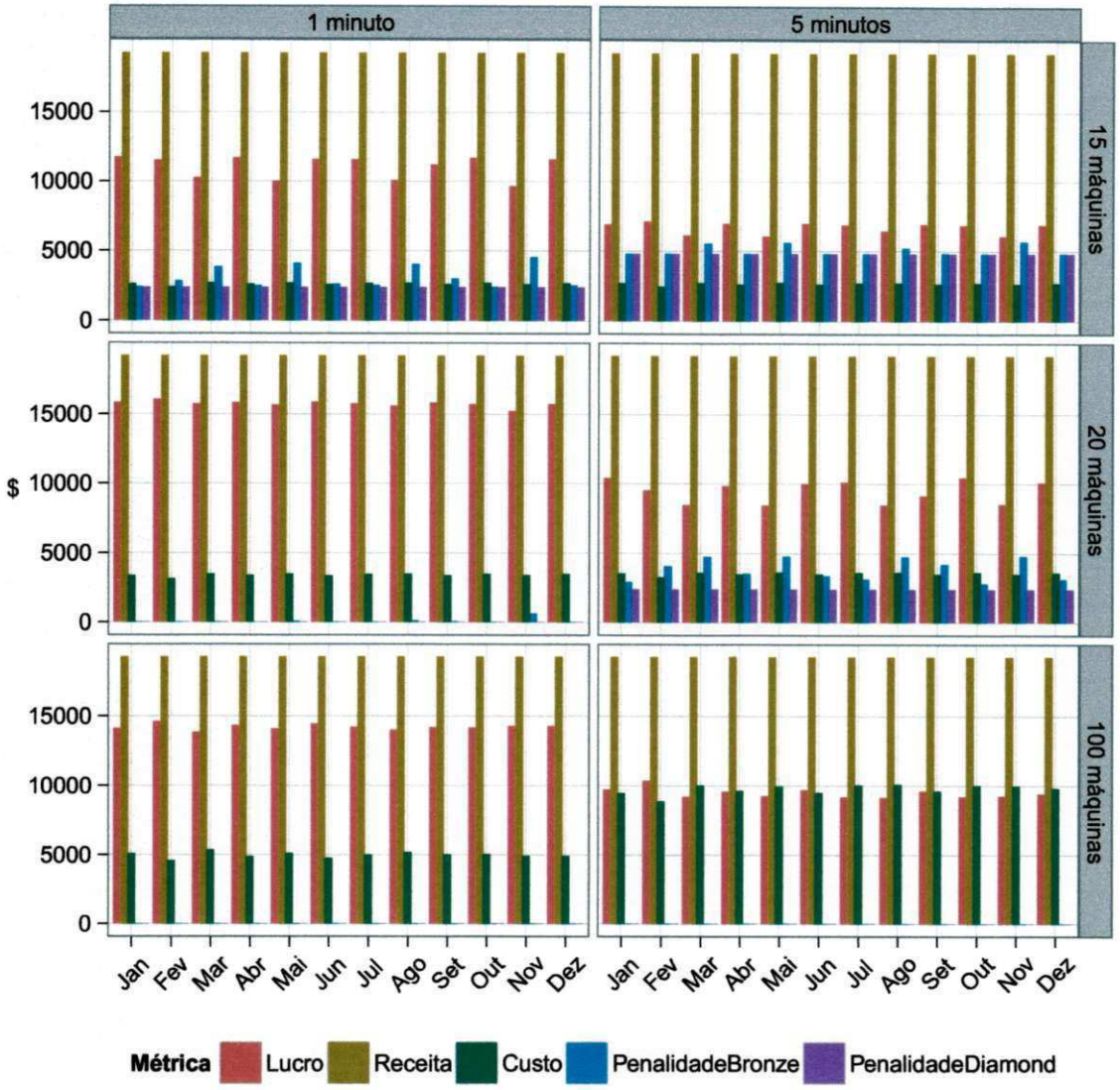


Figura 7.1: Resultados das Simulação: Modelo de Utilidade

na redução dessa porcentagem. Nesse caso, as decisões tomadas pelo algoritmo reativo com intervalo de 5 minutos, que levam a um custo maior de execução, acabam por provocar o superprovisionamento da infraestrutura no cenário com limite de 100 máquinas no mercado sob demanda. Dessa forma, uma quantidade menor de requisições infringem o SLA.

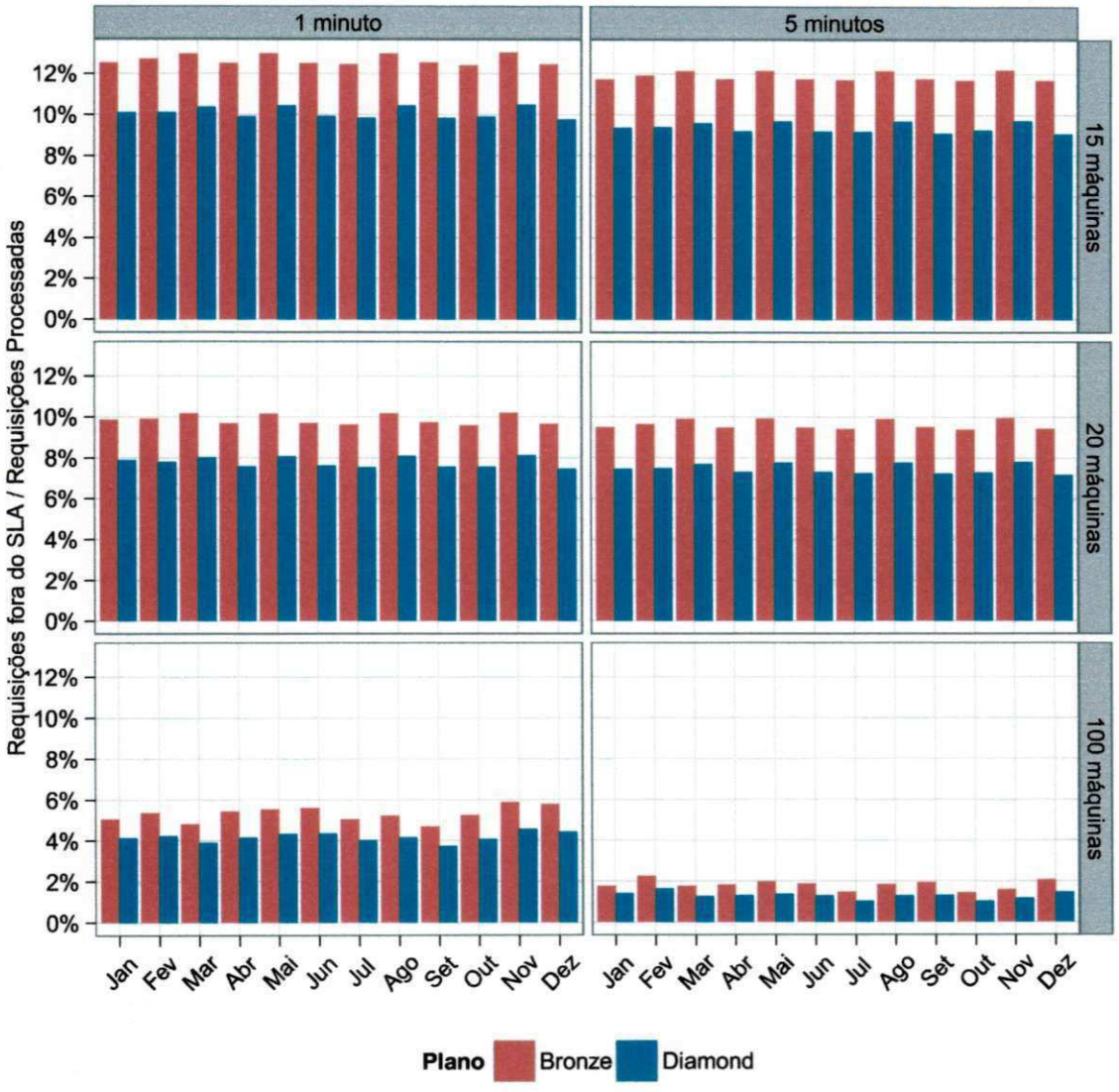


Figura 7.2: Porcentagem de requisições processadas que infringiram o SLA

Por fim, a atuação do controle de acesso nos cenários simulados pode ser vista na figura 7.3. Como esperado, os cenários com menor limite de quantidade de máquinas conseguem executar uma parcela menor das requisições (principalmente durante picos

na demanda). Por outro lado, o aumento do tempo entre as avaliações do algoritmo reativo impacta na demora em reagir a uma mudança na carga e faz com que o controle de acesso impeça uma quantidade maior de requisições de ser atendida.

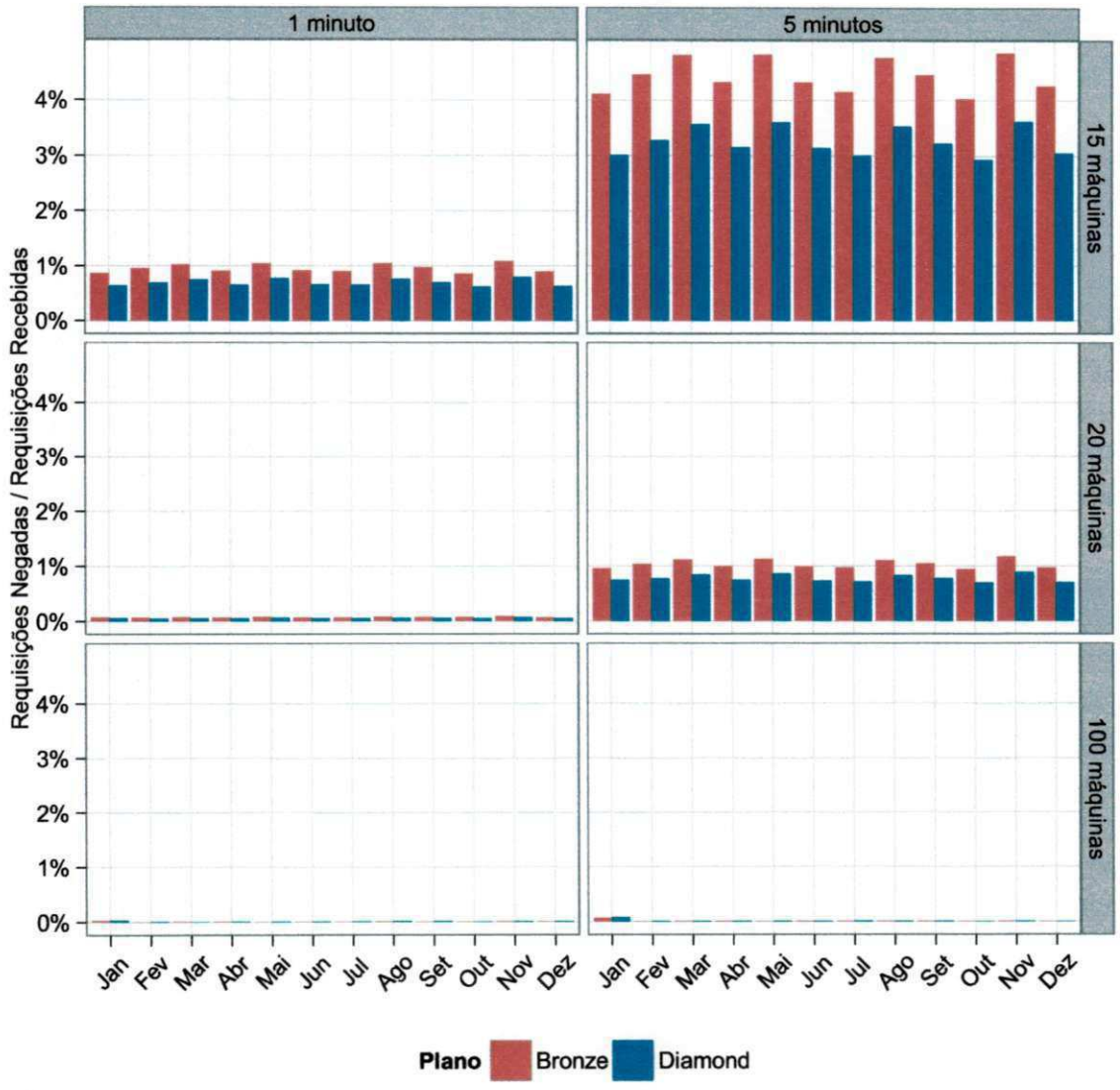


Figura 7.3: Porcentagens das requisições recebidas que foram rejeitadas pelo controle de acesso.

Os resultados acima mostram a importância do modelo de valoração implementado e dos resultados produzidos por ele na avaliação de um algoritmo no cenário de computação na nuvem. O experimento de simulação acima é um exemplo de como o framework apresentado neste trabalho de dissertação pode ser usado para explorar

características de algoritmos de provisionamento dinâmico na execução de aplicações Web no modelo SaaS.

Capítulo 8

Conclusões e Trabalhos Futuros

Este trabalho de dissertação apresentou o design e a implementação de um framework de simulação para aplicações Web de longa duração horizontalmente escaláveis. Tal framework pode ser usado para avaliar soluções para problemas de escalonamento e balanceamento de carga, provisionamento dinâmico, planejamento de capacidade, controle de acesso, entre outros. O design proposto inclui um fator de correção baseado na concorrência entre requisições escalonadas nas máquinas da infraestrutura executando a aplicação. Apresentamos também uma metodologia para a determinação desses valores com base em experimentos controlados de curta duração numa infraestrutura real.

O design proposto foi implementado na linguagem Java e uma metodologia de validação foi seguida para comparar o modelo de simulação implementado com um ambiente controlado de medição. Os dados coletados nesse experimento de validação mostram que o simulador obtém métricas com valores similares aos resultados coletados do sistema real controlado.

Por fim, apresentamos um exemplo de como configurar o simulador para avaliar um sistema de provisionamento dinâmico que atua sobre uma aplicação de uma única camada com um modelo de utilidade implementado usando o framework. Vimos que as métricas produzidas pelo modelo podem ser úteis na avaliação de um algoritmo de provisionamento dinâmico atuando sobre uma aplicação SaaS que executa em recursos adquiridos num provedor IaaS.

8.1 Limitações e Trabalhos Futuros

O modelo conceitual possui algumas limitações:

- O modelo conceitual considera que todos os clientes do provedor SaaS contratam o serviço no início do tempo e permanecem até o final da simulação. O simulador pode ser estendido para incluir o comportamento de entrada de novos clientes no sistema assim como saída de clientes existentes. Dessa forma, tanto taxas cobradas na adesão ao sistema como taxas cobradas pela saída prematura do sistema (antes de completar um ano de assinatura, por exemplo) podem ser incluídas no modelo de utilidade e simuladas com o uso do framework apresentado.

- O modelo de servidor de aplicação e, por consequência, o refinamento obtido com o uso de um fator de correção baseado na quantidade de requisições executadas em paralelo no momento da chegada de uma nova requisição são fortemente dependentes do tipo de aplicação considerado e do fato de modelar o processamento no servidor com uma única fila. Requisições que utilizam recursos como acesso ao disco, necessitam a inclusão desse tipo de recurso no modelo. A demanda das aplicações e serviços Web disponibilizados atualmente no modelo SaaS vão além do tipo de demanda modelado nesse trabalho. No entanto, esse trabalho é um passo inicial nessa direção e novas pesquisas com novos tipos de aplicações e modelos mais detalhados podem ser avaliados seguindo uma metodologia similar a apresentada aqui.

- A implementação padrão do simulador provê suporte apenas a aplicações de uma única camada que, nos experimentos realizados, for modelada como sendo a camada de aplicação. Como trabalho futuro pretende-se adicionar o comportamento de múltiplas camadas com um modelo mais específico para as características do servidor de cada camada.

Propomos alguns trabalhos futuros que se fundamentam nas limitações levantadas acima. Pode ser interessante para o pesquisador modelar chegada e saída de clientes do sistema. Aplicações com restrições mais fortes no SLA, e que não estejam sendo providas adequadamente geram a saída de clientes do serviço. No mercado de SaaS,

não é raro encontrar aplicações que modelam taxas que o cliente deve pagar ao desistir da aplicação. Da mesma forma, a chegada de novos clientes no sistema é modelada através do pagamento de uma taxa inicial de preparo do serviço. Adicionar ou remover clientes durante a simulação da aplicação produz uma perturbação em sistemas de provisionamento preditivos, tornando essa funcionalidade interessante.

O modelo de aplicação implementado modela unicamente a camada de aplicação. É comum encontrar trabalhos como Urgaonkar et al. [3] e Lee et al.[13] que consideram aplicações de várias camadas ou serviços compostos de vários outros serviços. Um trabalho futuro interessante seria incorporar esse comportamento ao modelo de simulação e avaliar aplicações desse tipo, usando os algoritmos propostos nesses trabalhos, para avaliar o comportamento do modelo de utilidade em cenários de longo prazo.

Por fim, para o servidor de aplicação utilizamos o modelo proposto por Menascé et al [45]. Nesse modelo, o recurso da máquina é modelado por uma única fila de processamento com política *Round Robin*. É interessante para outros tipos de aplicação considerar modelos mais refinados como o modelo de Redes de Fila com Camadas usado por Chi et al.[14].

Referências Bibliográficas

[1] S. Ranjan, J. Rolia, H. Fu, and E. Knightly, "Qos-driven server migration for internet data centers," in *Quality of Service, 2002. Tenth IEEE International Workshop on*, pp. 3 – 12, 2002.

[2] A. Chandra, W. Gong, and P. Shenoy, "Dynamic resource allocation for shared data centers using online measurements," in *Proceedings of the 11th international conference on Quality of service*, IWQoS'03, (Berlin, Heidelberg), pp. 381–398, Springer-Verlag, 2003.

[3] B. Urgaonkar, P. Shenoy, A. Chandra, P. Goyal, and T. Wood, "Agile dynamic provisioning of multi-tier internet applications," *ACM Trans. Auton. Adapt. Syst.*, vol. 3, pp. 1:1–1:39, Mar. 2008.

[4] V. Almeida and D. Menasce, "Capacity planning an essential tool for managing web services," *IT Professional*, vol. 4, pp. 33 – 38, jul/aug 2002.

[5] D. A. Menasce and V. Almeida, *Capacity Planning for Web Services: metrics, models, and methods*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 1st ed., 2001.

[6] Q. Zhang, L. Cheng, and R. Boutaba, "Cloud computing: state-of-the-art and research challenges," *Journal of Internet Services and Applications*, vol. 1, pp. 7–18, 2010. 10.1007/s13174-010-0007-6.

[7] S. Vijayakumar, Q. Zhu, and G. Agrawal, "Dynamic resource provisioning for data streaming applications in a cloud environment," in *Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on*, pp. 441 –448, 30 2010-dec. 3 2010.

[8] R. Buyya, S. Garg, and R. Calheiros, "Sla-oriented resource provisioning for cloud computing: Challenges, architecture, and solutions," in *Cloud and Service Computing (CSC), 2011 International Conference on*, pp. 1 –10, dec. 2011.

[9] D. A. Menascé, V. A. F. Almeida, R. Fonseca, and M. A. Mendes, "A methodology for workload characterization of e-commerce sites," in *Proceedings of the 1st ACM conference on Electronic commerce*, EC '99, (New York, NY, USA), pp. 119–128, ACM, 1999.

[10] M. Arlitt, D. Krishnamurthy, and J. Rolia, "Characterizing the scalability of a large web-based shopping system," *ACM Trans. Internet Technol.*, vol. 1, pp. 44–69, Aug. 2001.

[11] D. Villela, P. Pradhan, and D. Rubenstein, "Provisioning servers in the application tier for e-commerce systems," *ACM Trans. Internet Technol.*, vol. 7, Feb. 2007.

[12] X. Y. Wang, D. J. Lan, X. Fang, M. Ye, and Y. Chen, "A resource management framework for multi-tier service delivery in autonomic virtualized environments," in *Network Operations and Management Symposium, 2008. NOMS 2008. IEEE*, pp. 310 –316, april 2008.

[13] Y. C. Lee, C. Wang, A. Y. Zomaya, and B. B. Zhou, "Profit-driven service request scheduling in clouds," in *Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*, CCGRID '10, (Washington, DC, USA), pp. 15–24, IEEE Computer Society, 2010.

[14] R. Chi, Z. Qian, and S. Lu, "A heuristic approach for scalability of multi-tiers web application in clouds," in *Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS), 2011 Fifth International Conference on*, pp. 28 –35, 30 2011-july 2 2011.

[15] Z. Zhang, H. Wang, L. Xiao, and L. Ruan, "A statistical based resource allocation scheme in cloud," in *Cloud and Service Computing (CSC), 2011 International Conference on*, pp. 266 –273, dec. 2011.

[16] P. Magnusson, M. Christensson, J. Eskilson, D. Forsgren, G. Hallberg, J. Hogberg, F. Larsson, A. Moestedt, and B. Werner, "Simics: A full system simulation platform," *Computer*, vol. 35, pp. 50 –58, feb 2002.

[17] D. Menascé and V. Almeida, *Scaling for E-Business: Technologies, Models, Performance, and Capacity Planning*. Prentice Hall PTR, 2000.

[18] M. Arlitt and T. Jin, "A workload characterization study of the 1998 world cup web site," *Network, IEEE*, vol. 14, pp. 30 –37, may/jun 2000.

[19] J. Hellerstein, F. Zhang, and P. Shahabuddin, "An approach to predictive detection for service management," in *Integrated Network Management, 1999. Distributed Management for the Networked Millennium. Proceedings of the Sixth IFIP/IEEE International Symposium on*, pp. 309 –322, 1999.

[20] J. Guitart, J. Torres, and E. Ayguadé, "A survey on performance management for internet applications," *Concurr. Comput. : Pract. Exper.*, vol. 22, pp. 68–106, Jan. 2010.

[21] D. A. Menascé, D. Barbará, and R. Dodge, "Preserving qos of e-commerce sites through self-tuning: a performance model approach," in *Proceedings of the 3rd ACM conference on Electronic Commerce*, EC '01, (New York, NY, USA), pp. 224–234, ACM, 2001.

[22] A. Chandra, P. Pradhan, R. Tewari, S. Sahu, and P. Shenoy, "An observation-based approach towards self-managing web servers," *Comput. Commun.*, vol. 29, pp. 1174–1188, May 2006.

[23] B. Urgaonkar, G. Pacifici, P. Shenoy, M. Spreitzer, and A. Tantawi, "Analytic modeling of multitier internet applications," *ACM Trans. Web*, vol. 1, May 2007.

[24] B. Urgaonkar and P. Shenoy, "Cataclysm: Scalable overload policing for internet applications," *J. Netw. Comput. Appl.*, vol. 31, pp. 891–920, Nov. 2008.

[25] L. M. Vaquero, L. Rodero-Merino, J. Caceres, and M. Lindner, "A break in the clouds: towards a cloud definition," *SIGCOMM Comput. Commun. Rev.*, vol. 39, pp. 50–55, Dec. 2008.

[26] "Amazon Elastic Compute Cloud (Amazon EC2)." `http://aws.amazon.com/ec2`, Setembro 2012.

[27] "Cloud Hosting, Cloud Servers, Hybrid Hosting, Cloud Infrastructure from GoGrid." `http://www.gogrid.com`, Setembro 2012.

[28] "Google App Engine - Google Developers." `https://developers.google.com/appengine`, Setembro 2012.

[29] "Amazon ec2 instance purchasing options." `http://aws.amazon.com/ec2/purchasing-options`, Setembro 2012.

[30] P. D. Maciel, F. Brasileiro, R. A. Santos, D. Candeia, R. Lopes, M. Carvalho, R. Miceli, N. Andrade, and M. Mowbray, "Business-driven short-term management of a hybrid IT infrastructure," *Journal of Parallel and Distributed Computing*, vol. 72, pp. 106–119, Feb. 2012.

[31] P. Brereton and D. Budgen, "Component-based systems: A classification of issues," *Computer*, vol. 33, pp. 54–62, November 2000.

[32] M. Turner, D. Budgen, and P. Brereton, "Turning software into a service," *Computer*, vol. 36, pp. 38–44, October 2003.

[33] Y. V. Natis, "Introducing saas-enabled application platforms: Features, roles and futures," 2007.

[34] H. Liao and C. Tao, "An anatomy to saas business mode based on internet," in *Management of e-Commerce and e-Government, 2008. ICMECG '08. International Conference on*, pp. 215 –220, oct. 2008.

[35] "Globus Online." `http://www.globusonline.org`, Setembro 2012.

[36] "R-PHP." `http://dssm.unipa.it/R-php/`, Setembro 2012.

[37] J. Espadas, D. Concha, and A. Molina, "Application development over software-as-a-service platforms," in *Software Engineering Advances, 2008. ICSEA '08. The Third International Conference on*, pp. 97 –104, oct. 2008.

[38] R. Mietzner, A. Metzger, F. Leymann, and K. Pohl, "Variability modeling to support customization and deployment of multi-tenant-aware software as a service applications," in *Principles of Engineering Service Oriented Systems, 2009. PESOS 2009. ICSE Workshop on*, pp. 18 –25, may 2009.

[39] W. Iqbal, M. Dailey, and D. Carrera, "Sla-driven adaptive resource management for web applications on a heterogeneous compute cloud," in *Proceedings of the 1st International Conference on Cloud Computing*, CloudCom '09, (Berlin, Heidelberg), pp. 243–253, Springer-Verlag, 2009.

[40] H. Goudarzi and M. Pedram, "Multi-dimensional sla-based resource allocation for multi-tier cloud computing systems," in *Cloud Computing (CLOUD), 2011 IEEE International Conference on*, pp. 324 –331, july 2011.

[41] K. Katsalis, L. Tassiulas, and Y. Viniotis, "Distributed resource allocation mechanism for soa service level agreements," in *New Technologies, Mobility and Security (NTMS), 2011 4th IFIP International Conference on*, pp. 1 –6, feb. 2011.

[42] L. Wu, S. Garg, and R. Buyya, "Sla-based resource allocation for software as a service provider (saas) in cloud computing environments," in *Cluster, Cloud and Grid Computing (CCGrid), 2011 11th IEEE/ACM International Symposium on*, pp. 195 –204, may 2011.

[43] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. De Rose, and R. Buyya, "Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Softw. Pract. Exper.*, vol. 41, pp. 23–50, Jan. 2011.

[44] D. C. M. Maia, "Planejamento de Capacidade Dirigido a Negócios para Aplicações SaaS de Comércio Eletrônico," Master's thesis, Universidade Federal de Campina Grande, 2012.

[45] D. A. Menasce, L. W. Dowdy, and V. A. F. Almeida, *Performance by Design: Computer Capacity Planning By Example*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2004.

[46] K. Kant, V. Tewari, and R. Iyer, "Geist: a generator for e-commerce internet server traffic," in *Performance Analysis of Systems and Software, 2001. ISPASS. 2001 IEEE International Symposium on*, pp. 49 –56, 2001.

[47] R. Lopes, F. Brasileiro, and P. Maciel, "Business-driven capacity planning of a cloud-based it infrastructure for the execution of web applications," in *Parallel Distributed Processing, Workshops and Phd Forum (IPDPSW), 2010 IEEE International Symposium on*, pp. 1 –8, april 2010.

[48] J. Wilkes, *Utility Functions, Prices, and Negotiation*, pp. 67–88. John Wiley & Sons, Inc., 2009.

[49] R. Sargent, "Verification and validation of simulation models," in *Simulation Conference, 2005 Proceedings of the Winter*, p. 14 pp., dec. 2005.

Apêndice A

Arquivo de configuração do SaaSim

```
############################################################
#  Provisioning Configuration

planning.period=365

dps.heuristic=STATIC
dps.workload.parser=GEIST
dps.workload.pagesize=SECOND

############################################################
#  Aplicação

saas.application.factoryclass=saasim.sim.util.SimpleApplicationFactory
saas.application.numberoftiers=1
saas.application.heuristic=ROUNDROBIN_U
saas.application.startreplicas=10

saas.setuptime=150000
saas.sla.maxrt=3600000
```

```
machine.numberoftokens=200
machine.backlogsize=2147483647
machine.psquantum=1
machine.cf=false


############################################################
#  Provedor de Infraestrutura

iaas.number=1
iaas.provider.name=amazon
iaas.provider.types=m1_small
iaas.provider.ondemand_cpu_cost=0.085
iaas.provider.ondemandLimit=100
iaas.provider.reserved_cpu_cost=0.03
iaas.provider.reservationLimit=100
iaas.provider.oneYearFee=227.5
iaas.provider.threeYearsFee=350
iaas.provider.monit=0.15
iaas.provider.transIn=0
iaas.provider.costsTransIn=0|0
iaas.provider.transOut=1|10240|51200|153600
iaas.provider.costsTransOut=0|0.12|0.09|0.07|0.05

############################################################
#  Portfólio de planos da Aplicação SaaS

saas.plan.number=2
saas.plan.name=diamond
saas.plan.priority=1
saas.plan.price=300.00
```