Projeto de um broker de gerenciamento
adaptativo de recursos em computação em nuvem
baseado em técnicas de controle realimentado

Pedro Northon Nobile

SERVIÇO DE PÓS-GRADUAÇÃO DO ICMC-USP

Data de Depósito: 01/11/2012

Assinatura: _____

Projeto de um broker de gerenciamento adaptativo de recursos em computação em nuvem baseado em técnicas de controle realimentado

Pedro Northon Nobile

***Orientador:* Prof. Dr. Francisco José Monaco**

Tese apresentada ao Instituto de Ciências Matemáticas e de Computação - ICMC-USP, como parte dos requisitos para obtenção do título de Doutor em Ciências - Ciências de Computação e Matemática Computacional. *EXEMPLAR DE DEFESA*.

USP – São Carlos
Novembro de 2012

Ficha catalográfica elaborada pela Biblioteca Prof. Achille Bassi
e Seção Técnica de Informática, ICMC/USP,
com os dados fornecidos pelo(a) autor(a)

NN745p Nobile, Pedro Northon
p Projeto de um broker de gerenciamento adaptativo
de recursos em computação em nuvem baseado em
técnicas de controle realimentado / Pedro Northon
Nobile; orientador Francisco José Monaco. -- São
Carlos, 2012.
164 p.

Tese \ (Doutorado - Programa de Pós-Graduação em
Ciências de Computação e Matemática Computacional\) --
Instituto de Ciências Matemáticas e de Computação,
Universidade de São Paulo, 2012.

1. Computação em nuvem. 2. Controle realimentado.
3. Sistemas adaptativos. I. Monaco, Francisco José,
orient. II. Título.

Agradeço primeira e principalmente a Deus que me permitiu chegar até aqui para encarar mais um desafio na minha vida acadêmica. Foram muitos os obstáculos e muitas as provas nesse período que me fizeram, em alguns momentos, pensar que eu não conseguiria. Nada me ajudou tanto senão a minha fé em uma força maior que me manteve na direção e na busca pelos meus objetivos, ajudando a sobrepor cada barreira e a levantar-me sempre com mais força do que quando havia caído.

Sem dúvida, a maior felicidade de um homem reside na família, que fortificam as suas bases e tornam qualquer desafio possível. Por isso dedico muitos agradecimentos a toda minha família, em especial, agradeço a minha esposa Marcela que, muito paciente, suportou todas as minhas ausências e os meus mau-humores nesses tempos difíceis e foi o meu porto seguro, alguém que sempre me dá o amor, o carinho e o incentivo necessários para prosseguir com determinação e perseverança na busca dos meus objetivos.

Agradeço também as minhas duas grandes alegrias, de amor incontestável e inabalável, que são meus dois garotos João Pedro e Carlos Eduardo. Sempre avessos às responsabilidades de seu pai, provocavam o caos mais delicioso de ser vivido, uma bagunça que com avisos sorridentes avisam que era a hora do descanso e tornavam qualquer grande problema fácil de ser solucionado.

Agradeço a minha mãe, Luiza Mitie, e a meu irmão, Fábio Carlos, meus grandes incentivadores. É difícil medir a importância que as suas palavras de apoio e sua compreensão tiveram na minha trajetória, e por isso, também dedico meu trabalho a eles. Espero que tenham orgulho do que fiz e estou fazendo já que estamos tão longe e espero que tudo isso valha, de fato, toda essa distância.

Um homem sem amigos não é muito mais frágil e não posso me esquecer de todos que contribuíram comigo na minha jornada. Começo agradecendo o meu amigo e orientador, Prof. Dr. Francisco José Monaco, alguém muito importante na minha vida nesses últimos anos e que acreditou em mim mais do eu mesmo, alguém com quem compartilho os bons resultados desse projeto de pesquisa e que esteve comigo para os momentos felizes e de descontração.

Agradeço aos meus amigos de laboratório. Alguns que vão, outros que chegam, são muitos que dividem as mesmas angústias e felicidades diariamente, uma família solidária que se forma ao acaso e que fica no coração para sempre.

Agradeço também meus amigos de todas as horas, que são poucos, mas fiéis e muito leais. São aqueles que nos tiram do fundo do poço, cuidam de nós e fazem-nos sentir dispostos novamente. São esses que abandonam tudo para fazer aqueles programas necessários ao cansaço do corpo e ao descanso da mente e, assim, foram várias as pescarias que começaram quando o dia ainda era noite.

Não posso deixar de agradecer também aos funcionários e funcionárias da Programa de Pós-Graduação. É difícil encontrar pessoas que trabalham tão eficientemente e com tamanha produtividade que excedem suas obrigações do cargo e é isso que se encontra nessas pessoas. Sempre dispostos a ajudar, são pessoas que merecem meu reconhecimento e agradecimentos sinceros.

Computação em nuvem refere-se a um modelo de disponibilização de recursos computacionais no qual a infraestrutura de *software* e *hardware* é ofertada como um serviço, e vem se estabelecendo como um paradigma de sucesso graças a versatilidade e ao custo-efetividade envolvidos nesse modelo de negócio, possibilitando o compartilhamento de um conjunto de recursos físicos entre diferentes usuários e aplicações. Com o advento da computação em nuvem e a possibilidade de elasticidade dos recursos computacionais virtualizados, a alocação dinâmica de recursos vem ganhando destaque, e com ela as questões referentes ao estabelecimento de contratos e de qualidade de serviço. Historicamente, as pesquisas em *QoS* concentram-se na solução de problemas que envolvem duas entidades: usuários e servidores. Entretanto, em ambientes de nuvem, uma terceira entidade passa a fazer parte dessa interação, o consumidor de serviços em nuvem, que usa a infraestrutura para disponibilizar algum tipo de serviço aos usuários finais e que tem recebido pouca atenção das pesquisas até o momento, principalmente no que tange ao desenvolvimento de mecanismos automáticos para a alocação dinâmica de recursos sob variação de demanda. Este trabalho consiste na proposta de uma arquitetura de gerenciamento adaptativo de recursos sob a perspectiva do modelo de negócio envolvendo três entidades, focada na eficiência do consumidor. O trabalho inspira-se em técnicas de controle realimentado para encontrar soluções adaptativas aos problemas de alocação dinâmica de recursos, resultando em uma arquitetura de *broker* de consumidor, um respectivo protótipo e um método de projeto de controle para sistemas computacionais dessa natureza.

Abstract

Cloud computing refers to a computer resource deployment model in which software and hardware infrastructure are offered as a service. Cloud computing has become a successful paradigm due to the versatility and cost-effectiveness involved in that business model, making it possible to share a cluster of physical resources between several users and applications. With the advent of cloud computing and the computer elastic resource, dynamic allocation of virtualized resources is becoming more prominent, and along with it, the issues concerning the establishment of quality of service parameters. Historically, research on QoS has focused on solutions for problems involving two entities: users and servers. However, in cloud environments, a third party becomes part of this interaction, the cloud consumer, that uses the infrastructure to provide some kind of service to end-users, and which has received fewer attention, especially regarding the development of autonomic mechanisms for dynamic resource allocation under time-varying demand. This work aims at the development of an architecture for dynamic adaptive resource allocation involving three entities, focused on consumer revenue. The research outcome is a consumer broker architecture based on feedback control, a respective architecture prototype and a computer system feedback control methodology which may be applied in this class of problems.

Sumário

Resumo	iii
Abstract	v
Lista de Siglas	xiii
I Introdução	1
1 Introdução	3
1.1 Modelo de negócio de computação em nuvem	6
1.2 Gerenciamento eficiente de recursos	9
1.3 Proposta	12
1.4 Objetivo	13
1.5 Estrutura do documento	13
II Revisão	15
2 Revisão	17
2.1 Computação em nuvem	17
2.1.1 Características de computação em nuvem	19
2.2 O simulador CloudSim	24
2.3 Virtualização de recursos computacionais	27
2.4 Trabalhos relacionados	30
III Metodologia	41
3 Controle em sistemas computacionais	43
3.1 Considerações sobre a aplicação de controle em sistemas computacionais	44
3.2 Propriedades de interesse	47
3.3 Sistemas de controle de malha aberta e malha fechada	50
3.4 Modelagem de sistemas computacionais	54
3.5 Funções de transferência	62
3.6 Projeto de controle de sistemas discretos	69

3.6.1	Modelagem do sistema objetivo	69
3.6.2	Modelagem do sistema de malha fechada usando diagrama de blocos	71
3.6.3	Projeto do controlador	72
IV	Desenvolvimento, Experimentos e Resultados	77
4	Arquitetura do Broker de Consumidor	79
4.1	A Arquitetura	80
4.1.1	Monitor de máquinas virtuais	82
4.1.2	Monitor de requisições	86
4.1.3	Repositório de <i>Service Level Agreement (SLA)</i>	86
4.1.4	Monitor de rendimento	87
4.1.5	Despachador	89
4.1.6	Controlador e provisor de recursos	89
4.2	Projeto de sistema de controle proporcional-integral de terceira ordem com suavização de resposta por média móvel	91
5	Experimentos	101
5.1	Tempo de inicialização de máquinas virtuais	101
5.2	Configuração do <i>data center</i> e das máquinas virtuais utilizados na simulação	104
5.3	Avaliação de desempenho em regime transiente	106
5.4	Protótipo da arquitetura do <i>broker</i> de consumidor	116
5.4.1	Modelagem do sistema objetivo	122
5.4.2	Projeto do módulo controlador e provisor de recursos	126
V	Conclusão e Trabalhos Futuros	137
6	Conclusões e trabalhos futuros	139
6.1	Conclusões	139
6.2	Trabalhos futuros	142
VI	Referências	145
VII	Anexos	157
A	Dados utilizados para modelagem do sistema	159

Lista de Figuras

1.1	Interação de negócio e comunicação em um ambiente de computação em nuvem . .	7
1.2	Diferentes alternativas de alocação de recursos	8
1.3	Sistema de malha aberta	9
1.4	Sistema <i>feedforward</i>	10
1.5	Sistema <i>feedback</i>	10
2.1	Arquitetura de computação em nuvem.	22
2.2	Arquitetura em camadas do CloudSim	25
2.3	Possíveis combinações de escalonamento no CloudSim	26
2.4	Sistemas de controle de malha aberta	28
2.5	Arquitetura do <i>Feedback Control - Earliest Deadline First (FC-EDF)</i>	35
2.6	Principais componentes da arquitetura do ControlWare	36
2.7	Arquitetura genérica que combina filas e controle realimentado	37
2.8	Arquitetura para garantia de atrasos usando controle realimentado	37
2.9	Modelo do sistema para as classes de serviço ouro, prata e bronze	39
2.10	Diagrama de blocos do controlador para as classes ouro, prata e bronze	40
3.1	Sistema de controle de temperatura.	45
3.2	Saída de um sistema de controle estável.	49
3.3	Sistemas de controle de malha aberta	51
3.4	Sistema de controle de malha fechada	52
3.5	Sistema massa-mola.	55
3.6	Etapas para modelagem de caixa preta de um sistema dinâmico.	59
3.7	Representação de um polo no plano complexo.	66
3.8	Diagrama de blocos para um sistema de controle de malha fechada.	71
3.9	Diagrama de blocos representando os sistemas <i>feedforward</i> e <i>feedback</i>	72
4.1	Arquitetura proposta em alto nível.	81
4.2	Diagrama de estados que representa o ciclo de vida de uma máquina virtual	83
4.3	Fluxo da interação entre módulos da arquitetura e <i>Data center</i>	90
4.4	Sistema de controle <i>Proportional-Integral (PI)</i> de malha fechada com média móvel. .	91
5.1	Tempo de inicialização para diferentes condições de carga do servidor	103
5.2	Taxa de utilização para os tempos médios de processamento 35, 70 e 350	109
5.3	Número de instâncias de máquinas virtuais para os tempos médios de processamento 35, 70 e 350	111

5.4	Tempo médio de resposta para os tempos médios de processamento 35, 70 e 350	113
5.5	Número médio de requisições penalizadas para os tempos médios de processamento 35, 70 e 350 <i>Unidades de Tempo de Simulação (UTS)</i>	115
5.6	Curva de aumento de requisições penalizadas	117
5.7	Diagrama simplificado de classes do protótipo do <i>Broker</i>	119
5.8	Entrada de controle (número de máquinas virtuais) para modelagem do sistema objetivo	123
5.9	Resposta do sistema objetivo (taxa de utilização) à variação na entrada de controle	124
5.10	Comparação da resposta real medida com a resposta prevista pelo modelo	125
5.11	Diagrama de blocos do sistema de malha fechada com a função de transferência do sistema objetivo	129
5.12	Diferentes localizações para o polo p_3 de acordo com os valores do parâmetro do filtro de média móvel c	131
5.13	Taxa de utilização para $c = 0, 0.25, 0.50, 0.75$	132
5.14	Número de máquinas virtuais para $c = 0, 0.25, 0.50, 0.75$	133
5.15	Tempo médio de resposta para $c = 0, 0.25, 0.50, 0.75$	134
5.16	Quantidade de requisições penalizadas para $c = 0, 0.25, 0.50, 0.75$	134

Lista de Tabelas

3.1	Transformada Z dos principais sinais.	63
3.2	Propriedades da transformada Z	64
5.1	Resumo das requisições penalizadas para 35, 70 e 350 UTS	116
5.2	Valores de R^2 , <i>Root Mean Square Error (RMSE)</i> e correlação entre entrada e erro residual para o modelo estimado	126
5.3	Diferentes valores de c , p_3 , K_p e K_i	130
A.1	Dados usados na simulação.	159

Lista de Siglas

API Application Programming Interface

AR Autoregressive

ARMA Autoregressive Moving Average

ARX Autoregressive X

BIBO Bounded Input, Bounded Output

DMR Deadline Miss Ratio

EDF Earliest Deadline First

FC-EDF Feedback Control - Earliest Deadline First

IaaS Infrastructure as a Service

LTI Linear Time Invariant

MA Moving Average

MIMO Multiple Input, Multiple Output

MIPS Millions of Instructions Per Second

PaaS Platform as a Service

PD Proportional-Derivative

PI Proportional-Integral

PID Proportional-Integral-Derivative

QoS Quality of Service

RMSE Root Mean Square Error

SaaS Software as a Service

SASO Stability, Accuracy, Settling time, maximum Overshoot

SISO Single Input, Single Output

SLA Service Level Agreement

UTS Unidades de Tempo de Simulação

VMM Virtual Machine Monitor

Parte I

Introdução

Introdução

Computação em nuvem (*cloud computing*) refere-se a um modelo de disponibilização de recursos computacionais no qual a infraestrutura de *software* e *hardware* é ofertada e utilizada na forma de serviço através de uma rede de comunicação. Esse novo modelo vem se estabelecendo como um paradigma de sucesso na indústria em função de características vantajosas associadas ao investimento inicial de aquisição, custo de gerenciamento e manutenção (tanto corretiva quanto evolutiva), e escalabilidade sob demanda, quer para adquirir temporariamente mais capacidade, quer para dispensar aquela ociosa.

Esse cenário é o resultado da evolução ao longo das últimas décadas, em que ambientes de computação distribuída caminharam de máquinas paralelas para *clusters* de computadores, onde processos são alocados sobre unidades de processamento independentes e homogêneas, em termos de *hardware* e *software*, interconectados por meio de uma rede local dedicada de latência constante (Moura e Silva e Buyya, 1999). O contínuo avanço dessa tecnologia e seu emprego no tratamento de problemas cada vez mais complexos conduziram à ampla adoção de aglomerados computacionais, os quais passaram a serem interconectados a fim de prover maior capacidade computacional de modo compartilhado, dando origem, assim, a uma nova arquitetura de computação distribuída denominada grade computacional (*grid computing*) (Foster e Kesselman, 2004).

Ambientes de grade são caracterizados por recursos heterogêneos e geograficamente distribuídos, interconectados por meio de redes com latência variável. Seu desenvolvimento foi originalmente dirigido por aplicações científicas que são geralmente associadas a computação intensiva ou de alto desempenho. Modelos e tecnologias para sua implementação e emprego mais eficientes foram desenvolvidos, dando origem a arquiteturas com várias camadas, desde a infraestrutura física e lógica, até o nível de recursos e de aplicações de alto nível (Foster et al., 2008). Ao passo que

um número crescente de organizações começou a contar com grandes quantidades de máquinas em seus parques computacionais, essa perspectiva trouxe à realidade o conceito de computação utilitária (*utility computing*), em que recursos computacionais podem ser alugados e utilizados sob demanda, conforme a necessidade.

Concomitantemente, uma outra tecnologia que já vinha se desenvolvendo veio ao encontro da computação utilitária no tempo certo para potencializar o impacto do modelo de computação como serviço: a virtualização de sistemas computacionais. A virtualização é uma tecnologia que abstrai os detalhes do *hardware* e fornece recursos virtualizados para aplicações de alto nível. Um servidor virtualizado é comumente chamado de máquina virtual, e apresenta ao usuário ou aplicação a aparência de uma máquina física, distinta das outras máquinas virtuais que compartilham o mesmo *hardware*. Isso aumenta a flexibilidade do provedor de recursos que pode oferecer máquinas virtuais sob demanda alocadas em *hardware* de alta capacidade. Se usualmente nem todas as máquinas virtuais são operantes o tempo todo, respeitado um fator de demanda, o provedor pode atender a um número maior de usuários do que faria com máquinas físicas dedicadas. Além disso, é possível criar estratégias de precificação diferenciadas baseadas no tempo de utilização e potência alocada pela máquina virtual.

O Monitor de Máquina Virtual (*Virtual Machine Monitor (VMM)*) é o programa que está encarregado do gerenciamento das máquinas virtuais que compartilham os recursos reais. Essa vantagem se baseia na capacidade de partilha de recursos computacionais a partir de *clusters* de servidores para atribuir dinamicamente recursos virtuais para aplicações que precisam de recursos sob demanda, originando um modelo de negócio de oferta de serviços que foi denominado computação em nuvem, o qual se desenvolve em diversas modalidades de produtos, desde a disponibilização de infraestrutura, servidores virtuais e até aplicações (Vaquero et al., 2008). A designação tradicional de um provedor de serviços de computação em nuvem é a conceituação de centro de dados (*datacenter*), como um fornecedor de sistemas de computação virtualizados. Esse novo modelo de computação expandiu-se para além da utilização científica abrangendo praticamente todos os segmentos de aplicações em prestação de serviços na Internet.

Dentre as modalidades de oferta de recursos em computação em nuvem 3 são as principais formas:

- ***Infrastructure as a Service (IaaS)***: O consumidor contrata uma infraestrutura pura de *hardware* podendo escolher a capacidade da configuração (*i. e.* frequência de processamento, quantidade de memória, quantidade de armazenamento, taxa de transferência e quantidade de dados transmitidos), qual sistema operacional será executado e *softwares* estarão disponíveis arbitrariamente. O consumidor tem controle total da infraestrutura virtual contratada, mas não tem controle sobre a localização ou os recursos físicos de fato. É, por exemplo, o tipo de serviço oferecido pela Amazon EC2¹.

¹<http://aws.amazon.com/ec2/>

- **Platform as a Service (PaaS):** Geralmente usada para desenvolvimento colaborativo, o consumidor contrata uma plataforma com um conjunto de ferramentas específicas (*i. e.* controle de versões, ferramentas de comunicação e disponibilização de aplicações), usando as linguagens de programação suportadas pelo provedor, como é oferecido pela Google Apps², por exemplo. Nesse tipo de serviço o consumidor não tem acesso direto a configuração dos recursos físicos ou das camadas mais baixas de software.
- **Software as a Service (SaaS):** O consumidor contrata a utilização de uma aplicação que está hospedada e em execução numa nuvem, por exemplo, serviços oferecidos por um provedor de acesso a Internet. Esse é o tipo de serviço mais comum e, muitas vezes, o consumidor sequer tem ideia de que o serviço contratado faz parte de uma nuvem. Assim como em *PaaS*, nesse classe de serviço o consumidor também não tem acesso às camadas mais baixas da infraestrutura, por exemplo, Gmail³.

Duas propriedades chaves subjacentes ao conceito de computação em nuvem, e que tornam o modelo atrativo no contexto tecnológico atual são: ubiquidade e elasticidade.

Ubiquidade refere-se ao acesso a qualquer lugar e a qualquer instante, e corresponde ao conceito de pervasividade e das facilidades proporcionadas pela abrangência global da Internet, pelas redes sem fio e dispositivos móveis. A conveniência de armazenar arquivos na nuvem e ter acesso a eles a partir do computador pessoal, *smartphone* ou *tablet*, em casa, no trabalho ou em viagem está rapidamente assumindo o *status* de necessidade ordinária, seja para consulta de informações, para suprir capacidade limitada de armazenamento de dispositivos móveis ou embarcados, seja para utilização de ferramentas de *software* a partir de qualquer dispositivo.

Elasticidade, por sua vez, remete diretamente ao conceito de eficiência, tanto do ponto de vista econômico quanto energético, em consonância à perspectiva da preservação ambiental e desenvolvimento sustentável. A partir da virtualização de sistemas computacionais, possibilita-se o compartilhamento do conjunto de recursos físicos de uma máquina entre as múltiplas aplicações de múltiplos consumidores; provedores de nuvem fornecem servidores virtualizados a qualquer consumidor que se mostrar interessado em adquiri-los temporariamente, onerando-os mediante o tempo em que as capacidades contratadas forem utilizadas de fato, podendo dinamicamente aumentar ou diminuir o número de máquinas virtuais sob demanda.

A elasticidade, de particular interesse ao tema deste trabalho, refere-se ao gerenciamento eficiente de recursos, que pode ser implementado desde manual até automaticamente. A esse respeito, a crescente complexidade dos sistemas de computação distribuídos de larga escala hoje apresentados pelos modelos de computação em nuvem, atingem patamares sem precedentes cujo gerenciamento por meio de métodos tradicionais centralizados com supervisão manual, tende a ser cada vez menos praticável. A dimensão e complexidade dos recursos envolvidos, por exemplo, na infraestrutura de serviços de grande porte como redes sociais, mecanismos de busca e grandes aplicações

²<http://www.google.com/intl/pt-BR/enterprise/apps/business/>

³<http://www.gmail.com>

de comércio eletrônico, bancário, dentre outras, demanda estratégias de gerenciamento autônomo para atingir metas de desempenho, confiabilidade, qualidade de serviço e eficiência econômica e energética.

O desenvolvimento dos ambientes de computação em nuvem, assim, introduz novos desafios tecnológicos relacionados ao uso eficiente de recursos distribuídos, considerando aspectos relativos à heterogeneidade, latência de acesso não determinístico, possibilidade de falhas de comunicação, ampla distribuição geográfica dos elementos de processamento e a escalabilidade sem precedentes (Foster e Kesselman, 2004). Nesse contexto, técnicas de gerenciamento convencionais, baseadas em intervenções manuais e interações com usuários, mostraram-se cada vez menos eficientes (Horn, 2001) frente ao aumento da escala e sofisticação dos *data centers*, motivando pesquisas com o objetivo de tornar o ambiente auto-gerenciável.

A respeito desses trabalhos, que recentemente crescem em número na literatura da área, dois aspectos abordados merecem observação: o modelo de negócio da provisão de serviço em nuvem e a estratégia de gerenciamento eficiente de recursos.

1.1 Modelo de negócio de computação em nuvem

Acerca do primeiro aspecto, cabe notar que computação em nuvem é essencialmente um modelo de negócios viabilizado por um conjunto de tecnologias, em especial *data centers* de grande escala e a virtualização de sistemas operacionais.

Em se tratando de um serviço, portanto, um dos requisitos chaves para a provisão de recursos em nuvem é a Qualidade de Serviço (*Quality of Service (QoS)*) definida com base em parâmetros de interesse do usuário e formulada em um Contrato de Nível de Serviço (*Service Level Agreement (SLA)*) que estabelece obrigações e compensações entre os contratantes, conforme o cumprimento de cláusulas de qualidade e condições de uso acordadas.

Historicamente, pesquisas em qualidade de serviço estavam concentradas na relação binária entre usuários e servidores, principalmente *Web*. Modelos formais, especificações de *SLA*, ontologias, estratégias de negociação e técnicas de otimização e balanceamento de recursos tem sido formulada sob o paradigma de interação entre duas entidades: servidor e usuário.

A consolidação da computação em nuvem como uma alternativa para obtenção de capacidades computacionais tem, contudo, rearranjado os modelos de comunicação e de negócio entre provedores, consumidores e usuários finais, como pode ser visto na Figura 1.1, onde uma terceira entidade passou a fazer parte desse ambiente: o consumidor de serviços em nuvem.

Nesse cenário bastante comum que representa o contexto deste projeto de pesquisa, os provedores de nuvem dispõem de *data centers* de ampla capacidade computacional e fornecem infraestrutura, na forma de servidores virtualizados, para consumidores que desejam hospedar algum tipo de aplicação. A entidade consumidor refere-se aqui àquela que contrata serviços de provedor (*i. e.* infraestrutura) e usa os recursos contratados para disponibilizar uma aplicação (*e. g.* uma

aplicação *Web*) para seus usuários.

Figura 1.1 — Interação de negócio e comunicação em um ambiente de computação em nuvem

Por sua vez o consumidor, que detém a aplicação, precisa obter junto ao provedor a infraestrutura necessária para disponibilizá-la aos seus usuários. Mais que isso, o consumidor deve garantir que a *QoS* oferecida ao usuário final se mantenha dentro de níveis estipulados no contrato, mesmo sob variações repentinas e imprevisíveis da demanda. Na outra ponta do negócio, os usuários acessam a aplicação entregue, hospedada na nuvem, e esperam ser atendidos dentro dos parâmetros de qualidade acordados.

O objetivo de provedores de nuvem enquanto provedores de capacidades computacionais é entregar garantias de desempenho aos seus consumidores enquanto otimizam a utilização dos recursos no *data center* para reduzir custos com a operacionalização, à medida que os consumidores buscam diminuir seus custos com infraestrutura e manter a quantidade de penalizações em níveis mínimos possíveis. Não obstante essa lógica de interação, na prática muitas pesquisas concentram-se basicamente na abordagem de soluções para a garantia de qualidade de serviço experimentada por usuários de nuvem ou focam em problemas exclusivamente dos provedores (*i. e.* diminuição do consumo de energia, economia de recursos), poucos trabalhos abordam os problemas dos consumidores desses serviços.

Armbrust et al. (2010) estão entre os que ressaltam essa questão, formalizando um cenário desse tipo, no qual provedores de nuvem fornecem serviços do tipo *IaaS*, e cada consumidor do provedor se torna um provedor de serviços do tipo *PaaS* ou *SaaS*, enquanto que o usuário final é a entidade que consome de fato esses serviços. Apesar da relevância comercial, esse não é o cenário visto na maioria dos trabalhos de pesquisa que, geralmente, tendem a abordar apenas os problemas que envolvem duas entidades (*i. e.* provedor e consumidor). Ainda segundo os autores, provedores de *SaaS*, nesse caso a entidade consumidor, vem recebendo menos atenção que os usuários de *SaaS*.

Como uma de suas direções de contribuição, consoante ao exposto, este trabalho foca principalmente na entidade consumidor que, num cenário comercial, busca sempre a maior rentabilidade

possível com a entrega da aplicação face ao cumprimento dos contratos de prestação de serviços com os usuários e com o provedor. Essa rentabilidade recai principalmente sobre três fatores: **o custo**, o quanto o consumidor paga ao provedor de nuvem pela utilização dos recursos virtualizados que hospedam sua aplicação; **o ganho**, o quanto o consumidor arrecada dos usuários finais sempre que os parâmetros de *QoS* forem cumprido; e **as penalizações**, o quanto o consumidor deixa de arrecadar sempre que o contrato junto ao usuário for desrespeitado.

Para garantir a rentabilidade esperada é necessário que o consumidor tenha maior ganho e menores custo e penalizações possíveis. Do ponto de vista do consumidor, o gerenciamento de desempenho e a provisão dinâmica das capacidades computacionais disponíveis é imprescindível para atingir objetivos de rentabilidade. A Figura 1.2 ilustra duas situações extremas na quais a demanda por recursos varia com o tempo e os recursos são alocados de forma fixa (*i. e.* superestimado e subestimado). Ambas são contrastadas com uma alocação adaptativa, interessante ao rendimento do consumidor quando há variação da demanda.

Figura 1.2 — Diferentes alternativas de alocação de recursos

De fato, a alocação dinâmica de recursos é uma realidade que vem ganhando destaque desde as grades computacionais e passou a estar definitivamente em foco com a computação em nuvem. A provisão de capacidades computacionais dinamicamente segundo a demanda está intimamente ligada a existência de transações financeiras entre as partes. Não se pode abordar computação em nuvem sem abordar como acontece a negociação entre as entidades envolvidas e, por isso,

diversos trabalhos defendem que a alocação dinâmica de recursos computacionais compartilhados deve obedecer a mecanismos que sejam orientados ao mercado de computação (Lai et al., 2005; Buyya et al., 2009a; Jiang et al., 2009; Auyoung et al., 2004; An e Lesser, 2010; Lewis et al., 2009; Wolski, 2001).

1.2 Gerenciamento eficiente de recursos

Com respeito ao segundo aspecto, o gerenciamento de recursos em computação em nuvem, consideradas as condições realísticas do cenário atual, corresponde a efetivos desafios: *data centers* em escala sem precedentes, difícil previsibilidade de carga, desempenho não determinístico em função de características intrínsecas, concorrência entre processos dentre outros fatores, tornam a elaboração de políticas de alocação de recursos um problema não trivial.

A despeito das conquistas em pesquisa na área, o estudo da prática na indústria é ainda muitas vezes aquém das possibilidades. Em muitos casos, em sistemas de produção, o gerenciamento de recursos se resume ao planejamento de capacidade estática a partir de estimativas de demanda, possivelmente aprimorada com dados de variações sazonais. A menos que a demanda preste-se a esse tipo de estimativa — o que significa que seja constante ou não sofra alterações não antecipadas pelo conhecimento do passado — incorre-se no inerente risco de ineficiência por superdimensionamento pessimista, ou no descumprimento de especificações de qualidade de serviço por subdimensionamento otimista. Usualmente um equilíbrio de custo-benefício é procurado de forma a minimizar as perdas dadas pela função utilidade. Em linguagem técnica na área de sistemas de controle trata-se de um processo de malha aberta, virtualmente sem auto-adaptabilidade, como mostra Figura 1.3

Figura 1.3 — Sistema de malha aberta

Em utilizações mais sofisticadas, além do conhecimento das mudanças sazonais, variações na carga de trabalho imposta ao sistema podem ser antecipadas mediante o conhecimento de padrões na entrada. Séries temporais desempenham um papel importante na previsão de demanda ao antecipar eventos característicos, usualmente aprendidos a partir de análise de dados passados.

De qualquer modo, o que há de comum nessas técnicas é que essencialmente, existe um mecanismo adaptativo que reage a mudança de carga: detectando-se a variação na demanda, antecipam-

se seus potenciais efeitos por meio de medidas corretivas que minimizam as alterações de desempenho. Por exemplo, ao detectar um padrão indicativo de um súbito aumento da taxa de chegada de novas requisições, é possível alocar mais recursos para absorver a sobrecarga prevista. Em linguagem técnica em sistemas adaptativos é o que se chama de controle *feedforward*, como ilustrado na Figura 1.4.

Figura 1.4 — Sistema *feedforward*

Um controle *feedforward* reage a uma mudança na entrada e toma medidas pré-estabelecidas. Se as medidas pré-estabelecidas forem bem projetadas, a estratégia funciona; caso contrário, ou se variações dos parâmetros internos do sistema alterarem as condições que validam as medidas mitigadoras, haverá uma diminuição na capacidade do sistema em reagir à perturbação, da qual o mecanismo adaptativo, do modo como descrito na Figura 1.4, não toma ciência.

Alternativamente, um paradigma diferente em sistemas adaptativos é dado pelo princípio de controle realimentado. Basicamente, a diferença está no fato de que, em lugar de monitorar variações na perturbação na carga de trabalho, monitora-se a saída do sistema, isto é, o efeito da perturbação, como mostra a Figura 1.5.

Figura 1.5 — Sistema *feedback*

A ação corretiva do mecanismo de *feedback* tende a ser mais robusta no sentido de que, enquanto a saída não estiver no nível desejado, o mecanismo corretivo continua ativo.

Por um lado um mecanismo *feedforward* detecta a perturbação e tenta evitar que ela tenha efeito alocando ou desalocando recursos conforme uma regra pré-estabelecida (que precisa estar correta para que o reajuste seja efetivo). Por outro lado o mecanismo *feedback* permite que a perturbação chegue a saída; mas tão logo a tendência seja detectada, é capaz de corrigir o sistema e retorná-lo ao

ponto de operação, independente de uma regra de correção pré-estabelecida: enquanto os recursos não forem suficientes para anular a variação de carga, mais recursos são alocados e vice-versa (com um controlador propriamente projetado).

Em diversas situações, picos de carga podem ocorrer intempestivamente, por exemplo, sempre que novas funcionalidades do sistema se tornam populares ou quando um novo recurso é implantado. Como resultado, aumentos bruscos de carga podem ocorrer em diferentes locais a qualquer momento. A fim de lidar com mudanças imprevisíveis na demanda do sistema, um esquema de alocação automática de recursos é primordial para manter a *QoS* do serviço em níveis adequados (Buyya et al., 2010).

O conceito de computação autônômica, introduzido pela IBM (Horn, 2001) propõe um modelo abstrato de requisitos e propriedades de sistemas computacionais capazes de se auto-gerenciar em diversos níveis: auto-configuração, auto-recuperação, auto-otimização e auto-proteção. Arquiteturas e algoritmos específicos para atender a esses requisitos de auto-adaptação são específicos de cada aplicação e admitem distintas abordagens. Métodos centralizado de auto-gerenciamento, onde existem entidades únicas, ainda que redundantes, que concentram em si a responsabilidade e os dispositivos para coordenar a operação de sistemas distribuídos de grande escala, se por um lado são conceitualmente mais simples e construtivamente mais práticos, sofrem por outro lado dos obstáculos à escalabilidade, um desafio inerente às dimensões de grandes *data centers* e mesmo confederações de *data centers* (Buyya et al., 2010). Assim, tem havido justificado interesse em algoritmos distribuídos de auto-gerenciamento aplicáveis aos requisitos da computação autônômica. Dentre eles, métodos de otimização não determinísticos baseado em inteligência artificial e técnicas evolutivas que exploram auto-organização e auto-regulação via propriedades emergentes de sistemas distribuídos auto-adaptativos, como por exemplo, diversas abordagens bio-inspiradas em inteligência de enxames (Champrasert e Suzuki, 2006; Champrasert et al., 2012; Jiang et al., 2011).

Um fato que já vem sendo propriamente apontado na literatura (Hellerstein et al., 2005) é que o cerne da computação autônômica repousa nos princípios da teoria de controle realimentado, onde o contínuo monitoramento de uma variável de saída é comparado ao valor de referência desejado, sendo a discrepância entre ambos utilizada como sinal de auto-correção do próprio sistema. Tanto o advento da computação autônômica como conceito em si, bem como substancial fração da comunidade de pesquisa dedicada a seu estudo, tem focalizado seus objetivos sob a perspectiva de arquiteturas de sistemas e modelos abstratos de requisitos e propriedades funcionais e extra-funcionais. Contrastando com esse cenário, as ferramentas de análise e síntese de dispositivos auto-adaptativos desenvolvido ao longo de várias décadas na teoria de controle, que de fato correspondem às técnicas operacionais para síntese desses dispositivos, são ainda timidamente difundidas e empregadas aos domínios da computação.

A teoria de controle tem se desenvolvido como um ramo proeminente nas engenharias e ciências naturais, e conta com um rico arcabouço de ferramentas de modelagem matemáticas para descrever o comportamento de sistemas dinâmicos em termos da resposta à diferentes estímulos,

e para verificar propriedades de estabilidade, observabilidade de estados, e controlabilidade. Seu instrumental analítico oferece extensiva fundamentação para o projeto de estratégias de controle aplicáveis a sistemas lineares, não lineares, contínuos e discretos. A aplicação da teoria de controle em arquiteturas de sistemas computacionais para otimização de desempenho ou confiabilidade é, todavia mais recente e consideravelmente menos explorada que em outros domínios. Tem sido empregada, ainda que menos expressivamente, por exemplo, para clusterização dinâmica de processadores paralelos em problemas de particionamento de recursos, e para formatação de *throughput* em mecanismos de controle de congestão em redes de comunicação (Lu et al., 2001; Abdelzaher et al., 2003; Lu et al., 1999).

Particularmente, o ferramental analítico e de simulação desenvolvido no bojo de sua teoria dispõe de poderosos métodos para análise do comportamento transiente de sistemas dinâmico e compreender, por exemplo, os fatores que influenciam a dimensão do impacto na variável de saída, causadas por perturbações externas, o tempo de convergência e a qualidade da estabilização em torno do patamar de regime estacionário. Trata-se, portanto da perspectiva prática na engenharia de sistemas autonômicos e que subsidiam a realização de suas concepções arquiteturais. No caso de algoritmos de balanceamento de carga, isso significa prever o nível de desbalanceamento máximo causado por flutuações na carga de trabalho e variações nas capacidades do *data center*, bem como projetar sistematicamente estratégias de controle para reduzir esses impactos e acelerar a convergência, dadas restrições de diversas naturezas técnicas e econômicas.

1.3 Proposta

O argumento de pesquisa deste trabalho consiste na proposta de uma arquitetura de gerenciamento adaptativo de recursos sob a perspectiva do modelo de negócio com três entidades, focado na eficiência do consumidor (cliente *IaaS/PaaS*, fornecedor *SaaS*), baseado em uma estratégia de controle realimentado. A abordagem proposta difere-se do controle de malha aberta pela propriedade de adaptabilidade, e do controle tipo *feedforward* pelo fato de substituir a antecipação à variação de carga pela autorregulação a partir do monitoramento do desvio entre a qualidade de serviço efetiva e aquela acordada no *SLA*. Em lugar do ajuste empírico dos parâmetros do sistema de auto-gerenciamento, a Teoria de Controle, na qual se baseia o projeto do controle tipo *feedback*, fornece ferramental analítico para projetar-se sistematicamente o controlador a partir das especificações de desempenho desejadas. Uma das características da Teoria de Controle são as técnicas para se conhecer as propriedades dinâmicas do sistema alvo, o que permite compreender seu comportamento transiente e modificar tais propriedades em função de especificações de projeto, tais como máximo desvio de saída e tempo de assentamento em torno do valor de operação desejado.

1.4 Objetivo

Propor uma abordagem de implementação de uma arquitetura de computação em nuvem segundo o modelo de negócio focalizado no consumidor, em um modelo de usuário-consumidor-provedor, utilizando um mecanismo adaptativo de alocação de recursos baseado em controle realimentado, a partir das técnicas de análise e projeto inspiradas em Teoria de Controle.

Hipóteses:

- A despeito das dificuldades em se adaptar as técnicas de controle clássicas para sistemas computacionais, em especial, em computação em nuvem, é viável sua aplicação.
- Ganhos não negligenciáveis em *QoS* podem ser obtidos pela especificação de métricas de desempenho dinâmicas, as quais podem ser atingidas pelo projeto segundo técnicas de Teoria de Controle clássicas.
- É possível obter uma aproximação válida do sistema de computação em nuvem, considerado neste trabalho, por um sistema linear invariante no tempo.
- É possível mensurar o desempenho do sistema e comparar com aquele obtido utilizando-se outras abordagens de gerenciamento adaptativo de recursos.

Contribuições

- Uma arquitetura que implementa o modelo de negócios usuário-consumidor-provedor em sistemas de computação em nuvem.
- Avaliação das vantagens e limitações de técnicas de gerenciamento adaptativo de recursos baseado em controle realimentado.
- Um protótipo que implementa o modelo de negócios e mecanismos de gerenciamento de recursos.
- A sistematização de um método aplicável a problemas dessa natureza.
- Avanço na introdução e adaptação da teoria de controle em sistemas de computação.

1.5 Estrutura do documento

O restante deste documento está organizado da seguinte maneira:

- No capítulo 2 apresenta uma visão breve sobre os conceitos de computação em nuvem, bem como os tipos de serviços mais comuns, a virtualização de sistemas computacionais e ferramentas envolvidas e, por fim, é feita uma revisão da literatura atual mostrando o estado da arte relacionada ao tema de pesquisa e trabalhos relacionados que inspiraram o desenvolvimento deste.
- No capítulo 3 dá uma breve visão da teoria de controle e das considerações acerca das características de sistemas de computação, direcionando para aplicação e adaptação de técnicas de controle realimentado para sistemas de computacionais.
- No capítulo 4 é apresentada a arquitetura proposta nesse trabalho detalhando cada um dos módulos que a compõem. Este capítulo também traz um método de projeto de controle de malha fechada de sistemas computacionais desenvolvido neste trabalho.
- No capítulo 5 são explicados os experimentos realizados para validação da arquitetura, bem como é apresentado uma outra contribuição deste trabalho que é um protótipo de *broker* desenvolvido para uma das ferramentas de simulação mais utilizadas para computação em nuvem.
- No capítulo 6 são apresentados conclusões sobre os resultados obtidos e perspectivas futuras da continuidade desta pesquisa.
- As referências bibliográficas utilizados e os anexos marcam o fim deste documento.

Parte II

Revisão

Revisão

Este capítulo elenca os conhecimentos fundamentais para o entendimento deste projeto de pesquisa. As próximas seções tratam com suficiência os conhecimentos abordados e trazem as definições das técnicas e terminologia utilizadas. O capítulo está organizado da seguinte forma: a primeira seção aborda o conceito de computação em nuvem; a segunda seção faz uma breve explanação sobre as principais considerações que devem ser observadas no desenvolvimento de sistemas de controle para sistemas computacionais; a terceira seção aborda as funções de transferência para sistemas discretos, bem como extrai as propriedades de interesse em projetos de sistemas discretos e dá base para o entendimento do projeto de controle desenvolvido neste trabalho; por fim, a última seção busca agrupar técnicas e métodos que possam guiar o projeto de controle de sistemas computacionais.

2.1 Computação em nuvem

O rápido desenvolvimento de novas tecnologias para processamento e armazenamento têm feito com que os recursos de *hardware* se tornem cada vez mais acessíveis e com maiores capacidades. Na contramão desse avanço, há um desperdício desses recursos que acabam, na maior parte das vezes, sendo subutilizados e permanecem ociosos grande parte da sua vida útil ou até que sejam considerados ultrapassados. Tirando proveito dessa subutilização, paradigmas de computação distribuída como grades computacionais foram altamente explorados nos últimos anos. Mais recentemente, a computação em nuvem¹ surge como um novo paradigma e uma alternativa de consumo que vem ganhando destaque rapidamente, tornando-se uma tendência tanto no meio

¹Tradução do termo em inglês *Cloud Computing*.

acadêmico quanto no mercado de computação. Novos cursos, ferramentas e até mesmo eventos científicos abordam o tema como forma de chamar a atenção para a revolução que se segue.

Esse novo paradigma é um avanço recente da tecnologia de informação onde tanto a infraestrutura quanto as aplicações são providas como serviços para usuários finais obedecendo um esquema de pagamento por uso (Calheiros et al., 2011). De acordo com Klems et al. (2009), a provisão sob demanda de serviços computacionais confiáveis e escaláveis, usando um esquema de custo que cobra os consumidores pelo uso atual de serviço, tem sido o objetivo da indústria e pesquisa de computação distribuída já há algum tempo e a computação em nuvem vem ao encontro desses anseios.

Muitas definições sobre o tema podem ser encontradas na literatura e mostram as diferentes visões dos seus autores sob óticas diferentes. Enquanto alguns definem a computação em nuvem de um aspecto mais técnico-científico, tentando encaixá-la como uma evolução natural da computação distribuída, outros preferem vê-la de um ponto de vista mais econômico, comparando seu impacto no mercado com tecnologias precedentes.

Mell e Grance (2011) dão uma visão bastante técnica do assunto e definem a computação em nuvem como um novo modelo que possibilita o acesso, conveniente e ubíquo, a um conjunto de recursos compartilhados, configuráveis e acessíveis através de uma rede de comunicação. Buyya et al. (2009b) dão uma definição mais ampla do termo e dizem que a computação em nuvem pode ser definida como um tipo de sistema paralelo e distribuído, consistindo de uma coleção de computadores virtualizados e interconectados, que são providos dinamicamente e apresentados como um ou mais recursos de computação baseado em um *SLA*², estabelecido por negociação entre o provedor de serviços e o consumidor.

Bégin et al. (2008) fazem uma comparação extensa entre computação em nuvem e grades computacionais analisando e destacando as principais diferenças em aspectos como custo, desempenho, escalabilidade e facilidade de uso. Já Klems et al. (2009) fazem uma comparação conceitual entre nuvem e grades computacionais com base nos padrões de utilização estabelecidos para cada tecnologia, ressaltando que, enquanto grades são usadas para processamentos com pequenas durações, nuvens tendem a ser utilizadas para processamento de aplicações com longos ciclos de vida. No entanto, observa-se que, dada a versatilidade com que os recursos computacionais são contratados junto a provedores de nuvem, o seu custo atraente ao consumidor e a facilidade em adquirir e liberar os recursos, nuvens também têm sido usadas por aplicações de processamento intenso de curta duração.

Ainda com relação a diferença entre grades computacionais e nuvens, uma diferença facilmente identificável está na virtualização de *hardware*, pouco empregada em grades, mas explorada mais intensamente na computação em nuvem. O uso intenso de virtualização exige uma camada a mais de abstração para suportar o ciclo de vida das máquinas virtuais e suas aplicações, provida por ferramentas que implementam gerenciadores de virtualização.

²Termo em inglês que significa contrato de prestação de serviço.

Para este trabalho a computação em nuvem é vista como uma nova forma de consumo de computação em que objetivos de negócio podem ser atingidos por meio do atendimento de requisitos computacionais e na qual recursos computacionais de *hardware* e *software*, suportados por alguma tecnologia de virtualização, são oferecidos como serviços, por grandes *data centers*, a consumidores que os contratam sob demanda e pagando apenas pelo período em que utilizam os recursos contratados.

2.1.1 Características de computação em nuvem

Vários autores têm tentado compor um conjunto de características indispensáveis que devem estar presentes para evidenciar um ambiente completo de computação em nuvem. Muitas dessas características são comuns, enquanto outras dependem mais do ponto de vista do autor.

Armbrust et al. (2010) enumeram três novos aspectos do ponto de vista comercial proporcionados pela computação em nuvem listados a seguir:

- **Recursos computacionais infinitos:** O provedor deve apresentar ao consumidor a ideia de que os recursos computacionais disponíveis são infinitos e que esses recursos podem ser alocados e desalocados dinamicamente, a qualquer tempo, sob demanda, e rápido o suficiente para atender a uma variação de carga repentina diminuindo, mas não eximindo, a relevância ou o impacto do planejamento inicial de capacidade.
- **Eliminação de comprometimento inicial:** A eliminação do comprometimento inicial dos consumidores de computação em nuvem diz respeito à quantidade de recursos financeiros que devem ser investidos no início de um novo negócio. Isso permite que empresas pequenas comecem a trabalhar com nuvens consumindo pouca capacidade inicial e expandindo-as conforme a necessidade com a maturidade do negócio.
- **Pagamento por uso:** Uma das principais vantagens do consumo terceirizado de recursos em relação à aquisição permanente é permitir que o consumidor pague pelos mesmos pelo uso em pequenos intervalos de tempo, liberando esses recursos quando não forem mais necessários e não pagando pelo tempo em que ficam ociosos.

Além de propor esses aspectos, Armbrust et al. (2010) fazem uma análise do estado da arte no mercado de computação e classificam como provedores de nuvem somente grandes *data centers* que dispõem grande capacidade computacional, deixando de lado os pequenos e médios que, segundo os autores, dadas as limitações físicas não são capazes de usufruir de economia de escala, principalmente referente ao consumo de energia.

Dada sua magnitude, grandes *data centers* tem de lidar constantemente com problemas bastante complexos que, em alguns casos, não sequer foram cientificamente explorados. A falta de soluções

conhecidas para alguns desses problemas acaba gerando reflexos no mercado. Dentre os problemas que são comuns em grandes *data centers* podem ser citados a alta utilização por uma variedade de consumidores com aplicações e requisitos diferentes e a necessidade manter uma multiplexação entre as cargas de trabalho impostas por essas aplicações de forma que não comprometa o negócio entre as partes envolvidas.

Em outro trabalho, Badger et al. (2011) enumeram outras cinco características essenciais ao modelo de computação em nuvem:

- **Auto-serviço sob demanda:** Unilateralmente, o consumidor pode prover um recurso computacional, tal como uma nova instância de máquina virtual, automaticamente quando necessário sem a necessidade de intervenção de uma pessoa do lado do provedor de recursos. O consumidor deve estar apto a modificar a configuração dos seus recursos sempre que julgar necessário sem que haja intervenção humana no provedor. Isso só é possível quando ferramentas de gerenciamento são disponibilizadas pelo provedor ao consumidor.
- **Acesso amplo a rede:** Os recursos devem estar acessíveis através de uma rede de comunicação de forma ubíqua para que possam ser acessados por qualquer tipo de dispositivo com capacidade computacional.
- **Conjunto de recursos:** Os recursos computacionais do provedor devem estar agrupados para servir vários consumidores ao mesmo tempo. Com máquinas virtuais de diferentes configurações sendo atribuídas dinamicamente segundo a demanda do consumidor. Há um senso de independência de localização no qual o consumidor não tem conhecimento ou controle da localização exata dos recursos do provedor.
- **Elasticidade rápida:** Recursos podem rápida e elasticamente providos, tanto para aumento da capacidade quanto para liberação dos recursos computacionais. O consumidor deve ter a ideia de que os recursos são ilimitados e que podem ser adquiridos em qualquer quantidade ao tempo que julgar necessário.
- **Serviços medidos:** Sistemas de nuvem controlam e otimizam automaticamente o uso dos recursos por meio de medições em algum nível de abstração considerado apropriado para um determinado tipo de serviço (por exemplo, armazenamento, processamento, taxa de transmissão e contas de usuários ativas). O uso de recursos pode ser monitorado, controlado e relatado provendo transparência para provedor e consumidor sobre a utilização dos recursos.

Algumas aplicações são encaixam-se perfeitamente na ideia de nuvem e são bons exemplos dos benefícios que a terceirização de serviços pode trazer como por exemplo, aplicações com alta variação de carga e requisitos de *QoS*. Para essas aplicações, a capacidade computacional pode ser aumentada sempre que houver um aumento da carga imposta e diminuída quando a carga também diminuir; Aplicações cuja a carga é desconhecida, por exemplo, um novo serviço de

um portal, como não se sabe qual será a repercussão então qualquer estimativa de compra de recursos pode ser sub ou superestimada. Contratar serviços em nuvem permite que os recursos sejam ajustados a carga e que, após algum tempo, se tenha um panorama real da carga característica dessa aplicação; Processamento em lote, como é o caso de transações bancárias, exigem uma quantidade de recursos para uso intenso, mas por um período curto. Esses recursos podem ser contratados junto a um provedor somente para o período em que são necessários, sem a necessidade de aquisição permanente.

A computação em nuvem tem criado uma nova forma de comercialização de *hardware* e *software* que agora passam a ser oferecidos aos seus consumidores como serviços, como mostra a figura 2.1. Frequentemente, os serviços oferecidos por provedores de nuvem são divididos em três classes:

- **IaaS:** O consumidor contrata uma infraestrutura pura de *hardware* podendo escolher a capacidade da configuração (*i. e.* frequência de processamento, quantidade de memória, quantidade de armazenamento, taxa de transferência e quantidade de dados transmitidos), qual sistema operacional será executado e *softwares* estarão disponíveis arbitrariamente. O consumidor tem controle total da infraestrutura virtual contratada, mas não tem controle sobre a localização ou os recursos físicos de fato. É, por exemplo, o tipo de serviço oferecido pela Amazon EC2³.
- **PaaS:** Geralmente usada para desenvolvimento colaborativo, o consumidor contrata uma plataforma com um conjunto de ferramentas específicas (*i. e.* controle de versões, ferramentas de comunicação e disponibilização de aplicações), usando as linguagens de programação suportadas pelo provedor, como é oferecido pela Google Apps⁴, por exemplo. Nesse tipo de serviço o consumidor não tem acesso direto a configuração dos recursos físicos ou das camadas mais baixas de software.
- **SaaS:** O consumidor contrata a utilização de uma aplicação que está hospedada e em execução numa nuvem, por exemplo, serviços oferecidos por um provedor de acesso a Internet. Esse é o tipo de serviço mais comum e, muitas vezes, o consumidor sequer tem ideia de que o serviço contratado faz parte de uma nuvem. Assim como em *PaaS*, nesse classe de serviço o consumidor também não tem acesso às camadas mais baixas da infraestrutura, por exemplo, Gmail⁵.

Em qualquer uma das classes de serviço, fica claro que há uma terceirização de responsabilidades, onde o consumidor, cuja prerrogativa é de que seu provedor será capaz de cumprir com o estabelecido em contrato, confia ao provedor a manutenção da infraestrutura, plataforma ou serviço contratado.

³<http://aws.amazon.com/ec2/>

⁴<http://www.google.com/intl/pt-BR/enterprise/apps/business/>

⁵<http://www.gmail.com>

Figura 2.1 — Arquitetura de computação em nuvem.
Adaptado de Zhang et al. (2010).

Tradicionalmente, quando uma organização entende que seus recursos computacionais estão defasados e já não são mais suficientes para atender a carga imposta, investe na aquisição de novos equipamentos com maiores capacidades. Esse processo que vai da compra à disponibilização das aplicações é dispendioso e pode demorar semanas, dependendo da complexidade na aquisição e configuração, inviabilizando o atendimento de necessidades mais urgentes. Além disso, uma necessidade emergente pode vir de um problema pontual, necessitando de uma melhora temporária, e não permanente, dos recursos disponíveis. O tempo em que esses recursos são úteis é curto e depende apenas do atendimento da demanda emergente, não sendo necessário posteriormente. Em situações como essa, a aquisição permanente de recursos computacionais não é interessante, mesmo quando viável, e a contratação de recursos temporários junto a um provedor de nuvem torna-se uma alternativa bastante adequada.

Com o advento da computação em nuvem, o consumidor pode contratar o acesso a máquinas com configurações robustas em pouquíssimo tempo e com custos relativamente baixo quando comparados com os valores de compra das mesmas. Todo o processo de contratação e gerenciamento pode ser realizado utilizando ferramentas simples, como um navegador *Web*, e de modo ubíquo, de qualquer dispositivo com acesso à Internet, sendo que o consumidor será onerado somente pelo tempo em que utilizou esses recursos e não terá mais que se preocupar com a manutenção, a degradação natural ou a reposição de componentes.

A possibilidade de alugar máquinas ao invés de comprá-las torna possível que desenvolvedores com uma ideia inovadora e pouco dinheiro, que não podem comprometer grandes quantias na

aquisição de equipamentos caros logo no início do negócio sejam empreendedores (Armbrust et al., 2010). Da mesma forma, pesquisas que resultam em grande quantidade de processamento por um curto período (por exemplo, codificação de material genético e criptografia de dados), mas não dispõem de financiamento para compra de computadores robustos, podem contratar a mesma força computacional de que necessitam a um custo mais baixo junto a provedores de nuvem. De certa forma, alugar capacidade computacional é uma alternativa que permite diminuir custos e aumentar os rendimentos, e o que se observa é uma tendência das organizações que necessitam expandir sua capacidade computacional em optar pela contratação de serviços de provedores de nuvem, dada a rapidez com que esses recursos estarão disponíveis e facilidade de liberá-los quando não são necessários sem a necessidade de pagar pelo tempo ocioso.

Os provedores de nuvem podem oferecer seus serviços livremente ou então restringir o grupo de usuários que terá acesso aos seus recursos. Com base nesse comportamento, a disponibilização de um serviço de nuvem pode ser dividido em quatro modelos básicos (Badger et al., 2011):

- **Nuvens públicas:** É o modelo mais próximo de usuários comuns. Nesse modelo provedores de nuvem vendem seus serviços ao público em geral para qualquer usuário que tenha interesse em contratá-los.
- **Nuvens privadas:** Toda a infraestrutura e serviços disponíveis na nuvem são disponibilizados para uma única organização, independente dos recursos serem próprios ou terceirizados.
- **Nuvens comunitárias:** Nesse modelo a nuvem é compartilhada entre diversas organizações que possuem interesses em comum. Da mesma forma que as nuvens privadas, os recursos podem ser próprios ou terceirizados.
- **Nuvens híbridas:** Em nuvens híbridas há a composição da infraestrutura por duas ou mais nuvens (independente do modelo de disponibilização) que são disponibilizadas como se fossem uma única nuvem.

Além desses, um outro tipo de nuvem tem sido explorado, as nuvens confederadas⁶. Uma nuvem confederada é a interação entre mais de um *data center* com o intuito de expansão das capacidades computacionais disponíveis temporariamente, formando uma rede em que os participantes são beneficiados.

Pesquisas em nuvem tem abordados problemas específicos de alguma das classes de serviços oferecidos e veem obtendo resultados significativo. No entanto, aplicações reais podem envolver mais de uma classe como como mostrado pela figura 1.1. Por exemplo, um consumidor contrata uma *IaaS* ou *PaaS* junto a um provedor de nuvem e utiliza essa o serviço contratado para hospedar uma aplicação. O consumidor de nuvem pode então prover *SaaS* a clientes que estejam interessados em utilizar a sua aplicação. Casos assim, apesar de muito comuns ainda são pouco estudados.

market oriented.

⁶Tradução do termo em inglês *federated clouds* ou *federation of clouds*.

2.2 O simulador CloudSim

Hoje em dia, é muito comum que empresas optem por hospedar novas aplicações em provedores de nuvem ao invés de adquirir uma infraestrutura própria, principalmente, quando se tratam de aplicações *Web* que devem estar disponíveis para milhares ou mesmo milhões de usuários, como ferramentas de desenvolvimento colaborativo e comunicação. Além dessas aplicações são muito comuns em nuvem aquelas que foram, por muito tempo, pertinentes às grades computacionais com uso intenso de processador por um curto período de tempo, como o processamento de imagens e sinais.

Provedores de nuvem têm se mostrado como alternativas confiáveis e custo-efetivas à aquisição de infraestrutura própria e têm provocado o êxodo de muitas aplicações dos servidores internos de algumas organizações para servidores virtualizados em *data centers*. Com essa nova forma de disponibilização de aplicações, surge um novo campo a ser explorado que incita pesquisadores na busca por melhorias e soluções para os problemas existentes.

Em uma situação ideal, tanto os testes de novas aplicações como os experimentos ligados a uma determinada pesquisa seriam executados em provedores reais de nuvem, mas isso nem sempre é possível ou viável já que habitualmente envolve a contratação de um provedor que incorre em custos. Por exemplo, quando um teste de uma aplicação envolve a escalabilidade dos recursos (*i. e.* há a variação na quantidade de recursos alocados) principalmente com aumento dos mesmos, é preciso considerar que também há um aumento do custo envolvido que podem tornar os testes financeiramente inviáveis (*e. g.* um serviço de rede social disponível a muitos usuários).

Analogamente, um experimento que precisa ser executado repetidas vezes para que os resultados coletados atendam aos requisitos estipulados no planejamento de experimentos (*e. g.* os resultados devem estar dentro de um intervalo de confiança) também tem o custo de alocar uma quantidade de recursos por várias vezes.

Uma alternativa aos provedores reais são os simuladores, como o CloudSim. O CloudSim é um *framework* bastante robusto que permite a criação de simulações de ambientes de nuvem pela extensão das classes já existentes na sua *Application Programming Interface (API)*. O CloudSim permite que ambientes de nuvem de larga escala compostos de vários nós sejam simulados com a implementação simples de um programa de simulação. A Figura 2.2 ilustra a arquitetura em camadas do CloudSim.

O CloudSim permite que simuladores de ambientes de nuvem que abstraíam todo a manipulação dos recursos físicos que é desempenhada pelas camadas mais baixas da Figura 2.2. Dessa forma, é possível implementar a arquitetura proposta para o *Broker* do consumidor sem que seja necessário adaptar o código do núcleo do *framework*.

Existem dois níveis de alocação de recursos previstos pelo CloudSim: nível de **máquinas virtuais**, que aloca as máquinas virtuais dentro das capacidades físicas existentes, e nível de **carga de trabalho**, que aloca uma determinada requisição para uma das instâncias de máquinas virtuais

Figura 2.2 — Arquitetura em camadas do CloudSim

Fonte: Calheiros et al. (2011)

disponíveis. Além dos níveis, existem duas políticas primárias de alocação de recursos para esse níveis:

- **Space-shared**: Na política *Space-shared*, quando um recurso é alocado para uma tarefa, o mesmo permanece alocado, sem preempção, até que a tarefa seja finalizada.
- **Time-shared**: Na política *Time-shared*, quando um sistema é ocupado, há o compartilhamento desse recurso entre várias tarefas que competem pela utilização dos recursos.

Alternando as políticas de alocação com os níveis são obtidas quatro combinações diferentes, como mostra a Figura 2.3. Nessa Figura é considerada uma única máquina física com dois núcleos de processamento que será ocupada por duas máquinas virtuais VM_1 e VM_2 que processam oito tarefas $T_1, T_2, T_3, T_4, T_5, T_6, T_7$ e T_8 .

A Figura 2.3a mostra um exemplo em que as máquinas virtuais são alocadas nas máquinas físicas com a política *Space-shared* e a carga de trabalho é alocada nas máquinas virtuais ligadas usando a mesma política. Nesse caso, VM_1 ocupa os dois núcleos da máquina física até t_1 . Enquanto VM_1 ocupava os núcleos, o primeiro núcleo foi ocupado pela tarefa T_1 e o segundo núcleo foi ocupado pela tarefa T_2 até que ambas terminassem e dessem lugar para as tarefas T_3 e T_4 . A partir do tempo t_2 , VM_1 encerrou e deu lugar a VM_2 . Enquanto na vez de VM_2 , as tarefas T_5 e T_6 ocuparam os núcleos virtuais inicialmente e depois de finalizarem deram lugar às tarefas T_7 e T_8 .

Figura 2.3 — Possíveis combinações de escalonamento no CloudSim

Fonte: Calheiros et al. (2011)

Na Figura 2.3b é ilustrado um exemplo no qual as máquinas virtuais são alocadas segundo a política *Space-shared*, mas as tarefas são alocadas segundo a política *Time-shared*. Nessa combinação, é possível perceber a mesma situação para as máquinas virtuais que foi ilustrada na 2.3(a), no entanto, quando VM_1 ocupa a máquina física, o primeiro núcleo é compartilhado entre as tarefas T_1 e T_2 , enquanto as tarefas T_3 e T_4 competem pelo segundo núcleo. O mesmo acontece quando VM_1 finaliza e dá lugar a VM_2 que tem o primeiro núcleo ocupado por T_5 e T_6 e o segundo núcleo ocupado por T_7 e T_8 .

A Figura 2.3c ilustra uma combinação na qual as máquinas virtuais são alocadas pela política *Time-shared* e as tarefas são alocadas a cada instância de máquina virtual usando *Space-shared*. Nesse caso, VM_1 e VM_2 não executam mais sequencialmente e competem desde o início pelos dois núcleos. No entanto, uma vez que uma das máquinas ocupa um núcleo, apenas uma tarefa executa no núcleo virtual até que finalize e dê lugar a outra tarefa.

A última combinação possível é mostrada pela Figura 2.3d que aplica a política *Time-shared* tanto para a alocação de máquinas virtuais quanto para

Uma limitação da ferramenta é a impossibilidade inicial de criar um simulador em que as requisições sejam submetidas em tempos distribuídos por uma função de probabilidade. Todo o processamento de tarefas é feito em lote e as requisições chegam todas ao *data center* no mesmo instante. A implementação da arquitetura de *Broker* desenvolvida neste trabalho expande o CloudSim e suporta que requisições sejam submetidas a qualquer tempo.

2.3 Virtualização de recursos computacionais

Apesar do uso repaginado que se vê atualmente, a virtualização é um conceito que foi experimentado pela primeira vez na década de 70 e que, desde então, ficou fora de foco dos holofotes da comunidade científica (Raj e Schwan, 2007). Recentemente, com o advento da computação em nuvem, a virtualização de sistemas computacionais voltou à tona com novas tecnologias e ferramentas que compõem a base dos serviços oferecidos pelos provedores de nuvem. Basicamente a ideia por trás do conceito de virtualização é compartilhar os recursos físicos de uma máquina entre diversas instâncias de máquinas virtuais, promovendo quase que uma multiprogramação no nível de sistemas operacionais.

A virtualização atual baseia-se no fato de que os avanços nas tecnológicos em *hardware* criaram uma infraestrutura que vai além da demanda que o *software* é capaz de promover na maior parte do tempo. Quase sempre as aplicações que executam num servidor são insuficientes para consumir suas capacidades computacionais durante a maioria do tempo em que o mesmo permanece ligado, fazendo com que processador, memória e outros dispositivos sejam subutilizados e permaneçam ociosos a maior parte do tempo. É óbvio que essa subutilização é uma forma de desperdício de investimentos e recursos computacionais que, possivelmente, tornar-se-ão obsoletos sem terem sido utilizados em toda sua capacidade. Logo, são necessárias formas de aproveitar melhor os recursos para diminuir as perdas financeiras e resolver o problema de subutilização, e a virtualização vem ao encontro desses anseios.

Virtualizar sistemas é permitir que mais de uma instância de máquinas virtuais, com configuração e sistemas operacionais distintos, operem sobre um mesmo *hardware* físico, com isolamento e segurança, garantindo que cada aplicação está comprometida apenas com a máquina virtual em que executa (Figueiredo et al., 2003). Nesse caso, uma camada adicional de *software* conhecida por *VMM*⁷ (Agesen et al., 2010) disponibiliza um *hardware* virtual que comporá as máquinas virtuais. Essa camada de software, *VMM*, executa como uma aplicação numa máquina física junto com um sistema operacional nativo, comumente chamado de hospedeiro, e a essa união entre *VMM* e sistema operacional hospedeiro, dá-se o nome de **hipervisor**. O hipervisor é responsável por multiplexar os recursos físicos entre o sistema operacional hospedeiro e os sistemas operacionais hóspedes, aqueles que executam sobre as máquinas virtuais, fazendo com que todo esse processo seja transparente. A figura 2.4 ilustra os dois tipos de hipervisores mais comuns: **tipo 1**, em que sistemas operacional hospedeiro e hipervisor são uma única camada de *software*; e **tipo 2**, em que hipervisor e sistemas operacional hospedeiro são disjuntos.

Em hipervisores do tipo 1, do ponto de vista da interação entre máquinas virtuais e *hardware*, esse hipervisor executa diretamente sobre os recursos físicos reais e tenta adaptar o uso ao máximo possível para reduzir a sobrecarga da virtualização. Para promover a virtualização usando hipervisores do tipo 1, é necessário que os sistemas operacionais hóspedes possam ser adaptados

⁷Em português, Monitor de máquina virtual.

Figura 2.4 — Sistemas de controle de malha aberta

Barham et al. (2003). Dessa forma, várias estruturas do sistema hospedeiro, como o escalonador de processos e o gerenciador de memória, são modificadas. Esse processo de adaptação torna a virtualização possível em hipervisores do tipo 1, no entanto, essas alterações acabam com o princípio de transparência e tornam o sistema hóspede específico para virtualização, inclusive, com alterações substanciais para diminuir o consumo de recursos. São exemplos de hipervisores desse tipo 1: Microsoft Hyper-V⁸, Citrix XenServer⁹ e VMware ESX Server¹⁰.

Já hipervisores do tipo 2 não necessitam de adaptação dos hóspedes e, dessa forma, sistemas operacionais completos sem qualquer modificação podem ser usados diretamente nesse caso. Uma vantagem desses hipervisores sobre o anterior é a facilidade de instalação, uma vez que o hipervisor é tido como um outro programa do sistema operacional hospedeiro. Essa facilidade, no entanto, tem um preço, que é não suportar uma grande quantidade de instâncias de máquinas virtuais, devido à sobrecarga gerada pela camada de software intermediária, inviabilizando o uso em ambientes de produção. São exemplos de hipervisores do tipo 2: Microsoft Virtual Server¹¹ e VMware Server¹². Além desses, outros tipos de hipervisores (*e. g.* Monolítico e Microkernel) podem ser encontrados em Tulloch (2010).

Sistemas virtualizados são altamente flexíveis, uma vez que é possível criar instâncias de máquinas com configurações distintas, desde que essas não ultrapassem das capacidades computacionais disponíveis num mesmo hipervisor. Avanços recentes fizeram com que os *VMMs* atuais conseguissem dar às instâncias virtuais o mesmo desempenho, ou próximo disso, experimentado por sistemas operacionais nativos. Além de diminuir os desperdícios de investimento e aumentar a utilização dos recursos físicos, uma outra grande vantagem da virtualização está na rapidez com que se é possível recuperar um sistema virtualizado de falhas. Por exemplo, no caso de falha de uma máquina virtual, uma outra instância pode ser iniciada rapidamente, tomando o lugar daquela

⁸<http://www.microsoft.com/HyperV>

⁹<http://www.xensource.com>

¹⁰<http://www.vmware.com/products/vsphere/esxi-and-esx/index.html>

¹¹<http://www.microsoft.com/windowsserversystem/virtualserver/>

¹²<http://www.vmware.com/products/server/overview.html>

que falhou. Outra possibilidade é num ambiente com vários servidores físicos, executando hipervisores, com várias instâncias de máquinas virtuais, no caso de falha de uma máquina física, as instâncias de máquinas virtuais em execução podem ser migradas para outros hipervisores que não falharam.

Assim como sistemas operacionais nativos, sistemas operacionais virtualizados também passam pelo processo de inicialização quando uma nova instância é ligada, isso significa que há um atraso entre o momento que a instância foi ligada até que a mesma esteja disponível, sendo que esse intervalo de tempo varia muito de caso-a-caso. Segundo Fito et al. (2010), em um serviço *IaaS*, uma máquina virtual pode levar até sessenta segundos desde o ligamento até estar disponível.

Apesar do avanços tecnológicos propiciados pela virtualização de sistemas, algumas questões que não dizem respeito à tecnologia, mas a sua aplicação, precisam ser resolvidas, por exemplo, ainda não há um consenso quanto a utilização de programas cujas licenças de uso sejam cobradas. Ainda não é certo qual a melhor forma de proceder quando uma instância virtual que contém sistemas operacionais e programas com licença pagas deve ser disponibilizada. Uma alternativa é deixar a responsabilidade pelas licenças para os consumidores (*e. g.* um consumidor contrata uma infraestrutura junto a um provedor e fica responsável pelos programas e licenças que serão disponibilizados nessa infraestrutura, inclusive nas suas réplicas, caso necessário), no entanto, isso pode fazer com que uma infraestrutura contratada junto a um provedor seja usada para disponibilizar cópias de programas não autorizadas. Por outro lado, (Armbrust et al., 2010) tem uma proposta interessante, na qual propõe que, assim como o *hardware*, as licenças de programas sejam cobradas pelo tempo em que o mesmo foi usado.

Independente do tipo de hipervisor utilizado, a virtualização de sistemas computacionais pode ser classificada em: **virtualização total**, **paravirtualização** e **virtualização assistida por hardware**.

A virtualização total é composta por técnicas de tradução binária e execução direta no *hardware*. Por motivos de desempenho, sempre que for possível, os programas virtualizados devem ser executados diretamente sobre os recursos físicos reais. Por motivos de estabilidade, a tradução binária deve acontecer sempre que a execução direta de instruções virtuais puderem causar algum tipo de instabilidade no sistemas (*e. g.* sobrescrever o valor de algum registrador que não deveria ter seu valor alterado). Nesse tipo de virtualização, todo o *hardware* é virtualizado e o *VMM* fornece às máquinas virtuais versões virtuais de todos os recursos físicos (*e. g.* dispositivos virtuais de entrada e saída e BIOS).

Na paravirtualização há um conjunto de chamadas, as *hypercalls*, que adicionam uma camada entre os sistemas operacionais virtualizados e o hipervisor com o intuito de diminuir a sobrecarga dos sistemas virtualizados e aumentar o desempenho dos mesmos. Para isso, é necessário que os sistemas virtualizados hóspedes sejam modificados e acessem as *hypercalls* por meio de *drivers*. Naturalmente, alguns sistemas operacionais não podem ser virtualizados por meio desse tipo de virtualização já que não é possível modificá-los, no entanto, a instalação de programas específicos nos sistemas hóspedes pode tornar possível a compatibilidades desses sistemas com hipervisores de paravirtualização.

Por fim, a virtualização assistida por *hardware* surgiu a partir do momento em que a virtualização de sistemas computacionais começou a ficar mais comum e principalmente a indústria de processadores começou a olhar mais de perto essa tecnologia. Com isso, processadores capazes de suportar nativamente a virtualização de vários sistemas operacionais começaram a ser produzidos e possibilitaram que uma nova forma de virtualização fosse implementada, na qual o *VMM* passou a executar com privilégios maiores que os suportados por um sistema operacional nativo.

Qualquer que seja o tipo de virtualização ou hipervisor empregados, o certo é que a virtualização de sistemas computacionais mudou os limites com que as capacidades computacionais eram empregados e tornou possível que um recurso (*i. e.* infraestrutura, plataforma ou programa) sejam adquiridos facilmente como um serviço. A virtualização tornou a computação em nuvem factível em suas diversas vertentes e provedores detentores de grandes *data centers* empregassem suas capacidades computacionais ociosas para fornecerem serviços aos mais diversos tipos de consumidores, reduzindo assim o desperdício e o prejuízo com uma infraestrutura subutilizada.

2.4 Trabalhos relacionados

Diversas pesquisas têm concentrado esforços na resolução de problemas que envolvem garantia de requisitos de *QoS* em sistemas distribuídos tanto para grades computacionais quanto para computação em nuvem. Esta seção dá um panorama dos trabalhos que estão sendo desenvolvidos e a forma como são propostas soluções para o problema proposto.

No mercado de computação, é cada vez mais comum a opção de consumidores pela terceirização de recursos em detrimento da aquisição de recursos próprios de *hardware* e *software*. Além da versatilidade propiciada pela terceirização de recursos, essa nova forma de aquisição tem se mostrado como uma alternativa custo-efetiva para o aumento momentâneo e repentino das capacidades computacionais permitindo que aplicações *Web* sejam disponibilizadas prontamente para seus usuários finais.

Pretensões até pouco tempo irreais, como o aumento imediato de recursos físicos sob demanda, agora são possíveis com a computação em nuvem. Grandes aplicações *Web* com múltiplas camadas já podem ser hospedadas em servidores virtualizados contratados de provedores de nuvens onde cada camada da aplicação está fisicamente dispersa por servidores virtuais que possuem a capacidade de aumentar e diminuir seus recursos a qualquer tempo.

Nessa nova abordagem onde *hardware* e *software* são elásticos, algumas questões substanciais vêm a tona. Novos temas começam a chamar a atenção e passam a ser questões fundamentais discutidas e exploradas por pesquisas em ciência da computação na busca de soluções viáveis que possam ser aplicada nesse novo paradigma.

Pesquisas recentes mostram grande interesse na busca por soluções de alocação dinâmica de recursos, principalmente, em situações onde há variação da carga de trabalho ou das características internas do sistema. As soluções consistem de técnicas que buscam encontrar, em um conjunto

de recursos computacionais compartilhados, uma forma de alocação das capacidades físicas para atender a expectativa de provedores de serviço, consumidores e usuários finais.

Do ponto de vista da granularidade dos recursos, a provisão dinâmica pode ser implementada de duas maneiras diferentes: por meio de recursos com granularidade **fin**a e recursos com granularidade **gross**a. Na primeira, trata-se de uma forma de alocação de recursos com aumento ou diminuição de capacidade daqueles já alocados prévia e individualmente. Alguns exemplos comuns de alocação de recursos com granularidade fina são:

- O aumento ou diminuição no número de instâncias de processos ou *threads* de um servidor de aplicação que já está em execução.
- A modificação, em tempo de execução, da quantidade de memória ou na capacidade de processamento sem a necessidade de reiniciar o sistema operacional.
- A alteração nos limites de armazenamento.
- A mudança nas taxas de transferência.

Por outro lado, os recursos também podem ser alocados por granularidade grossa, em que a alteração na quantidade de recursos não influencia diretamente os recursos que já estão sendo utilizados, e a alocação de um recurso específico (*e. g.* processador) implica na necessidade de alocar outros recursos (*e. g.* uma máquina virtual inteira). Exemplos comuns de granularidade grossa são:

- A inicialização ou desligamento de instâncias de máquinas virtuais.
- A alteração no número de servidores físicos dedicados a um propósito específico.

Em comparação com a alocação de recursos de granularidade grossa, a alocação dinâmica de recursos com granularidade fina envolve soluções mais complexas, já que para que esses recursos sejam alocados individualmente é necessário que o gerenciador de virtualização e os sistemas operacionais hóspedes utilizados suportem a elasticidade de recursos em tempo de execução. Ainda assim, mesmo que essa elasticidade seja possível, há uma quebra de transparência na alocação de recursos, visto que o sistema operacional hóspede deve estar ciente de que seus recursos são virtualizados. Em geral, seja com granularidade fina ou grossa, o objetivo é sempre o mesmo: garantir que a quantidade de recursos disponíveis sejam suficientes, e sempre que possível eficientes, para atendimento da demanda.

Na busca por soluções de provisão dinâmica de recursos computacionais, diversas técnicas, algumas naturais da computação e outras vindas de outras áreas da ciência, têm sido empregadas com sucesso. Dentre as soluções propostas, três têm sido explorada com maior ênfase e apresentam resultados significativos: **inteligência artificial**, **séries temporais** e **teoria de controle**.

Inomata et al. (2011) defendem que quantidades consideráveis de tempo e dinheiro são gastos para projetar, configurar, inicializar e monitorar recursos computacionais e que um dos objetivos da computação em nuvem, em um futuro muito próximo, é o gerenciamento e a atribuição de recursos automáticos aos seus consumidores e propõe um método baseado na carga de trabalho de máquinas virtuais para provedores de *IaaS*, com base nas condições estabelecidas pelo usuário.

Inteligência artificial é um termo demasiado genérico para expor uma única técnica. O que se quer dizer com esse termo é que diversas estratégias utilizadas atualmente para solucionar o problema de alocação dinâmica de recursos têm sido desenvolvidas por pesquisas na área de inteligência artificial, sendo mais comuns aquelas que utilizam lógica *fuzzy*, algoritmos de mineração de dados e aprendizado de máquina.

Os métodos de aprendizado de máquina mostram-se muito interessantes para soluções imediatas em que não há tempo hábil ou conhecimento que permitam extrair um modelo do sistema que consome os recursos. Essas técnicas permitem, inclusive, que abordagens não lineares sejam empregadas para implementar um gerenciador de recursos, responsável pela alteração dinâmica das capacidades computacionais.

Alguns métodos de mineração de dados permitem que um gerenciador de recursos seja posto em execução, sem a necessidade de qualquer treinamento em relação ao sistema objetivo. Isso significa dizer que um gerenciador robusto pode ser desenvolvido sem que qualquer informação especialista sobre o sistema objetivo e sua carga de trabalho. É claro que, como consequência, há que se prever a aparição de um erro aleatório que, em alguns casos, faz com o ação desempenhada pelo gerenciador de recursos seja inadequada.

Dentre as abordagens mais comuns de inteligência artificial utilizadas estão aquelas que se baseiam em lógica *fuzzy*, como na resolução de problemas de otimização (Xiong et al., 2011; Quiroz et al., 2009) de particionamento de capacidades (*i. e.* encontrar entre os recursos disponíveis a divisão de recursos que melhor atende a demanda).

A capacidade de adaptação de sistemas que utilizam abordagens *fuzzy* reduz a necessidade de conhecimento especialista e como consequência a modelagem do gerenciador se torna mais simples e reduz o tempo de projeto de uma solução. Essa redução da fase de projeto faz com que alguns trabalhos adotem técnicas *fuzzy* para configuração de parâmetros em controladores (Diao et al., 2002a,b).

Xu et al. (2008) propõem um sistema de gerenciamento em dois níveis para alocação de recursos virtuais individuais que utiliza tanto a teoria de controle como a modelagem *fuzzy*. No primeiro nível de gerenciamento é usada uma abordagem baseada em teoria de controle em que um controlador global é usado para reagir as alterações de carga da aplicação e determinar os recursos necessários para atendê-las. O controlador global é usado para que os recursos do *data center* sejam alocados aos servidores virtuais que vão, de fato, fornecer acesso a algum tipo de aplicação. No segundo nível é usado um controlador local, baseado em lógica *fuzzy*, para particionar da maneira mais adequada os recursos entre as instâncias de máquinas virtuais segundo características da carga.

Estratégias desenvolvidas para solucionar problemas de alocação dinâmica em grades computacionais podem ser adaptadas para a realidade das nuvens. Jiang et al. (2009) apresentam um método de alocação dinâmica de recursos, baseado em modelagem *fuzzy*, para resolver a adaptação de recursos em grades computacionais em situações em que várias aplicações apresentam diferentes requisitos de *QoS*.

Soluções baseadas em métodos de previsão estatística, principalmente as resultantes da análise de séries temporais, são uma alternativa para construção de gerenciadores inteligentes de recurso e tem sido bastante exploradas para resolver problemas de alocação dinâmica de capacidades computacionais (Fito et al., 2010; Chandra et al., 2003; Kusic e Kandasamy, 2007; Zhang et al., 2007; Levy et al., 2003). As séries temporais formam uma área de conhecimento da estatística com amplo conjunto de teorias, métodos e ferramentas matemáticas que auxiliam na análise e no projeto de previsores. São frequentemente usadas na indústria para solução de problemas de previsão de demanda e na análise de comportamentos financeiros na economia. Todavia, algumas soluções apresentam boa adequação para problemas computacionais.

Alguns trabalhos buscam identificar padrões de comportamento na carga de trabalho e, com base em algoritmos previsivos ou reativos, otimizar a provisão de recursos das máquinas virtuais. Zhang et al. (2007) propõem a provisão dinâmica de recursos com base na modelagem e análise de comportamento de aplicações por longos períodos de tempo. É realizado um monitoramento das aplicações para determinar fases de utilização. Em cada fase são analisadas similaridades e diferenças que possam levar a um padrão de comportamento com base em técnicas de *clustering*. Contudo, a provisão de recursos é baseada apenas no padrão de utilização e não leva em conta a carga que será imposta ao sistema.

Quiroz et al. (2009) utilizam modelos Box-Jenkins (Box e Jenkins, 2011) para criar uma abordagem descentralizada para particionamento de recursos em grades e nuvens que busca detectar, por meio de monitoramento, padrões e tendência da carga de trabalho para otimizar o uso dos recursos nas máquinas virtuais. A estratégia proposta também explora abordagens de mineração de dados para identificar padrões de carga e otimizar a provisão de recursos. Os autores propõem uma abordagem descentralizada na qual cada nó de processamento (*i. e.* máquina virtual) é responsável por uma classe de requisições identificada. Nobile et al. (2007) também propõem o uso de modelos de séries temporais para modelagem de tráfego de tempo real e previsão de demanda, que é usada para criar partições de conexões para diferentes classes de prioridade.

Iqbal et al. (2011) acreditam que provedores de nuvem podem construir modelos que representem o perfil de utilização de diferentes aplicações com base na utilização do *hardware*, mas defendem que técnicas convencionais com treinamento e modelagem *off-line* são impraticáveis nesses casos. Os autores propõem o uso de um método de caixa preta para previsão de capacidade que, inicialmente, identifica padrões de aplicações *Web* dos arquivos de *logging* pelo uso de técnicas de aprendizado de máquina não supervisionado e, a partir desses dados, constrói um modelo capaz de prever a capacidade da aplicação para um padrão de carga específico.

Em seu trabalho, Zhang et al. (2011) afirmam que a demanda sobre cada servidor em nuvem pode variar bastante em tempo de execução e trata o problema de alocação de recursos e o balanceamento de carga como um desafio para os provedores de nuvem. Os autores introduzem uma nova abordagem de predição estatística e um mecanismo de avaliação de recursos disponíveis para fazer decisões sobre a alocação de recursos, pela previsão da demanda de recursos de cada máquina virtual. A abordagem proposta é composta de duas partes, na primeira parte é feita uma análise dos dados históricos em tempo de execução para previsão da demanda de recursos de cada máquina virtual; numa segunda etapa, é feita uma escolha de qual máquina física tem recursos suficientes para executar cada instância virtual.

Chandra et al. (2003) usam modelagem de séries temporais para capturar comportamento transiente da carga de trabalho de aplicações. Com base no comportamento identificado, algoritmos de previsão para estimar valores esperados de carga para os próximos instantes. Com isso, é possível otimizar os recursos de acordo com o que foi previsto.

Além das propostas que usam métodos estatísticos e que envolvem técnicas de inteligência artificial, outra frente vem ganhando destaque, a teoria de controle. Desenvolvida principalmente pelas engenharias e aplicada há tempos pela indústria, a teoria de controle vem sendo usada recentemente pela computação como uma proposta de solução reativa para gerenciamento de recursos quando sistemas são submetidos a situações imprevisíveis (Ying et al., 2003; Chen et al., 2005; Urgaonkar et al., 2005; Padala et al., 2007; Lim et al., 2009; Xiong et al., 2010).

Infelizmente, a teoria de controle ainda é pouco explorada em ciência da computação e áreas relacionadas mesmo não sendo uma teoria recente. Embora teoria de controle tenha sido aplicada com sucesso nas engenharias, principalmente elétrica e mecânica, somente algumas pesquisas recentes em computação têm mostrado bons resultados.

Vários problemas de sistemas de computação podem ser resolvidos usando teoria de controle como, por exemplo, problemas de contenção na camada de enlace de dados para redes sem fio (Boggia et al., 2007), uso otimizado de recursos computacionais em um *cluster* de processadores e problemas de escalonamento de tarefas em servidores (Wang e Chen, 2008).

No contexto de arquiteturas orientadas a serviço de larga escala, a adaptação automática constitui um conceito muito atraente e um desafio é desenvolver mecanismos que consigam simultaneamente garantir o cumprimento de ambos requisitos funcionais, concentrados toda a lógica do negócio, e não funcionais, concentrados os requisitos do negócio.

No início, a teoria de controle foi aplicada a sistemas computacionais para resolver, principalmente, problemas de garantias de parâmetros de *QoS* para aplicações *Web*. Esse primeiro passo foi fundamental para que se criasse um ponto de partida para aplicação das técnicas de controle realimentado em sistemas de computação, que viria a tornar-se uma promissora linha de pesquisa para os dias atuais.

Um dos primeiros trabalhos de pesquisa que utilizou a teoria de controle na solução de problemas de sistemas computacionais foi em escalonamento de tempo real, introduzido por Stankovic

e Son (1999). O estudo apresenta um algoritmo de escalonamento de tempo real baseado em controle realimentado chamado *Feedback Control - Earliest Deadline First (FC-EDF)*, o qual representa um ponto inicial na longa jornada de criar uma teoria e prática de aplicação de controle em problemas de tempo real.

FC-EDF integra um controlador *Proportional-Integral-Derivative (PID)* com um escalonador *Earliest Deadline First (EDF)*. A figura 2.5 mostra a arquitetura proposta para o *FC-EDF*. No modelo todas as tarefas são consideradas independentes (*i. e.* uma nova tarefa não depende das anteriores).

Figura 2.5 — Arquitetura do *FC-EDF*

Fonte: Stankovic e Son (1999).

O objetivo do *FC-EDF* mantém a **taxa de perdas de deadline**¹³ em zero ou muito próximo disso para as tarefas que foram aceitas pelo controle de admissão. Em termos de sistemas de controle, o sistema de escalonamento usa a *DMR* no instante de tempo t como a variável de controle e $DMR = 0$ como entrada de referência.

Para lidar com a sobrecarga de requisições que podem aumentar indefinidamente a taxa de utilização sentida pelo sistema, um controle de admissão e um controlador de nível de serviço foram incluídos no escalonador. O controle de admissão pode controlar o fluxo de carga de trabalho no sistema e controlador de nível de serviço pode ajustar a carga de trabalho dentro do sistema. O controlador *PID* usa o erro da *DMR* para calcular quantos processadores são necessários para

¹³Comumente, o termo a métrica é referenciada pela sigla em inglês *Deadline Miss Ratio (DMR)*.

processar as requisições que aguardam para serem processadas no sistema. Outros trabalhos envolvendo escalonamento realimento foram propostos por Lu et al. (1999); Stankovic et al. (2001); ?); Lu et al. (2002, 2006).

Em um outro trabalho, Zhang et al. (2002) descrevem o projeto e a implementação de uma arquitetura de *middleware* para controlar parâmetros de *QoS* baseado na teoria de controle. A figura 2.6 ilustra os principais componentes da arquitetura do *middleware* chamado de **ControlWare**.

Figura 2.6 — Principais componentes da arquitetura do ControlWare

O *middleware* utiliza uma variedade de *softwares* e bibliotecas para mapear especificações de *QoS* em sistemas computacionais. A interface entre o software de serviço e os *loops* de controle é chamado de **SoftBus** que é um protocolo distribuído que executa entre várias máquinas diferentes e forma um *backbone* virtual entre os servidores, sensores, atuadores e controladores envolvidos.

Ying et al. (2003) abordam o problema de garantia de atraso de *QoS* relativa em servidores *Web Apache*¹⁴ em condições altamente imprevisíveis, e propõem um *framework* que utiliza técnicas combinadas de previsão para os tamanhos das filas de cliente conectados com controle realimentado para alocação de recursos. A figura 2.7 ilustra uma arquitetura genérica em que o previsor baseado em filas funciona como um controlador *feedforward*.

É usado um previsor para cada classe de serviço, esse previsor tenta antecipar o tamanho das filas nos servidores e, com base, nessa informação determinar uma quantidade sugerida de recursos que são necessários para atender a uma determinada classe dentro dos parâmetros de *QoS*. Essa informação é comparada com a entrada de controle dada pelo controlador realimentado que mede os níveis de desempenho e compara com uma entrada de referência.

Para solucionar o problema das garantias de atrasos também para *QoS* absoluta, Lu et al. (2006) projetam e implementam um arquitetura adaptativa, usando teoria de controle, de servidor *Web*

¹⁴<http://httpd.apache.org/>

Figura 2.7 — Arquitetura genérica que combina filas e controle realimentado

Fonte: Ying et al. (2003)

com um protótipo para o servidor *Web Apache*. O intuito do servidor proposto é prover garantias de atraso nas conexões TCP, tanto relativa quanto absoluta, para diferentes classes de serviço. A figura 2.8 ilustra a arquitetura proposta.

Figura 2.8 — Arquitetura para garantia de atrasos usando controle realimentado

Fonte: Lu et al. (2006)

Os autores partem da premissa de que grande parte do tempo de uma comunicação desse tipo é tomado pelo algoritmo de estabelecimento de conexões do protocolo TCP, e assim, procuram

controlar a quantidade de processos que atenderão as requisições de sincronização enviadas pelos usuários.

Em um trabalho parecido ao de Ying et al. (2003), Urgaonkar et al. (2005) também propõem uma técnica de alocação dinâmica de recursos para um aplicação *Web* de múltiplas camadas (*i. e.* banco de dados, interface com usuário e lógica da aplicação). Nessa técnica um modelo de filas é usado para estimar a quantidade de recursos que cada camada da aplicação demanda. Uma vez que se sabe o quanto de recursos cada camada exige, estratégias preditivas e reativas são combinadas para provê-los.

Grande parte dos esforços de pesquisa de soluções usando teoria de controle em computação foram concentrados na garantia de parâmetros de *QoS* relativa e absoluta para aplicações *Web*. Essas pesquisas obtiveram resultados relevantes e abriram um novo panorama para aplicação de teoria de controle em outros tipos de sistemas computacionais, como é o caso da computação em nuvem. Pela própria natureza como os recursos computacionais são adquiridos, a computação em nuvem é um ambiente que favorece a alocação e desalocação dinâmica de capacidades computacionais e, sendo assim, controle realimentado pode ser uma importante ferramenta para lidar com esses problemas.

A alocação dinâmica de recursos é uma realidade que vem ganhando destaque desde as grades computacionais e passou a estar definitivamente em foco com a computação em nuvem. A provisão de capacidades computacionais dinamicamente segundo a demanda está intimamente ligada a existência de transações financeiras entre as partes. Não se pode abordar computação em nuvem sem abordar como acontece a negociação entre as entidades envolvidas e, por isso, diversos trabalhos defendem que a alocação dinâmica de recursos computacionais compartilhados deve obedecer a mecanismos que sejam orientados ao mercado de computação (Lai et al., 2005; Buyya et al., 2009a; Jiang et al., 2009; Auyoung et al., 2004; An e Lesser, 2010; Lewis et al., 2009; Wolski, 2001).

Kusic e Kandasamy (2007) criam um *framework* para alocação dinâmica de recursos usando um *Limited Lookahed Control (LLC)*, juntamente com um modelo de previsão, com o objetivo de maximizar o rendimento. A figura 2.9 ilustra o modelo do sistema para as três classes de serviço propostas pelos autores.

As requisições de cada uma das classes que chegam ao despachador são enviadas a filas diferentes. Cada fila está associada a um *cluster* de servidores com quantidades distintas. Também é mantido um *cluster* com servidores ligados, mas não alocados para nenhuma das classes. Para alocação de servidores a cada *cluster* é usado um controlador como o mostrado pela figura 2.10.

Um previsor é usado para receber o intervalo entre chegadas de uma classe e o tempo médio de processamento e então produzir previsões. Então um controlador é usado para calcular o tamanho de cada *cluster* com base no tempo de resposta dado pelo modelo. Com base no tamanho dos *cluter* tanto modelo quanto o sistema objetivo são ajustados.

Figura 2.9 — Modelo do sistema para as classes de serviço ouro, prata e bronze
 Fonte: Kusic e Kandasamy (2007)

Indo ao encontro dos anseios de quem pretende adquirir novas capacidades computacionais, Klems et al. (2009) propõem um *framework* para comparação de custos envolvidos na implantação de um *data center* e na contratação de serviços em nuvem. São identificados aspectos chaves que permitem ao consumidor comparar os custos de investimento em uma infraestrutura própria ou na terceirização. Esse artigo, no entanto, nada aborda os custos de utilização de recursos e dá uma visão de medição de custos apenas do lado do provedor.

Historicamente, pesquisas em qualidade de serviço estavam concentradas na relação binária entre usuários e servidores, principalmente *Web*. Com o advento das nuvens, da provisão dinâmica de recursos sob demanda e as diretrizes dadas pelo mercado, uma terceira entidade passou a fazer parte desse ambiente, o consumidor de serviços em nuvem. Enquanto muitas pesquisas concentram-se basicamente na abordagem de soluções para a garantia de qualidade de serviço experimentada por usuários de nuvem ou focam em problemas exclusivamente dos provedores (*i. e.* diminuição do consumo de energia, economia de recursos), poucos trabalhos abordam os problemas dos consumidores desses serviços.

A alocação dinâmica de recursos é de grande interesse por parte dos provedores que podem permitir que seus consumidores aumentem a quantidade de recursos computacionais a qualquer tempo, além de que a disponibilização de um bom mecanismo desse tipo pode constituir um importante diferencial de mercado em relação a outros provedores. Contudo, um mecanismo adequado de provisão dinâmica de recursos é principalmente do interesse do consumidor, uma vez que pode ser determinante no rendimento experimentado por essa entidade. Do ponto de vista do consumidor como mostrado na figura 1.1, prover adequadamente os recursos significa gastar o suficiente

Figura 2.10 — Diagrama de blocos do controlador para as classes ouro, prata e bronze
 Fonte: Kusic e Kandasamy (2007)

com os recursos do provedor e garantir que os recursos são suficientes para atender a demanda dos usuários finais.

Fito et al. (2010) propõem uma arquitetura completa de hospedagem de nuvem, como foco nos provedores, com possibilidade de negociação entre *data centers* para que os recursos sejam vistos pelo consumidor como ilimitados. A arquitetura proposta permite que *data centers* tenham seus recursos aumentados temporariamente por negociação com outros provedores. A arquitetura proposta visa diminuir as penalizações sofridas pelo provedor nos casos de quebra de contrato de prestação de serviços.

Em outro trabalho Lim et al. (2009) propõem que consumidores de nuvem devem desenvolver controladores externos aos serviços de nuvem, usando funções disponíveis na *API* do provedor para gerenciar os recursos disponíveis. Os autores também propõem uma política de controle baseadas em limiares proporcionais, em que os limites inferiores e superiores de um intervalo de controle são modificados conforme o comportamento do sistema. Neste trabalho, bons resultados são obtidos quando um controlador do tipo integral é usado para um sistema cuja carga de trabalho não provoca grandes alterações na quantidade de recursos.

Com foco no consumidor, Buyya et al. (2009a) propõem uma arquitetura orientada a mercado para controle de admissão num ambiente de computação em nuvem. Os autores chamam a atenção para o desenvolvimento de programas que atualmente deve considerar a utilização por uma quantidade muito grande usuários, muitas vezes como um serviço. O intuito da arquitetura é garantir que somente a quantidade suportável de arquitetura terá acesso ao serviço executado pelas máquinas virtuais que o hospedam. A arquitetura proposta define uma série de componentes padrões que devem estar presentes numa arquitetura de nuvem voltada a mercado e que também fazem parte da arquitetura proposta neste trabalho.

Parte III

Metodologia

Controle em sistemas computacionais

Ao longo dos anos, a teoria de controle tem sido aplicada de forma bem sucedida como uma importante ferramenta na resolução de problemas em diversas áreas da engenharia. Na indústria, o controle automático é essencial na automação de processos de fabricação e compõe parte significativa da funcionalidade (*e. g.* controles de velocidade e controles de temperatura) de alguns produtos manufaturados (Ogata, 2009).

A teoria de controle desenvolveu um inovador campo de atuação para engenharia e algumas vertentes de ciências naturais, e conta como uma rica coleção de ferramentas de linguagem matemática. Essas ferramentas auxiliam na descrição do comportamento de sistemas dinâmicos em termos de como eles respondem a diferentes estímulos, e também na verificação de propriedades de interesse como observabilidade, controlabilidade e estabilidade. Igualmente, oferece uma base extensa para estratégias de projetos de controle aplicadas a sistemas lineares e não lineares.

Apesar das muitas aplicações de controle estarem presentes em produtos do dia-a-dia (*e. g.* o termostato de um refrigerador), a aplicação em sistemas computacionais ainda é incipiente e pouco definida. Mesmo tarefas simples como a seleção de comportamentos que se deseja observar no projeto de um controlador (por exemplo, a escolha da variável de saída que se deseja controlar em *software* e de que forma proceder o controle para atingir os objetivos) pode não ser uma tarefa trivial em sistemas computacionais.

Enquanto a teoria de controle é aplicada com sucesso em outras áreas da ciência, em computação as pesquisas apenas começaram e ainda há muito para ser pesquisado e aplicado (Hellerstein e Singhal, 2009). Recentemente, pesquisas em ciência da computação têm voltado seus olhares para as possíveis aplicações da teoria de controle em sistemas de computação autônomos, onde a planta a ser controlada é formada por programas de computador. Técnicas de controle realimentado têm

sido usadas para alcançar objetivos de desempenho específicos em sistemas de computação e tem conseguido bons resultados principalmente na aplicação em escalonamento realimentado, clusteração dinâmica de múltiplos processadores em problemas de particionamento de recursos e no aumento de *throughput* em problemas de controle de congestionamento.

Esta seção procura elencar os conceitos e definições necessárias para o entendimento deste trabalho de pesquisa, além de alguns fatores e peculiaridades que devem ser considerados quando se pensa no controle de sistemas computacionais, seja para o emprego em aplicações que executam localmente (e. g. o controle da quantidade de memória utilizada por um programa) ou distribuída (e. g. balanceamento da carga entre servidores de aplicação no *backend* de um *cluster*). Entretanto, os conceitos apresentados aqui não têm o objetivo de compor uma referência completa de teoria de controle ou controle realimentado, uma vez que se trata de uma vasta teoria e deve ser melhor explorada em textos específicos e próprios sobre o assunto.

Antes de mais nada é preciso ter em mente que a aplicação da teoria de controle clássica em computação, nos moldes como é aplicada em outras áreas, não é uma tarefa direta, tampouco simples. Projetar sistemas de controle para sistemas de computação significa lidar com a necessidade constante de adaptações de métodos clássicos para a realidade desses sistemas. Essas adaptações tem por objetivo tornar o projeto de controle viável, menos complicado e evitar que o mesmo decorra em resultados insatisfatórios. Na verdade, o uso de controle em computação traz algumas singularidades que devem ser observadas desde a modelagem do sistema objetivo até a implantação do sistema de controle e que podem ser determinantes no sucesso da aplicação como um todo. Em seu trabalho, Hellerstein (2010) propõe um *framework* para testar implementações de controle em sistemas computacionais e enumera algumas dificuldades enfrentadas por projetistas de controle que tem muito em comum com as considerações apresentadas na próxima seção.

3.1 Considerações sobre a aplicação de controle em sistemas computacionais

Para entender as particulares do projeto de controle para um sistema de computação é interessante entender como funciona o controle para um sistema convencional contínuo. Em um sistema de controle de temperatura de um forno elétrico, como o ilustrado na Figura 3.1, a temperatura do forno é aferida por um termômetro analógico que entra em equilíbrio com o meio e então passa a informação da temperatura medida para um conversor analógico-digital. Após converter o sinal analógico da temperatura para digital, o conversor repassa o valor digitalizado para uma interface que o disponibiliza para o controlador. O controlador, que nesse caso é um programa de computador, mas que poderia ser um circuito elétrico-eletrônico construído para tal, compara o valor da temperatura atual do forno com a referência dada por uma entrada previamente programada. Caso haja diferença entre os valores comparados, o erro resultante é usado pelo controlador para gerar

um sinal ou entrada de controle, cujo objetivo é diminuir o erro. O sinal de controle passa por uma interface até chegar ao amplificador e posteriormente ao relé, onde é usado para controlar a corrente que passa pela resistência dentro do forno, consequentemente, ajustando a temperatura do mesmo.

Figura 3.1 — Sistema de controle de temperatura.

Fonte: Ogata (2009)

Um bom sistema de controle manterá a temperatura do forno sempre igual ou muito próxima ao valor dado pela entrada programada, absorvendo a ação de fatores fora de controle (por exemplo, as variações de temperatura do ambiente onde está o forno) que podem resultar em variações da temperatura interna do forno.

O esquema apresentado na Figura 3.1 é bastante comum e pode ser aplicado em muitos outros sistemas com as devidas adaptações. Em sistemas como o apresentado, vários componentes (por exemplo, interfaces, conversor e amplificador) são dedicados a uma única e específica função. Ao contrário do que acontece no caso do controle de temperatura, em computação os recursos físicos tais como processador, memória e armazenamento são comumente compartilhados por vários usuários e processos. Observe que no sistema ilustrado, o termômetro é responsável apenas pela aferição da temperatura interna do forno quando em equilíbrio com o meio e não divide sua funcionalidade com nenhum outro equipamento que faz parte do sistema. O mesmo não acontece com sistemas computacionais, já que recursos físicos são compartilhados pelo sistema operacional e pelos múltiplos processos e *threads* que concorrem durante suas execuções para ocupá-los por um determinado período de tempo.

Como esses recursos são limitados e devem estar disponíveis para todas as aplicações, o sistema operacional dispõe de mecanismos para multiplexá-los. No caso do processador, a decisão de que processo ou *thread* será executado depende da política de escalonamento adotada que, do ponto de vista de controle, pode não ser a mais adequada.

Em se tratando de controle onde o sistema objetivo é um programa de computador, as funcionalidades que compõem um sistema de malha fechada¹ (por exemplo, o controlador ou um sensor) podem ser implementadas como funções internas, *threads* ou mesmo processos independentes e, dessa forma, competem com o próprio sistema objetivo pelos recursos físicos disponíveis. Isso

¹Os sistemas de malha fechada são detalhados na seção 3.3.

significa dizer que, se o sensor é um processo independente, enquanto o sensor ocupa o processador com a aferição da resposta, provavelmente o sistema controlado não está em execução, pelo menos não no mesmo núcleo do processador.

Esse compartilhamento no nível físico faz com que os recursos multiplexados no tempo, como os núcleos do processador, sejam ocupados por entidades de programas em intervalos de tempo limitados e esparsos. Assim, é preciso considerar que sistemas computacionais são, em essência, discretos, apesar da ideia de continuidade de tempo provocada pela multiprogramação.

A análise de sistemas discretos utilizando técnicas de sistemas contínuos pode resultar em modelos que não representam a realidade que se deseja avaliar e, então, é justo que essas técnicas sejam adaptadas ou que técnicas próprias para análise de sistemas discretos sejam empregadas em sistemas de computação.

Outro aspecto relevante quando se trata o controle de sistemas computacionais é que o monitoramento da saída ou de um estado do sistema é, por si só, um procedimento custoso já que também consome recursos. Monitorar sistemas de computação sem interferir no desempenho do mesmo tem sido um desafio estudado por pesquisadores há bastante tempo.

Há também o fato de que a informação desejada nem sempre está disponível para aferição. A leitura de uma saída de um programa pode requerer a instrumentação do código que produz essa saída, o que não é possível quando não se tem acesso ao código fonte. Em casos onde a instrumentação não é possível, como em *softwares* comerciais não livres, as informações de interesse ficam restritas àquelas disponíveis em arquivos de *logging* de execução ou acessíveis por uma *API* disponibilizada pelo próprio desenvolvedor do programa. Em última instância, sendo inviável a utilização das informações disponíveis, é possível usar estratégias de engenharia reversa e reengenharia para instrumentar um sistema objetivo (Salehie e Tahvildari, 2009).

Existem também as informações impossíveis de serem acessadas por *software*, e outras que quando são acessíveis não fornecem informações com o nível de precisão adequado, por exemplo, o consumo exato de bateria de um determinado dispositivo obtido por uma chamada de sistema. Nesses casos pode ser necessário desenvolvimento de equipamento específico para realizar as leituras reais diretamente no *hardware* sem a interferência do sistema operacional. Sempre que o sensoramento por *software* for possível e oferecer resultados precisos, programas com funções de monitoramento como Ganglia² e Nagios³ podem ser bastante úteis.

Mesmo que as informações das variáveis de interesse estejam disponíveis e sejam exatas, ainda há que se analisar se estarão disponíveis em um tempo hábil para o controlador. Atrasos são relativamente comuns em sistemas computacionais e podem fazer com que uma determinada informação só esteja disponível quando não é mais válida ou já tenha perdido sua utilidade. Alguns exemplos de atraso estão relacionados à espera pela conclusão de uma operação de entrada e saída por um processo, ao travamento de um arquivo por parte de outro programa e problemas de atraso

²<http://ganglia.sourceforge.net/>

³<http://www.nagios.org/>

na rede de comunicação. Além do mais, controlar um sistema computacional implica na execução de mais processos ou *threads* em concorrência, que pode alterar a utilização de memória e processador, o que é pouco impactante em programas convencionais, mas pode ser determinante para sistemas críticos.

A menos que utilização do sistema apresente comportamento regular e previsível, é preciso considerar que a interação com o usuário adiciona mais aleatoriedade a um sistema que já é estocástico. Isso pode tornar o sistema mais difícil ou até impossível de ser modelado. Em casos assim, técnicas de controle robusto, de aplicação mais difícil, podem ser utilizadas para projetar controladores que cumpram bem seus objetivos.

3.2 Propriedades de interesse

Quando há o interesse em controlar a dinâmica de um determinado sistema é interessante, algumas vezes necessário, que o sistema apresente comportamentos previsíveis para certas propriedades. A não ocorrência desses comportamentos pode dificultar a modelagem do sistema e do controlador e pode, inclusive, incorrer na impossibilidade de controlar o sistema objetivo. Algumas propriedades são particularmente importantes e devem ser observadas quando o sistema alvo é um sistema computacional (Hellerstein et al., 2004). Essas propriedades são estabilidade, exatidão, tempo de ajuste e amplitude máxima⁴.

- **Estabilidade:** A estabilidade é a propriedade que define a capacidade do sistema em manter sua saída estável sobre um ponto de operação, ou dentro de um intervalo, para entradas previamente conhecidas. A estabilidade é uma propriedade fundamental do sistema que se pretende controlar, uma vez que não é possível controlar sistemas instáveis. Existem diferentes maneiras de analisar a estabilidade, uma das formas mais aceitas relaciona a limitação da entrada com a limitação da resposta do sistema. Nesse método um sistema é dito estável somente se nenhuma entrada limitada por um intervalo gera uma resposta sem limites definidos⁵. Isso garante que o sistema tenha comportamento previsível e saídas esperadas quando valores dentro de um intervalo limite de operação forem usados como entradas de controle.
- **Exatidão:** Um sistema de controle tem exatidão quando ao entrar em regime estacionário o desvio entre a entrada de referência e a saída medida é nulo ou muito próximo disso. Um sistema de controle exato é necessário para que os objetivos de controle, por exemplo, regular o valor da taxa de utilização do sistema em 50%, sejam atingidos plenamente. Perceba que a estabilidade não é condição suficiente para exatidão, dado que o sistema pode convergir para um novo valor em regime estacionário diferente da referência e lá permanecer estacionado,

⁴Em inglês, essas propriedades formam uma sigla *Stability, Accuracy, Settling time, maximum Overshoot (SASO)*.

⁵Esse método é conhecido pela sigla em inglês *Bounded Input, Bounded Output (BIBO)*.

fazendo com que o erro em regime estacionário permaneça estável, mas não nulo. Quando o erro em regime estacionário não é nulo o sistema não é exato, no entanto, ponderações podem ser consideradas para a exatidão do sistema quando o erro for diferente de zero, por exemplo, quando se usa uma política de controle proporcional, cujo erro em regime estacionário nunca é nulo.

- **Tempo de ajuste:** O tempo de ajuste ou tempo de assentamento é o tempo necessário para um sistema voltar ao estado de regime estacionário após um período em regime transiente. Em sistemas discretos, o tempo de ajuste é dado pela quantidade de intervalos de amostra necessários para alcançar o estado de regime estacionário. É interessante que um sistema de controle apresente um tempo de ajuste reduzido, convergindo rapidamente para o valor de regime estacionário, já que quanto maior o tempo de ajuste, mais tempo o sistema permanece sem atender os objetivos de controle. O tempo de ajuste é uma propriedade que mede o quão rápido um controlador reage dada uma condição de perturbação que leva o sistema além do regime estacionário.
- **Amplitude máxima:** A amplitude máxima ou simplesmente *overshoot* mede o quanto o controlador reagiu a mais que o necessário para corrigir uma situação de perturbação, seja essa reação positiva ou negativa. É uma porcentagem que quantifica quanto a resposta do sistema foi além do valor de regime estacionário no período em regime transiente. Cabe ressaltar que, em linhas gerais, o *overshoot* tende a ter efeito contrário ao tempo de ajuste. Para um sistema estável, quanto maior o *overshoot* menor será o tempo de ajuste e vice-versa. A presença de *overshoot* deixa claro que o controlador reagiu com intensidade superior a necessária no estado transiente e mede quão bem o sistema reage em situações de perturbação. Um bom controlador deve ter uma amplitude máxima próxima de zero quando possível.

A Figura 3.2 ilustra a resposta de um sistema estável para alguns instantes depois que uma perturbação em degrau foi introduzida no sistema. A fim de facilitar o entendimento, a origem do tempo foi deslocada para o momento em que a perturbação foi inserida no sistema.

Pelo gráfico da Figura 3.2 é possível supor que o sistema é estável, uma vez que a entrada aplicada foi um degrau (*i. e.* entrada dentro de um intervalo limitado) e a resposta do sistema convergiu. A condição apresentada não é suficiente para concluir que o sistema é estável para qualquer entrada, mas é possível concluir que o sistema apresenta comportamento estável para a entrada degrau. Após a perturbação, é possível notar um comportamento transiente no intervalo $0 \leq k < 15$. Quando em $k = 15$ o sistema entra novamente em regime estacionário. Logo, o tempo de ajuste T_s ilustrado pode ser calculado por uma subtração simples dos intervalos observados onde a perturbação acabou, K_f , e começou, K_i , como mostra a equação 3.1.

Figura 3.2 — Saída de um sistema de controle estável.

$$T_s = K_f - K_i$$

$$T_s = 15 - 0$$

$$T_s = 15 \quad (3.1)$$

Como suposição, após o período em regime transiente a saída em regime estacionário deveria ser $y'_r = 45$, no entanto, percebe-se que o sistema converge para outro valor, $y_r = 40$. Isso significa que o sistema mesmo sendo estável não é exato, já que seu erro em regime estacionário é diferente de zero. O erro em regime estacionário e_r obtido para o sistema analisado foi calculado e é mostrado pela equação 3.2.

$$e_r = y'_r - y_r$$

$$e_r = 45 - 40$$

$$e_r = 5 \quad (3.2)$$

A resposta do sistema atinge a maior amplitude $y(k) = 60.8$ quando $k = 1$. O cálculo do *overshoot* máximo M é mostrado pela equação 3.3.

$$M = \frac{y(1) - y_r}{y_r}$$

$$M = \frac{60.8 - 40}{40}$$

$$M = 0.52 \quad (3.3)$$

O cálculo da amplitude máxima da saída do sistema mostra que para essa entrada degrau a resposta do sistema chegou a ser 52% maior que o valor necessário. O conhecimento da amplitude máxima permite projetar um sistema que possa suportar essas variações repentinas em períodos transientes. Essa análise é importante principalmente para o planejamento de capacidade.

Supondo que o sistema em questão fosse um sistema elétrico e que a saída observada é a corrente, em amperes (A), que passa por um circuito que suporta nominalmente $50A$. Na situação ideal o valor máximo em regime estacionário seria de $45A$, ou como a situação real ilustrada, seria de $40A$, ambos abaixo do valor suportado pelo circuito. No entanto, durante o período em regime transiente a amplitude máxima foi de $60.8A$, que ultrapassa o limite suportado pelo circuito. Esse tipo de situação que vai além das expectativas suportadas pode fazer com o sistema entre em colapso ou tenha sua vida útil diminuída.

Um sistema de controle ideal deve ser estável, apresentar erro nulo, ter o tempo de ajuste nulo e não apresentar *overshoot*. É claro que sistemas de controle ideais são impossíveis na prática, já que as próprias características físicas e a dinâmica do sistema objetivo impossibilitam manter suas propriedades de interesse em níveis tão rígidos. Assim, o projeto de um sistema de controle real deve garantir que a dinâmica do sistema seja mantida dentro de patamares considerados aceitáveis.

3.3 Sistemas de controle de malha aberta e malha fechada

Sistemas de controle de malha aberta⁶ são aqueles em que a resposta atual do sistema não é considerada como um parâmetro para as entradas futuras. Não há a comparação da saída real observada com a entrada referência, a resposta do sistema nem mesmo é aferida para ser comparada com a entrada. Esses sistemas são projetados sobre a premissa de que os valores de entrada são suficientes para gerar a saída desejada desconsiderando a ação de perturbações, mesmo sabendo que essas perturbações podem fazer com que a saída desejada não seja atingida. Por exemplo, um sistema de ar condicionado convencional sem leitura da temperatura, uma vez ajustado o termostato as variações de temperatura ambiente não são consideradas na temperatura final.

⁶o termo em inglês *open-loop* é comumente utilizado para referir-se à sistemas de controle de malha aberta.

Em qualquer sistema de controle de malha aberta a saída não é comparada com a entrada referência, e isso implica que a presença de perturbações não levam a saída para o valor desejado. Muitos sistemas de computação são projetados e implementados como sistemas de controle de malha aberta em que se acredita que os valores atribuídos a determinadas variáveis de configuração levarão o sistema ao desempenho desejado. Um exemplo muito comum são os servidores *Web*, quando se determina o tamanho máximo de um *buffer* de clientes conectados sem considerar se os mesmos estão ativos, enviando requisições, ou inativos, esperando que suas requisições sejam respondidas. Para sistemas de computação a abordagem de controle de malha aberta funciona bem enquanto o sistema está em estado de regime estacionário, desconsiderando variações ao longo do tempo. Isso significa que a relação conhecida entre entrada e saída que foi previamente estabelecida no projeto não é verdadeira o tempo todo. A figura 3.3 mostra dois diagramas de blocos que representam a forma geral de um sistema de controle de malha aberta.

Figura 3.3 — Sistemas de controle de malha aberta

O diagrama de blocos da figura 3.3(a) representa uma função genérica $g(k)$, considerando um sistema discreto, onde k é o tempo de amostra na unidade de tempo utilizada. O sistema que se deseja controlar está representado pelo bloco G e converte um sinal de entrada $u(k)$ em um sinal de saída $y(k)$. Já a figura 3.3(b) mostra um sistema de controle de malha aberta em que a entrada de controle $u(k)$ está sujeita a uma perturbação $d(k)$ que em um instante qualquer pode ser adicionada ao seu valor, assim como a saída $y(k)$ pode estar sujeita a ruídos, representados por $n(k)$, que fazem como que a aferição da resposta resulte em um valor diferente do objetivo.

Para configurar a saída do sistema $y(k)$ para o valor desejado é necessário que a entrada $u(k)$ seja ajustada apropriadamente com base na relação de transformação existente de entrada para saída. Em sistema de malha aberta esse procedimento deve ser realizado manualmente e, uma vez ajustada a entrada de referência, o valor permanece estático até que um novo valor seja ajustado. Se algum parâmetro do sistema é alterado ao longo do tempo (*i. e.* uma mudança de G para G' causada por um desgaste interno, a aparição de uma perturbação ou de ruídos) o valor da entrada deve ser reajustado novamente para adequação a nova realidade do ambiente ou para compensar a alteração nas condições internas do sistema. Esse procedimento não é automático como nos sistemas de malha fechada.

Um sistema que busca manter uma relação estabelecida entre a saída e a entrada de referência por comparação e usa a diferença da comparação como entrada para um controlador é chamado

de controle de malha fechada⁷ ou controle realimentado. Um exemplo fácil de ser entendido é o piloto automático de um veículo, no qual a medida da velocidade atual do veículo é medida por um sensor e então comparada com a velocidade referência programada pelo motorista, havendo diferença devido a alguma perturbação, por exemplo, alteração do relevo da via por onde o veículo transita, o controlador envia sinais a um atuador que acelera ou freia o veículo para garantir que o mesmo permanece sempre próximo da velocidade configurada. A figura 3.4 mostra um diagrama de blocos que representa um sistema de malha fechada.

Figura 3.4 — Sistema de controle de malha fechada

Pela figura 3.4 é possível identificar os elementos que fazem parte de um sistema de controle de malha fechada completo.

- **Resposta do sistema ($y(k)$):** A resposta ou saída do sistema é o comportamento dinâmico que se deseja regular, otimizar ou absorver distúrbios.
- **Saída medida ($t(k)$):** A saída medida é a aferição da saída do sistema quando na presença de um ruído aleatório.
- **Entrada de referência ($r(k)$):** A entrada de referência é o objetivo de controle, é o valor no qual se almeja que a saída seja mantida.
- **Ruído aleatório ($n(k)$):** Ruído aleatório ou simplesmente ruído é um distúrbio que provoca uma aferição incorreta da saída do sistema.
- **Transdutor (H):** O transdutor é um bloco opcional em um sistema de controle de malha fechada, cuja a função é transformar o sinal da saída medida em um sinal que possa ser comparado com a entrada de referência.
- **Saída do transdutor ($w(k)$):** A saída do transdutor é o sinal de saída medido transformado.
- **Erro ($e(k)$):** O erro é a diferença instantânea entre a entrada de referência e a saída do transdutor.

⁷Assim como para controle de malha aberta, o termo em inglês *feedback control* para referir-se à sistemas de controle realimentados.

- **Planta (G):** O sistema dinâmico objetivo ou alvo que possui o comportamento que se deseja controlar.
- **Perturbação ($d(k)$):** A perturbação é um distúrbio que pode alterar a forma como a entrada de controle age sobre o sistema objetivo.
- **Entrada de controle ($u(k)$):** A entrada de controle ou sinal de controle é o parâmetro que afeta o comportamento do sistema e pode ser ajustado dinamicamente.
- **Controlador (C):** O controlador detém as políticas de controle e é responsável por transformar o erro em um sinal de controle.

Controle realimentado refere-se a uma operação que, na presença de perturbações, sejam causadas por fatores externos (*e. g.* no caso de um sistema computacional um exemplo comum é o aumento repentino no número de usuários devido ao comportamento sazonal dos usuários da aplicação no início do dia de trabalho) ou fatores internos (*e. g.* tarefas agendadas de manutenção do sistema ou falha de um equipamento de *hardware*), tende a reduzir a diferença que passa a existir entre a saída do sistema e a entrada de referência escolhida.

Em sistemas de malha fechada a saída medida é comparada com a entrada de referência e a diferença é usada como um sinal que alimenta o controlador que então atua para diminuir essa diferença levando a saída para o valor esperado. A referência pode ser comparada com a saída medida ou com uma função da saída que, tipicamente, adiciona atrasos ou suavizações por meio de médias dos valores observados anteriormente. A principal característica de sistemas de controle de malha fechada está na necessidade de um laço de realimentação cujo intuito é diminuir o erro do sistema.

Na abordagem de malha fechada mostrada na figura 3.4, o bloco H funciona como um sensor que mede a saída do sistema $y(k)$ e produz um sinal $w(k)$. Esse sinal pode ser o próprio sinal da saída ou então um sinal transformado por um filtro ou defasado em algumas unidades de amostra. Para fins de entendimento, suponha que os blocos C , G e H que fazem parte do sistema de malha fechada são constantes proporcionais. Quando se aplica um sinal de referência $r(k)$ na entrada do sistema, espera-se que a saída medida tenha o valor esperado correspondente. O desvio entre a referência e a saída num determinado instante k é dado pelo erro $e(k)$ como mostra a equação 3.4.

$$e(k) = r(k) - w(k) \quad (3.4)$$

Esse sinal de erro é injetado no bloco controlador C que baseado nas políticas de controle estabelecidas produz um sinal de controle $u(k)$, como mostra a equação 4.4.

$$u(k) = Ce(k) \quad (3.5)$$

O sinal de controle $u(k)$ é passado ao sistema objetivo G , forçando o sistema a levar a saída para o valor desejado. Se atualmente o sistema está com a saída no valor correto, então $u(k)$ deve

estar no valor correto. Cabe ressaltar que o sinal $u(k)$ pode estar sujeito a interferências causadas por perturbações, $d(k)$. A saída produzida pelo sistema alvo G é dado pela equação 3.6.

$$y(k) = G[u(k) + d(k)] \quad (3.6)$$

Assim como o sinal $u(k)$ está sujeito a perturbações, a saída do sistema pode estar sujeita a ruídos $n(k)$ durante o monitoramento. O sinal $t(k)$, como mostra a equação 3.7, é resultado da ação do ruído na aferição da saída do sistema.

$$t(k) = y(k) + n(k) \quad (3.7)$$

O sinal $t(k)$ é usado como entrada pelo bloco H , cuja saída $w(k)$, dada pela equação 4.6, é comparada com o valor de referência.

$$w(k) = Ht(k) \quad (3.8)$$

Qualquer alteração indesejada em $y(k)$ devido a um fator interno de G (*i. e.* uma alteração na constante proporcional de G causada por um desgaste) ou externo (*i. e.* perturbações e ruídos) fará com que o sinal $w(k)$ sofra uma alteração correspondente. Uma alteração em $w(k)$ faz com que a magnitude do erro aumente, e como consequência o controlador reaja, gerando um efeito oposto ao erro. O aumento do erro faz com que o sistema reaja forçando novamente a saída para o valor desejado, havendo erro, o sistema sempre será levado para a saída que corresponde ao sinal de referência. Isso faz com que os sistemas de malha fechada absorvam bem situações de regime transientes, ao contrário daqueles de malha aberta.

3.4 Modelagem de sistemas computacionais

Um projeto de sistema de controle clássico se divide em duas etapas sequenciais: **identificação do sistema** e **projeto do controlador** (Parekh et al., 2002). Para um bons resultados na fase de projeto do controlador é necessário que o sistema tenha sido identificado corretamente na etapa anterior. Identificar um sistema consiste em modelar matematicamente o seu comportamento de interesse de forma que seja possível entender e analisar suas características dinâmicas. O modelo matemático de um sistema é composto pelas equações que representam a dinâmica do mesmo com exatidão ou precisão razoável (Ogata, 2009). Um sistema pode ser representado por muitos modelos matemáticos diferentes, uns mais precisos, outros mais simplificados, dependendo de quão importante é a descrição de determinadas características para a análise.

Em geral, é possível representar sistemas físicos (*e. g.* eletrônicos, químicos e biológicos) por equações diferenciais obtidas a partir leis básicas da Física. Esses sistemas de equações di-

ferenciais descrevem uma determinada dinâmica do sistema no domínio do tempo e, com o desenvolvimento dessas equações, é possível identificar propriedades fundamentais para o projeto de controle.

Nas engenharias sistemas mecânicos são bastante semânticos e amplamente utilizados como exemplo na modelagem de sistemas. A figura 3.5 ilustra um sistema mecânico massa-mola simples onde um objeto de massa m está preso a uma mola cuja constante de elasticidade é igual a k fixa em uma superfície plana. Para simplificar a modelagem, considerações podem ser feitas, como supor que o sistema está livre da influência de outras forças.

Figura 3.5 — Sistema massa-mola.

Segundo a lei de *Hooke* para elasticidade de corpos, a deformação causada num objeto elástico é proporcional a força aplicada, sendo sua forma algébrica dada pela equação 3.9.

$$\vec{F} = k\Delta x \quad (3.9)$$

A esquerda da figura 3.5 é possível observar o sistema em repouso. Isso significa que a força resultante (*i. e.* soma de todas as forças atuantes) do sistema é nula, $\vec{F}_r = 0$, em outras palavras, o sistema está em repouso quando a força peso $\vec{P}$ do objeto tem intensidade igual à força restauradora da mola $\vec{F}_m$. Sabendo que $\vec{P} = m\vec{g}$, onde $\vec{g}$ é a aceleração da gravidade, $\vec{F}_m = kx_0$ da lei de elasticidade, e assumindo que as forças com o mesmo sentido da gravidade são positivas, a equação 3.10 pode ser desenvolvida a partir da equação da força resultante.

$$\begin{aligned}\vec{F}_r &= 0 \\ \vec{P} - \vec{F}_m &= 0 \\ m\vec{g} - kx_0 &= 0\end{aligned}$$

$$m\vec{g} = kx_0 \quad (3.10)$$

Ao lado direito da figura 3.5 é possível observar a mola depois de um deslocamento Δx provocado quando a mola é puxada para baixo. Isso provocou um esticamento da mola que a tirou do estado de repouso e quebrou o equilíbrio existente entre as forças. Logo, quando a força que puxa a mola para baixo deixar de ser aplicada, espera-se que o objeto suba, uma vez que a força restauradora da mola é maior que força peso do objeto, resultando numa força com sentido para cima, contrária ao esticamento da mola. Quando o objeto subir a mola será contraída causando uma força restauradora com sentido para baixo que, somada a força a força peso, fará o objeto descer de novo, continuando nesse ciclo indefinidamente. Sabe-se que a força peso $\vec{P}$ do objeto não foi alterada e que o sistema está livre de outras forças. Além disso, a nova força restauradora da mola agora é $\vec{F}_{m2} = k(x_0 + \Delta x)$. Para calcular a nova força resultante do sistema é possível usar a terceira lei de *Newton*, $\vec{F} = m\vec{a}$, onde $\vec{a}$ é a aceleração linear do objeto. Logo, a nova força resultante do sistema é dada pela equação 3.11.

$$\begin{aligned}\vec{F}_r &= m\vec{a} \\ \vec{P} - \vec{F}_{m2} &= m\vec{a}\end{aligned}$$

$$m\vec{g} - k(\Delta x + x_0) = m\vec{a} \quad (3.11)$$

Substituindo o resultado encontrado em 3.10 na equação 3.11, obtém-se a partir da força resultante do sistema a equação 3.12.

$$\begin{aligned}kx_0 - k(\Delta x + x_0) &= m\vec{a} \\ kx_0 - k\Delta x - kx_0 &= m\vec{a} \\ -k\Delta x &= m\vec{a}\end{aligned} \quad (3.12)$$

Da Cinemática têm que a velocidade média ($\vec{v}$) e a aceleração média ($\vec{a}$) são equivalentes a $\dot{x}$ e $\ddot{x}$, respectivamente. A equação 3.13 mostra a relação entre a variação do espaço e aceleração.

$$\begin{aligned}\vec{v} &= \frac{dx}{dt} = \dot{x} \\ \vec{a} &= \frac{d\vec{v}}{dt} = \frac{d^2x}{dt^2} = \ddot{x}\end{aligned}\tag{3.13}$$

Portanto, substituindo a equação 3.13 na equação 3.12, o modelo matemático que representa a dinâmica do sistema massa-mola da figura 3.5 é dado pela equação diferencial ordinária de segundo ordem 3.14. É importante ter em mente que esse modelo não é único e representa o sistema de acordo com as limitações descritas antes da modelagem. Nesse caso a modelagem desconsiderou a ação de outras forças (*e. g.* atrito com o ar e transformação de energia cinética em calor).

$$\begin{aligned}-kx &= m\ddot{x} \\ m\ddot{x} + kx &= 0\end{aligned}\tag{3.14}$$

É preciso ressaltar que o sistema modelado é simples e serve apenas ao propósito de entendimento. Sistemas reais são mais complexos e mais difíceis de serem modelados a partir das equações diferenciais que descrevem sua dinâmica. Todavia é possível aproximar sistemas mais complexos de sistemas mais simples fazendo as ponderações corretas.

Descrever um sistema por suas equações diferenciais, ou equações de diferença no caso de sistemas discretos, requer entendimento pleno da dinâmica do mesmo, o que nem sempre é possível. Quanto mais complexo o sistema, mais abstrata é a sua dinâmica e mais difícil sua modelagem. Geralmente, sistemas computacionais são muito abstratos, e seu comportamento é difícil de ser modelado e expresso por métodos convencionais, como no exemplo do sistema massa-mola. Difícilmente as leis e teoremas fundamentais da Física serão aplicados em sistemas de computação que têm dinâmica muito peculiar como foi discutido na seção 3.1. A dificuldade de modelar sistemas computacionais como técnicas tradicionais tem motivado o desenvolvimento de abordagens que necessitem de menos de conhecimento especialista do sistema, mas que consigam descrever a dinâmica do sistema de forma satisfatória.

Uma forma de modelar sistemas abstratos é tratá-los como uma caixa preta e então tentar extrair uma função matemática que descreva suficientemente bem o seu comportamento dentro de um intervalo.

Parekh et al. (2002) propõem um método para descrever a dinâmica de um sistema linear invariante no tempo com base em dados históricos de execuções anteriores. Essa estratégia utiliza modelos bem conhecidos de predição de séries temporais auto-regressivos *Autoregressive Moving Average (ARMA)* e *Autoregressive X (ARX)* para descrever a dinâmica de um sistema por meio de suas equações de diferença. Nobile et al. (2007) mostraram que a modelagem de sistemas dinâmicos discretos por séries temporais é eficaz e propôs uma arquitetura em que tráfego de tempo real é modelado como série temporal para previsão de demanda e antecipação de provisão usando modelos *ARMA*.

Um modelo *ARMA* é a combinação linear de dois modelos, um *Autoregressive (AR)* com um modelo *Moving Average (MA)*. No modelo *AR*, o valor atual de um sistema é descrito pelos valores anteriores regredidos e um ruído aleatório, como mostra a equação 3.15, onde $\epsilon(t)$ é um ruído aleatório, $\tilde{y}(t)$ é a resposta do modelo no instante de tempo t , $\{\tilde{y}(t-1), \tilde{y}(t-2), \dots, \tilde{y}(t-n)\}$ são os valores da série temporal observados até o instante $t-1$ e $\{a_1, a_2, \dots, a_n\}$ são os coeficientes de regressão que dão a influência de cada observação anterior na atual.

$$\tilde{y}(t) = a_1\tilde{y}(t-1) + a_2\tilde{y}(t-2) + \dots + a_n\tilde{y}(t-n) + \epsilon(t) = \sum_{i=1}^n a_i\tilde{y}(t-i) + \epsilon(t) \quad (3.15)$$

Para facilitar a análise de uma série de valores históricos, uma boa estratégia é trabalhar com valores normalizados, ou seja, trabalhar com a diferença entre os valores da série original e um valor normal. O valor que representa o ponto normal da série é uma escolha subjetiva e duas possibilidades bastante comuns são o ponto de operação para o qual a série converge e permanece estacionária ou uma média dos valores observados. Para ambos os casos, os valores normalizados são dados por $y(t) = \tilde{y}(t) - \mu$, onde μ é o ponto normal escolhido.

Dois princípios básicos devem ser considerados quando na modelagem de sistemas dinâmicos: **causalidade** e **superposição**. O princípio da causalidade afirma que a saída atual de um sistema depende somente das entradas anteriores. Já o princípio da superposição afirma que a saída de um sistema submetido a uma entrada que é a combinação de várias entradas pode ser obtida pela soma das saídas de cada entrada individual.

O princípio da superposição é especialmente útil porque define o conceito de linearidade de sistemas dinâmicos. Um sistema dinâmico é linear quando o princípio da superposição é válido para aquele sistema. Esse princípio facilita a análise de sistemas dinâmicos, uma vez que é possível quebrar a análise do sistema em pequenas partes mais fáceis de serem resolvidas e juntá-las para obter a solução final. Abdelzaher et al. (2003) demonstram que sistemas computacionais podem ser aproximados por modelos lineares.

Sistemas lineares podem ser classificados como variantes ou invariantes no tempo. Definindo o conceito para sistemas discreto, um **sistema linear discreto variante no tempo** é um sistema dinâmico cujos coeficientes da equação de diferença que descreve seu comportamento são funções do tempo. Sistemas dinâmicos lineares discretos cujos coeficientes da equação de diferença que descreve seu comportamento são constantes são ditos **sistemas lineares discretos invariantes no tempo** e são os sistemas de interesse desta seção.

Para sistemas lineares discretos invariantes no tempo que obedecem ao princípio da causalidade é possível encontrar uma equação de diferença nos moldes de um modelo *ARMA*. Seja um sistema de malha aberta *Single Input, Single Output (SISO)* como o ilustrado na figura 3.3(a), ignorando seus detalhes internos e tratando-o como uma caixa preta cuja entrada é dada por $x(t)$ e a saída

é dada por $y(t)$, a forma geral de um modelo *ARMA* para um sistema nesses moldes é dada pela equação 3.16.

$$y(t) = \sum_{i=1}^n a_i y(t-i) + \sum_{j=0}^m b_j x(t-j) \quad (3.16)$$

Na equação 3.16, a_i são os coeficientes que dão a influência das respostas anteriores do sistema na resposta atual, enquanto b_j são os coeficientes que dão a influência das entradas anteriores na saída atual. Se ao invés de a_i e b_j , os coeficientes fossem dados por funções $a_i(t)$ e $b_j(t)$ o sistema seria variante no tempo.

Hellerstein et al. (2004) propõem 4 passos para o desenvolvimento de uma modelo autoregressivo para um sistema dinâmico como mostra o diagrama de fluxo mostrado na figure 3.6. Esses passos podem ser aplicados na modelagem de um sistema de computação abstrato considerando-o como uma caixa preta e levando em conta o fato de serem discretos.

Figura 3.6 — Etapas para modelagem de caixa preta de um sistema dinâmico.

Fonte: Hellerstein et al. (2004)

O **escopo do modelo** deve definir seus limites e abrangência, em outras palavras, deve definir o modelo em termos de quais variáveis serão usadas como entrada e saída e de que forma elas serão consideradas. Na definição de escopo deve-se deixar claro qual será a ordem do modelo que se pretende obter na análise. É claro que quanto mais auto-regressivo, maior a ordem e mais

descritivo é o modelo. No entanto, uma ordem muito alta pode causar um superajustamento do modelo, fazendo com que o mesmo só trabalhe bem para os dados usados na modelagem. Alguns trabalhos mostram que modelos de ordem baixa fornecem descrição suficiente para resolução de muitos problemas e apresentam bons resultados (Quiroz et al., 2009; Kusic e Kandasamy, 2007; Padala et al., 2007; Lim et al., 2009; Hellerstein e Parekh, 2002). Logo, uma boa estratégia para definir a ordem do modelo é começar com um modelo de primeira ordem e avaliar o ajuste, caso o ajuste seja inadequado, aumentar a ordem numa segunda iteração.

Além da ordem, o escopo também deve considerar que a carga imposta ao sistema para a coleta de dados é significativa e descreve sua atuação normal. Ao contrário do que é feito na análise de desempenho baseada em teoria de filas, em que o foco da análise de desempenho está na carga de trabalho, na modelagem do sistema o foco da análise está na entrada de controle e a carga de trabalho deve ser vista como uma perturbação imposta. Assim é importante que os dados coletados para modelagem do sistema sejam obtidos quando o sistema está sujeito a uma carga de trabalho que exprima o funcionamento do sistema em condições normais.

A análise da carga de trabalho é muito importante nessa etapa, visto que se a carga for muito estocástica o modelo encontrado pode ser variante no tempo, o que torna o projeto de controle muito mais difícil. Para sistemas com alta variação de carga e, conseqüentemente, variantes no tempo, uma alternativa viável é dividir e classificar a carga de trabalho em intervalos, de forma que nesses intervalos a carga de trabalho tenha comportamento aproximadamente constante, e modelar diferentes sistemas para cada intervalo. Para cada sistema modelado, um projeto de controlador será necessário e o sistema de controle deve ser capaz de alternar entre esses controladores de acordo com a carga experimentada pelo sistema.

Sistemas de computação tendem a ser muito estocásticos, fazendo com que a resposta do sistema seja muito variável. A modelagem de controlador com base num modelo que apresenta muita variação de saída, além de ser mais difícil identificar o comportamento do sistema, pode resultar num controlador que atue ininterruptamente. Lembrando que o controlador também compete pelos recursos disponíveis, isso pode ter um impacto mais negativo que positivo para o objetivo de controle. Em casos assim é possível considerar a transformação da resposta do sistema para obter dados mais representativos. A transformação pode ser feita por meio de análise estatística dos dados usando filtros ou considerando medidas de representatividade (*e. g.* médias, percentis e desvios).

Uma vez definido o escopo do modelo é possível **projetar os experimentos**, devem ser escolhidas as entradas de controle que serão utilizadas para estimar o modelo. O sistema será submetido a esses dados e as respostas serão coletadas e confrontadas com as entradas para que se encontre uma relação clara de como a dinâmica do sistema atua na transformação das entradas em saídas. Tendo encontrado essa relação, é possível estimar os parâmetros do modelo *ARMA* que representa o sistema.

Para encontrar a relação entre as entradas de controle e a resposta do sistema é preciso que as entradas de controle escolhidas sejam suficientes para tal, de forma que o intervalo escolhido

seja representativo para operação do sistema de controle. Isso significa que o intervalo de entradas de controle escolhido deve representar possíveis entradas de controle que serão produzidas pelo controlador após o projeto. Além disso, os sinais de entrada devem ser escolhidos de forma que consigam captar todo o comportamento das saídas, ou seja, o intervalo de entradas escolhido deve excitar o sistema suficientemente de forma que as respostas sejam as mais variadas possíveis cobrindo grande parte da região de operação do sistema. É preciso também que o intervalo de entradas seja igualmente espaçado e experimentado. O mesmo espaçamento entre as entradas garante que o sistema seja excitado da mesma forma por várias entradas de controle evitando que sejam geradas respostas tendenciosas.

Visto que o sistema foi submetido às entradas de controle e as respostas foram coletadas, há que se **estimar os parâmetros** para a estrutura do modelo escolhida. Existem vários métodos estatísticos para estimação de parâmetros, sendo que um dos mais comuns é o método dos **mínimos quadrados**, cuja ideia básica é encontrar o valor com melhor ajuste para um conjunto de dados minimizando a soma dos quadrados dos erros residuais. Para isso, é interessante que os valores das entradas e saídas do sistema objetivo sejam normalizadas sobre um ponto de operação ou média.

Após os parâmetros serem estimados e o modelo que representa o sistema ser encontrado, é necessário avaliar o modelo para saber se o mesmo é ou não adequado. Existem diversos métodos para saber se o modelo está adequado ou não, um método direto é a simulação que permite verificar se as respostas geradas pelo modelo encontrado são equivalentes. A resposta gerada pela simulação $\hat{y}(k)$ pode ser confrontada com a resposta gerada pelo experimento $y(k)$ por meio de um gráfico que confronta $\hat{y}(k)$ x $y(k)$. Quando mais próximos estiverem os pontos da diagonal formada pela função $f(x) = x$, mais ajustado é o modelo, quanto mais distante, menos adequado o modelo é.

Existem métodos mais algébricos para julgar o ajustamento de um modelo. Um método bem utilizado é o **raiz quadrada da média do erro**⁸ para as respostas do sistema. Em síntese, essa métrica estima o desvio padrão dos erros entre resposta experimentada e estimada. A equação 3.17 mostra como calcular essa métrica tendo N valores normalizados das saídas experimentada $y(k)$ e estimada $\hat{y}(k)$.

$$E = \sqrt{\frac{1}{n} \sum_{i=1}^n [y(i+1) - \hat{y}(i+1)]^2} \quad (3.17)$$

Quanto menor for o $RMSE$, mais ajustado é o modelo. Outra métrica interessante é a variabilidade R^2 calculada entre as saídas experimentada e estimada, como mostra a equação 3.18.

$$R^2 = 1 - \frac{\text{var}(y - \hat{y})}{\text{var}(y)} \quad (3.18)$$

A variabilidade R^2 será um valor entre 0 e 1, sendo que quanto maior o valor, mais ajustado é o modelo.

⁸Este método é conhecido pela sigla em inglês *Root Mean Square Error (RMSE)*.

Um terceiro método é calcular o coeficiente de correlação entre a entrada e o erro residual como mostra a equação 3.19, sendo que quanto menor for a medida, mais ajustado é o modelo. Essa métrica busca explicar se há uma relação causa-efeito entre a entrada de controle e o erro.

$$r = \frac{\sum_{i=1}^n e(i)u(i)}{\sqrt{\sum_{i=1}^n e(k)^2 \sum_{i=1}^n u(k)^2}} \quad (3.19)$$

É claro que as métricas mostradas aqui não determinam se o modelo é ou não adequado definitivamente. Saber se um modelo está bom ou não simplesmente comparando os resultados encontrados para as métricas escolhidas é subjetivo e exige um pouco de conhecimento especialista por conta de quem modela o sistema.

O método de caixa preta de modelagem de sistemas dinâmicos proposto por Hellerstein et al. (2004) utiliza modelos estatísticos extraídos de séries temporais *AR*, *ARX* e *ARMA*, uma variação desse método pode ser encontrada em Parekh et al. (2002); Hellerstein e Parekh (2002). Além de modelos de séries temporais, métodos numéricos de interpolação polinomial como o método de *Lagrange* e o método de *Newton* podem ser usados para aproximar uma relação entre a entrada e saída de um sistema de caixa preta de um modelo.

3.5 Funções de transferência

Analisar sinais e sistemas no domínio do tempo nem sempre é uma tarefa fácil, algumas vezes nem mesmo é possível dada a dificuldade de manipulação de operações nesse domínio. A análise no domínio do tempo pode envolver complexas operações entre equações diferenciais e, sendo assim, uma estratégia usada por projetistas de controle é transformar as equações diferenciais de sistemas lineares invariantes no tempo para outro domínio em que seja mais fácil de manipulá-los.

Um método de transformação muito comum é a transformada de *Laplace* na qual sinais com funções complexas no domínio do tempo são transformados representações simplificadas no domínio da frequência. Nesse domínio as operações entre os sinais transformados tornam-se mais simples e podem ser resolvidas de maneira quase trivial com operações algébricas elementares. Por exemplo, um sinal descrito por uma função seno no domínio do tempo, $f(t) = \text{sen}(bt)$, com a transformada de *Laplace* seria transformado numa fração simples como mostra a equação 3.20.

$$\mathcal{L}\{f(t)\} = \mathcal{L}\{\text{sen}(bt)\} = F(s) = \frac{s}{s^2 + b^2} \quad (3.20)$$

Logo, uma operação entre senos no domínio do tempo resultariam em uma operação entre frações no domínio da frequência. A transformada de *Laplace* para uma função no domínio do tempo é dada pela equação 3.21 e mais informações podem ser obtidas em (Ogata, 2009).

$$\mathcal{L}\{f(t)\} = F(s) = \int_0^{\infty} e^{-st} f(t) dt \quad (3.21)$$

A transformada de *Laplace* é muito útil quando são os sinais em questão são contínuos, mas não é uma boa opção quando os sinais são discretos, avaliados segundo intervalos de tempo, como acontece na maioria dos sistemas digitais. Para esses casos, a melhor estratégia é usar um método análogo, a transformada Z (Jury, 1964). A transformada Z dá uma forma de codificar sinais no domínio Z equivalentes ao domínio do tempo, de forma que seja possível extrair informações sobre os sinais e o sistema (por exemplo, o valor final em regime estacionário e o tempo de ajuste) de forma mais fácil.

A transformada Z transforma um sinal $u(k)$, em que k é a quantidade de intervalos de amostra, em uma série infinita dos valores de $u(k)$ em termos da variável z . A equação 3.22 define a transformada Z para um sinal qualquer $u(k) = \{u(0), u(1), \dots\}$.

$$U(z) = u(0)z^0 + u(1)z^{-1} + u(2)z^{-2} + \dots = \sum_{k=0}^{\infty} u(k)z^{-k} \quad (3.22)$$

O sinal discreto $u(k)$ é equivalente e pode ser obtido de um sinal contínuo $u'(t)$ quando $u(k) = u'(kT_s)$, em que T_s é o intervalo de amostra.

Alguns sinais são particularmente importantes dada a utilidade e a proporção com que são encontrados na modelagem de sistemas discretos. Além disso, a combinação desses sinais cujas transformadas são conhecidas permite que outros sinais mais complexos sejam obtidos. Dentre os sinais mais comuns encontrados estão **impulso**, **degrau**, **rampa**, **exponencial** e **seno**. A tabela 3.1 traz a descrição desses sinais no domínio do tempo para $k \geq 0$ e suas transformadas.

Tabela 3.1 — Transformada Z dos principais sinais.

Sinal	Domínio do tempo (k)	Transformada Z
Impulso	$u(0) = 1$	$U(z) = 1$
Degrau	$u(k) = 1$	$U(z) = \frac{z}{z-1}$
Exponencial	$u(k) = a^k$	$U(z) = \frac{z}{z-a}$
Rampa	$u(k) = k$	$U(z) = \frac{z}{(z-1)^2}$
Seno	$u(k) = \text{sen}(k\theta)$	$U(z) = \frac{z \text{sen}(\theta)}{z^2 - (2\cos(\theta))z + 1}$

Operações algébricas simples podem ser feitas no domínio Z para combinar sinais por meio de suas transformadas em Z . Essas combinações podem ser facilitadas pelas propriedades da transformada mostradas na tabela 3.2.

A primeira propriedade da tabela 3.2 diz que a multiplicação de um sinal por um escalar no domínio do tempo resulta na multiplicação do mesmo escalar pela transformada do sinal no domínio Z . A segunda propriedade diz que a soma de dois sinais no domínio do tempo resulta na soma das transformadas de cada sinal no domínio Z . A terceira propriedade diz que o atraso de um sinal de n intervalos de amostra no domínio do tempo resulta na multiplicação de z^{-n} pela transformada do sinal sem o atraso no domínio Z . Por fim, a última propriedade diz que o deslocamento do sinal

Tabela 3.2 — Propriedades da transformada Z .

Propriedade	Domínio do tempo	Domínio Z
Multiplicação por escalar	$y(k) = au(k)$	$Y(z) = aU(z)$
Adição	$y(k) = u(k) + w(k)$	$Y(z) = U(z) + W(z)$
Atraso	$y(k) = u(k - n)$	$Y(z) = z^{-n}U(z)$
Deslocamento	$y(k) = u(k + n)$	$Y(z) = z^n U(z) - z^n u(0) - \dots - zu(n - 1)$

em n intervalos de amostra no domínio do tempo resulta na multiplicação de z^n pela transformada do sinal sem o deslocamento, subtraída do somatório $\sum_{i=0}^{n-1} z^{n-i} u(i)$.

Uma vez que um sinal foi transformado e por meio de sua análise considerações acerca das propriedades do sinal foram extraídas, é possível aplicar a transformada inversa, isto é, transformar do domínio Z de volta ao domínio do tempo, para obter-se a representação do sinal no tempo. É possível também que as transformadas de vários sinais sejam combinadas em Z e então a transformada inversa seja aplicada para obter uma representação combinação dos mesmos no domínio do tempo. A técnica mais comum para realizar a transformada inversa é o fatoramento da transformada de forma que seja possível identificar os sinais presentes na tabela 3.1, se necessário fatorando em frações parciais para simplificar a transformada em termos identificáveis.

A transformada não serve apenas para facilitar a manipulação de sinais. Manipulando as transformadas é possível encontrar relações entre os sinais de entrada e de saída de um sistema e, por meio dessas relações, compreender sua dinâmica. Uma função que dá a relação das transformadas da saída pela entrada do sistema é chamada de **função de transferência**. Por exemplo, seja o sistema ilustrado na figura 3.3(a) linear e invariante no tempo e supondo que a sua saída seja fornecida pela regressão linear de primeira ordem representada pela equação de diferença 3.23.

$$y(k + 1) = ay(k) + bu(k) \quad (3.23)$$

Aplicando-se a transformada na equação 3.23, obtém-se a equação 3.24.

$$\begin{aligned} zY(z) - zy(0) &= aY(z) + bU(z) \\ zY(z) - aY(z) &= bU(z) - zy(0) \\ Y(z)(z - a) &= bU(z) - zy(0) \end{aligned}$$

$$Y(z) = \frac{bU(z)}{(z - a)} - \frac{zy(0)}{(z - a)} \quad (3.24)$$

É possível perceber que, como há um deslocamento de um intervalo de amostra para $y(k + 1)$, pelas propriedades da tabela 3.2, a condição inicial $y(0)$ aparece na equação 3.24. Para encontrar

a função de transferência do sistema, as condições iniciais são consideradas nulas, logo, $y(0) = 0$. Substituindo as condições iniciais nulas na equação 3.24 e dividindo a transformada do sinal de saída $Y(z)$ pela transformada do sinal de entrada $U(z)$, obtém-se a equação 3.25 que representa a função de transferência do sistema G . Em situações em que as condições iniciais são não nulas, é possível obter a resposta do sistema usando a propriedade de superposição (Ogata, 2009).

$$Y(z) = \frac{bU(z)}{(z - a)}$$

$$G(z) = \frac{Y(z)}{U(z)} = \frac{b}{(z - a)} \quad (3.25)$$

Uma vez obtida a função de transferência de um sistema, as propriedades mais importantes no domínio do tempo podem ser obtidas analisando o denominador da função de transferência no domínio Z , como mostra a equação 3.26.

$$G(z) = \frac{N(z)}{D(z)} \quad (3.26)$$

A função numerador $N(z)$ e denominador $D(z)$ são representados por polinômios em Z . As raízes da equação obtida igualando o polinômio numerado a zero, $N(z) = 0$, são chamadas de **zeros**. O polinômio que representa o denominador $D(z)$ é comumente conhecido por **polinômio característico** do sistema e, quando $D(z) = 0$ tem-se a **equação característica**.

A equação característica é particularmente importante na análise do sistema por causa das suas raízes, sejam elas reais ou complexas. As raízes da equação característica são também chamadas de **polos** da função de transferência e permitem inferências importantes sobre um sinal ou sistema analisado. Para entender como um polo influencia a dinâmica do sistema é preciso entender como um polo se distribui no plano complexo. A figura 3.7 mostra algumas possíveis localizações de polos no plano complexo.

O círculo na figura 3.7 representa um círculo com raio unitário a partir da origem do plano e é bastante útil para analisar os efeitos da localização dos polos, enquanto $P1$, $P2$ e $P3$ são polos com diferentes localizações e magnitudes. A localização de um polo no plano complexo permite prever como será o comportamento da saída do sistema no domínio do tempo.

Com relação a localização dos polos da figura 3.7 é possível afirmar que, sendo p um dos polos polo de um sistema linear invariante no tempo no plano complexo:

- Se $|p| < 1$, então a saída do sistema no domínio do tempo converge para um valor em estado de regime estacionário. Isso equivale a dizer que o polo encontra-se dentro do círculo unitário. É o caso do polo $P2$.
- Se $|p| > 1$, então a saída do sistema no domínio do tempo cresce indefinidamente, quando o polo se encontra fora do círculo unitário. É o caso do polo $P3$.

Figura 3.7 — Representação de um polo no plano complexo.

- Se $|p| = 1$, então a saída do sistema no domínio do tempo é limitada, não converge, mas também não cresce indefinidamente. É o caso do polo *PI*.
- Se $p < 0$, então a saída do sistema no domínio do tempo oscila.

A saída do sistema no domínio do tempo também pode oscilar quando o polo é complexo, ou seja, não está no eixo real, ou então quando o sistema não é de primeira ordem. Além disso, quanto menor for a magnitude do polo, isto é, $|p|$ mais próximo de zero, mais rápida será a convergência da saída para um valor de regime estacionário.

O valor de regime estacionário é o valor para o qual o limite da saída converge quando k é muito grande ou tende ao infinito. É uma propriedade de muito interesse para entender a dinâmica do sistema depois de um período em regime transiente. A principal utilidade do valor de regime estacionário está em poder calcular o erro em regime estacionário, que é a diferença entre a entrada de referência e a saída medida em estado de regime estacionário. Segundo (Hellerstein et al., 2004), o valor em regime estacionário pode ser obtido pelo teorema 1.

Teorema 1 (Valor final) *Se todos os polos da função de transferência $(z - 1)F(z)$ estão dentro do círculo unitário, então*

$$\lim_{k \rightarrow \infty} f(k) = \lim_{z \rightarrow 1} (z - 1)F(z) \quad (3.27)$$

Segundo o teorema 1 se um sistema estável é representado pela função de transferência $F(z)$ e todos os polos da função de transferência $(z - 1)F(z)$ tiverem magnitude menor que 1, então o valor de regime estacionário desse sistema pode ser calculado pela equação 3.27.

A estabilidade é outra propriedade de interesse que pode analisada por meio da função de transferência. Há várias maneiras de classificar a estabilidade, uma bem aceita é a que relaciona a estabilidade de um sistema linear invariante no tempo com relação a limitação da entrada com consequente limitação da saída. Esse critério de estabilidade é conhecido por *BIBO*. Para que um sistema seja considerado estável, para qualquer entrada dentro de um limite, a saída deve também estar dentro de um limite. O teorema 2 formaliza esse critério de estabilidade.

Teorema 2 (BIBO) *Um sistema representado cuja função de transferência é $F(z)$ é estável pelo critério BIBO se, e somente se, todos os polos de $F(z)$ estão dentro do círculo unitário.*

Pelo teorema 2 é possível concluir que um sistema só será estável se todos os polos da sua função de transferência tiverem magnitude menor que 1.

Para um sistema dinâmico que se enquadre nos teoremas 1 e 2, uma alteração na entrada fará com que o sistema saia do estado de regime estacionário e entre num período em regime transiente. Como o sistema é estável, se a entrada for mantida em um novo valor, após alguns intervalos de amostra o sistema irá estabilizar e estacionar em um novo valor de saída, entrando novamente no estado de regime estacionário.

É possível calcular a taxa com que a saída mudou com relação a alteração da entrada. Essa taxa leva o nome de **ganho em regime estacionário**. O ganho em regime estacionário de um sistema pode ser calculado pela relação entre a saída em regime estacionário e a entrada constante que resulta em tal saída. Para descrever a dinâmica de um sistema, comumente, o ganho em regime estacionário é calculado para uma entrada degrau unitário. Para uma entrada degrau unitário, um sistema com função de transferência $G(z)$ terá seu ganho em regime estacionário quando $z = 1$, como mostra a equação 3.28.

$$\frac{y_r}{u_r} = G(1) \quad (3.28)$$

As funções de transferência facilitam muito o entendimento dos sinais e da dinâmica de um sistema. Manipulações permitem que conjuntos de blocos sejam simplificados na análise de um sistema de malha fechada. É possível reduzir toda a dinâmica de um sistema de malha fechada a uma ou poucas funções de transferência que descrevem a saída em relação as entradas. Por exemplo, tendo o sistema de malha fechada ilustrado na figura 3.4, as funções de transferência da saída em relação às entradas $r(k)$, $d(k)$, e $n(k)$ são dadas pelas equações 3.29, 3.30 e 3.31, respectivamente.

$$F_r(z) = \frac{Y(z)}{R(z)} = \frac{C(z)G(z)}{1 + C(z)G(z)H(z)} \quad (3.29)$$

$$F_d(z) = \frac{Y(z)}{D(z)} = \frac{G(z)}{1 + C(z)G(z)H(z)} \quad (3.30)$$

$$F_n(z) = \frac{T(z)}{Y(z)} = \frac{1}{1 + C(z)G(z)H(z)} \quad (3.31)$$

Para obter a função de transferência geral do sistema de malha fechada é possível, por superposição, combinar as equações 3.29, 3.30 e 3.31 na equação 3.32

$$F(z) = F_r(z) + F_d(z) + F_n(z) \quad (3.32)$$

As funções de transferência facilitam muito a análise de sinais e sistemas dinâmicos. Mesmo assim, há situações em que quando um sistema tem uma ordem muito grande, ou seja, vários polos, a análise pode ser complicada já que cada polo tem seu peso no comportamento do sistema. Para facilitar a análise de um sistema com muitos polos é possível usar aproximar o sistema para outro de primeira ordem com um único polo dito **polo dominante**. Não há uma definição formal para o conceito de polo dominante, mas uma definição muito aceita é de que o polo dominante é o polo de maior magnitude cuja magnitude é significativamente maior que os demais. Para sistemas com muitos polos em que exista um deles é dominante, a função de transferência aproximada para um sistema de primeira ordem pode ser obtida pela equação 3.33.

$$G'(z) = \frac{G(1)(1 - p')}{z - p'} \quad (3.33)$$

Propriedades de interesse como o tempo de ajuste e o máximo *overshoot* podem ser obtidas da função de transferência aproximada. É importante, no entanto, que a representatividade da aproximação seja avaliada, caso isso seja possível.

Para um sistema estável, o tempo de ajuste k_s é a quantidade de intervalos de amostra que o sistema leva para chegar suficientemente perto do valor de regime estacionário. Segundo (Ogata, 2009) um critério universalmente aceito é o critério de 2%. Esse critério define que o tempo de ajuste de um sistema é a quantidade de intervalos de amostra no período em regime transiente que o sistema leva até ficar a 2% do valor de regime estacionário. Em sistemas de primeira ordem ou sistemas com polo dominante, o tempo de ajuste pode ser aproximado pela equação 3.34, em que p é o polo dominante.

$$k_s = \frac{-4}{\log |p|} \quad (3.34)$$

Um projeto de controle adequado deve observar a amplitude máxima atingida pela resposta durante o regime transiente. Isso porque no regime transiente a amplitude pode levar a resposta a valores que vão além da capacidade suportada pelo sistema. A amplitude só é um problema quando a resposta no regime transiente atinge magnitudes maiores que o valor no regime estacionário após

o tempo de ajuste. Para um sistema em que $|y_{max}|$ é a maior magnitude da resposta atingida durante o regime transiente e $|y_r|$ é a magnitude do valor do regime estacionário, a amplitude máxima M_p é dada por 3.35.

$$M_p = \begin{cases} 0, & \text{se } |y_{max}| \leq |y_r| \\ \frac{|y_{max} - y_r|}{|y_r|}, & \text{se } |y_{max}| > |y_r| \end{cases} \quad (3.35)$$

Para um sistema de primeira ordem, o valor de M_p pode ser obtido diretamente pelo polo. Para sistemas de ordem maior, o valor da amplitude máxima pode ser obtido por meio da aproximação de um sistema equivalente com o polo dominante.

Hellerstein et al. (2004) define diferentes formas de estimar a amplitude máxima como base na natureza do polo dominante p , como mostra a equação 3.36.

$$M_p = \begin{cases} 0, & \text{se } p \text{ é real e } |p| \geq 0 \\ |p|, & \text{se } p \text{ é real e } |p| < 0 \\ r^{|\frac{\pi}{\theta}|}, & \text{se } p \text{ é complexo e } p = re^{j\theta} \end{cases} \quad (3.36)$$

3.6 Projeto de controle de sistemas discretos

Esta seção elenca uma sequência de etapas importantes que podem auxiliar no projeto de um sistema de controle de malha fechada para um sistema de computação, obtidos com a experiência na análise de controle para sistemas desse tipo.

O sequenciamento dessas etapas não é rígido e pode ser alterado segundo as necessidades do projeto. As etapas aqui enumeradas empregam técnicas bem desenvolvidas e conhecidas que podem ser úteis e auxiliar o projetista a obter controladores para sistemas computacionais.

Para que essas etapas sejam úteis, algumas limitações devem ser consideradas quanto às características do sistema objetivo para o qual se deseja projetar um controlador. Esse sistema deve *Linear Time Invariant (LTI)* e *SISO*.

3.6.1 Modelagem do sistema objetivo

Antes de mais nada é fundamental que o sistema computacional possa ser modelado matematicamente, isto é, deve ser possível exprimir a dinâmica do sistema por meio de uma função que represente a forma como o mesmo se comporta e responde a uma entrada variável. A modelagem faz-se necessária para que propriedades de interesse do sistema possam ser obtidas pela análise

do modelo. Sempre que for viável, a modelagem analítica do sistema é a melhor opção já que representa de fato sua dinâmica.

Sistemas reais quase sempre são não lineares, no entanto, a modelagem de sistemas lineares é muito mais simples de ser obtida, tendo que as manipulações algébricas são muito mais simples e que uma série de considerações sobre a modelagem são aplicáveis somente àqueles que são lineares. Certas não linearidades podem ser aproximadas para comportamentos lineares, desde que se considere a existência de um erro aleatório consequente da aproximação. Caso a existência do erro não seja decisiva na descrição do comportamento do sistema, a aproximação para um comportamento linear é uma boa alternativa e deve ser considerada pelo projetista. Alguns trabalhos mostram que a aproximação de sistemas não lineares para lineares é possível e apresenta bons resultados (Hellerstein e Parekh, 2002; Wang et al., 2005; Zhu e Agrawal, 2012; Padala et al., 2009).

Da mesma forma, a modelagem de sistemas variantes no tempo traz maiores dificuldades que a modelagem de sistemas invariantes no tempo. A existência de variações faz com que os coeficientes da função de transferência de um sistema não sejam constantes e, em se tratando de um sistema linear obtido por meio de uma regressão linear, é equivalente a dizer que uma observação passada influencia as respostas futuras de formas diferentes, com pesos diferentes. Assim, quando possível, toda variância no tempo deve ser eliminada, por exemplo, um sistema linear que possui uma variação com média constante e baixo desvio padrão (*e. g.* o tempo médio de processamento de uma requisição dado por uma distribuição do *Poisson*) poderia aproximar esse comportamento variante para um comportamento invariante no tempo sob o valor da média.

A existência de variações no sistema linear impede que funções de transferência sejam obtidas e possam ser usadas para descrever a dinâmica do sistema. Para casos em que as variações são persistentes uma aproximação possível é considerar mais de um sistema linear invariante no tempo para representar um sistema linear variante no tempo, e assim, ter vários projetos de controladores que se alternam de acordo com as características instantâneas do sistema variante. Se os resultados obtidos não forem adequados é preciso usar outra estratégia de modelagem alternativa às funções de transferência como a representação do sistema em espaço de estados (Paraskevopoulos, 2001).

A representação do sistema no espaço de estados permite descrever a dinâmica do sistema por meio de uma ou mais variáveis de estados que podem ser combinadas em vetores ou matrizes de estados. Toda a análise do sistema é feita no domínio do tempo com manipulações das variáveis envolvidas que não precisam ser necessariamente a saída medida do sistema. Na verdade, as variáveis de estado podem ser variáveis que sequer seja possível de mensurar. Algumas propriedades do sistema como a observabilidade e a controlabilidade só são analisáveis com a representação do sistema no espaço de estados. Além disso, representar o sistema em espaço de estados permite a análise de sistemas *Multiple Input, Multiple Output (MIMO)*, ao contrário das funções de transferência que servem apenas para análise de sistemas *SISO*.

Apesar do projeto de controle ser um etapa posterior a modelagem do sistema, é preciso que na modelagem já seja considerada as variáveis que representarão a entrada de controle e a saída

medida. Isso implica que o projetista deve ter em mente a forma como o sistema de controle reagirá para atingir o objetivo proposto. Uma vez que se tem conhecimento de como o sistema de controle vai interagir com o sistema objetivo, é possível aplicar o exposto na seção 3.4 para obter a representação matemática da planta.

3.6.2 Modelagem do sistema de malha fechada usando diagrama de blocos

A representação do sistema de malha fechada usando diagrama de blocos dá um panorama gráfico da interação entre os componentes envolvidos e não exige conhecimento específico sobre o comportamento do sistema. Por esses motivos, pode ser uma das primeiras etapas do projeto de um sistema de controle.

Uma vez que o sistema de malha fechada está representado em um diagrama de blocos e as funções de transferência dos envolvidos foi obtida, é possível combinar essas funções de transferência com o objetivo de entender a dinâmica do sistema ou mesmo simplificá-lo, reduzindo-o a uma quantidade menor de blocos. O diagrama de blocos permite que funções de transferência que englobam a dinâmica de mais de um bloco sejam obtidas. A figura 3.8 ilustra dois diagramas de bloco para um sistema de malha fechada, na figura 3.8(a) é ilustrado um sistema de malha fechada com dois blocos, sendo que, por meio de manipulações, é possível obter um sistema de malha aberta equivalente que pode substituí-lo, como mostrado na figura 3.8(b).

(a)

(b)

Figura 3.8 — Diagrama de blocos para um sistema de controle de malha fechada.

Em geral, um sistema de controle de malha fechada pode ser visto como dois sistemas de malha aberta simples, um sistema *avante*⁹ e um sistema *realimentado*¹⁰, como mostra a figura 3.9. Pela figura é possível perceber que todos os blocos da malha foram reduzidos a apenas dois blocos que são combinações das funções de transferência de cada um dos blocos que compõem a malha.

⁹tradução do termo em inglês *feedforward*.

¹⁰tradução do termo em inglês *feedback*.

Figura 3.9 — Diagrama de blocos representando os sistemas *feedforward* e *feedback*.

3.6.3 Projeto do controlador

O objetivo do projeto de um sistema de controle de malha fechada é desenvolver um controle automático que seja capaz de fazer com que o sistema se adapte a eventuais perturbações e ruídos. Alguns controladores são bastante populares e muito úteis na resolução de vários problemas, todavia, não há um controlador ideal para todas as situações e, sendo assim, o projeto do controlador para um sistema objetivo deve ser analisado caso-a-caso. Da mesma forma, não há um método único para o desenvolvimento de um controlador mais adequado e o bom andamento e a boa adequação do projeto do controlador vão depender em muito da experiência do projetista sobre os sistemas envolvidos. Esta seção contém a descrição em linhas gerais de passos do projeto de um sistema de controle de malha fechada e também enumera algumas ferramentas que podem auxiliar no projeto.

Antes de mais nada é preciso saber que tipo de reação se espera do sistema quando o controlador estiver ativo. Isso significa estabelecer a forma como o controlador atuará para atingir o objetivo de controle proposto. É possível optar por uma reação rápida do controlador a qualquer variação da carga de trabalho imposta, ou então uma reação mais lenta, eliminando a influência de picos isolados e de pequenas variações da carga, também é possível que se queira que o controlador faça com que o sistema o máximo possível imune às perturbação ou ruído, entre outros comportamentos desejados. Para cada tipo de reação esperada, uma política de controle deve ser empregada para atingir os objetivos do controlador. As principais políticas de controle são: **proporcional, derivativa e integral**.

Na política proporcional, a intensidade da reação do controlador é proporcional ao erro entre a entrada de referência e a saída medida, ou a saída do transdutor, isto é, quanto maior o erro calculado, maior será a entrada de controle imposta a planta. É importante notar que, para um controlador proporcional, a entrada de controle é diferente de nula se, e somente se, o erro calculado também for diferente de nulo. Caso não exista erro, não há entrada de controle e a inexistência de uma entrada de controle provocará a aparição de um erro. Logo, um controlador proporcional só funciona na existência de um erro não nulo que causará um efeito contrário para que o erro seja corrigido, ciclicamente. O controlador proporcional provê uma forma rápida de reação do sistema em regime transiente, com baixo tempo de assentamento já que reage à proporção do erro

causando, no entanto, a existência de um erro no regime estacionário. A entrada de controle em função do tempo para a política proporcional e a função de transferência para um bloco controlador proporcional, cujo ganho é dado por K_p , são dadas pelas equações 3.37 e 3.38, respectivamente.

$$u(k) = K_p e(k) \quad (3.37)$$

$$C(z) = \frac{U(z)}{E(z)} = K_p \quad (3.38)$$

A política derivativa faz com que a entrada de controle seja proporcional não ao erro calculado, mas a sua variação. Nessa política, a variação do erro é vista como uma tendência que deve ser considerada na reação do controlador. Um sistema cujo erro varia pouco é visto como um sistema com pouca tendência a alteração e, por isso, necessita de pouca correção. Por outro lado, um sistema que tem variações abruptas da saída e, consequentemente, do erro calculado, pode indicar uma tendência de aumento na magnitude do erro ocasionada por uma perturbação, o controlador então tenta antecipar a reação com base nessa possível tendência. A entrada de controle em função do tempo para a política derivativa, bem como a função de transferência para o controlador derivativo, cujo ganho é dado por K_d , são dadas pelas equações 3.39 e 3.40, respectivamente.

$$u(k) = K_d [e(k) - e(k-1)] \quad (3.39)$$

$$C(z) = \frac{U(z)}{E(z)} = K_d \frac{z-1}{z} \quad (3.40)$$

Já na política integral, a variação da entrada de controle é proporcional ao erro calculado. Um controlador integral funciona como um acumulador que mantém a entrada de controle constante mesmo quando o sistema não apresenta erro algum. A entrada de controle só será alterada se houver uma variação entre o valor da saída medida e a entrada de referência, sendo a alteração da entrada proporcional a essa variação. A entrada de controle para a política integral e a função de transferência do controlador integral, cujo ganho é dado por K_i , são dadas pelas equações 3.41 e 3.42, respectivamente.

$$u(k) = u(k-1) + K_i e(k) \quad (3.41)$$

$$C(z) = \frac{U(z)}{E(z)} = K_i \frac{z}{z-1} \quad (3.42)$$

Cada uma das políticas de controle faz com que o sistema apresente um comportamento distinto em relação às demais. A política integral tem como característica levar o sistema a um regime estacionário em que o erro calculado é nulo e anomalias pontuais na resposta do sistema são absorvidas, no entanto, o tempo de ajuste necessário para atingir um novo regime estacionário

geralmente é maior que aquela das outras políticas, mostrando que a política do controlador integral é um pouco mais lenta. A política proporcional faz com que o controlador tenha uma reação mais rápida que a integral, com menor tempo de assentamento, mas em consequência, mantém um erro quando atinge o regime estacionário. A política derivativa apresenta uma reação ainda mais rápida que a política proporcional, com menor tempo de ajuste, mas sob a pena de poder elevar a amplitude máxima do sistema para níveis insuportáveis, já que qualquer anomalia gerada por um ruído aleatório ou perturbação pode causar uma reação superestimada e gerar grandes oscilações na resposta do sistema.

A escolha da política mais adequada depende, então, de que comportamento se espera do controlador quando o mesmo estiver ajustando a planta e, muitas vezes, optar por uma única política de controle pode não ser o suficiente. Em casos assim, uma alternativa viável é a combinação de várias políticas para que o objetivo de controle possa ser atingido, como é o caso dos controladores *Proportional-Derivative (PD)*, *Proportional-Integral (PI)* e *PID*. Esses controladores atuam sob a combinação das políticas vistas previamente e suas funções de transferência podem ser obtidas pela combinação das funções de transferência das políticas de controle obedecendo as propriedades expostas na tabela 3.2. As funções de transferência para as políticas *PD*, *PI* e *PID* são dadas pelas equações 3.43, 3.44 e 3.45, respectivamente.

$$C(z) = \frac{U(z)}{E(z)} = K_p + K_d \frac{z-1}{z} \quad (3.43)$$

$$C(z) = \frac{U(z)}{E(z)} = K_p + K_i \frac{z}{z-1} \quad (3.44)$$

$$C(z) = \frac{U(z)}{E(z)} = K_p + K_d \frac{z-1}{z} + K_i \frac{z}{z-1} \quad (3.45)$$

Tendo que a política de controle foi escolhida, é preciso projetar o controlador de acordo com o comportamento desejado. Isso significa dizer que os parâmetros do controlador devem ser estimados de acordo com o que se espera como comportamento do sistema de malha fechada nos regimes estacionário e transiente.

A resposta em regime transiente depende basicamente da localização dos polos, no plano complexo, da função de transferência do sistema de controle de malha fechada. Os parâmetros aplicados às políticas de controle são tratados como ganhos que têm a capacidade de alterar a localização no plano complexo levando esses polos às regiões que apresentam as propriedades requeridas. Tratando um ganho como um parâmetro variável na fase de projeto do controlador, a escolha do melhor valor para o ganho é fundamental para o sucesso do controlador. É importante notar que a variação do ganho implica diretamente na movimentação dos polos da função de transferência do sistema de controle de malha fechada no plano complexo.

Um método muito utilizado para analisar a influência de um ganho no comportamento de um sistema é o **método do lugar das raízes**. A ideia básica por trás do método consiste em representar

graficamente as raízes de uma equação, variando um parâmetro do sistema, no caso o ganho do controlador, de zero a infinito. Com esse método é possível analisar graficamente como as raízes de uma equação se movimentam no plano complexo para um ganho qualquer que pode ser considerado o ganho do controlador. Como os polos são as raízes da equação característica, o método pode ser aplicado diretamente no denominador da função de transferência do sistema. Nesse caso, todo o projeto de controle resume-se a escolher no gráfico o valor do ganho que coloca as raízes no lugar que melhor expressas as propriedades de interesse. Ferramentas de manipulação matemática como Octave¹¹ e Matlab¹² podem ser usados para auxiliar no processo de geração do gráfico do método, uma vez que consiste de um método numérico. Uma das grandes vantagens do método do lugar das raízes é que todas as possíveis combinações são ilustradas num único gráfico que cobre todos os valores possíveis do ganho analisado. Nota-se, no entanto, que o método funciona adequadamente quando há um único parâmetro a ser estimado, o que significa um controlador que atua com base em uma única política de controle. Para uma combinação de políticas, o método deve ser adaptado para a análise de um ganho por vez. A adaptação consiste em fixar alguns dos parâmetros em valores prováveis e variar aquele que se pretende estimar.

Analisar mais de um ganho no gráfico de lugar das raízes não é uma tarefa simples e pode ter como consequência uma estimativa incorreta dos valores. Para casos em que há mais de um parâmetro a ser estimado, outras técnicas alternativas podem ser aplicadas.

Uma forma de estimar os parâmetros de um controlador é por meio do **método empírico**. No método empírico sistema de malha fechada é experimentado em diversas situações para diferentes valores para cada um dos parâmetros envolvidos. A resposta do sistema e as propriedades de interesse são avaliadas para cada execução e, com base nesses dados, o conjunto que apresenta os resultados mais significativos é escolhido. É claro que o método empírico requer maior conhecimento especialista do projetista do sistema para determinar quais são os valores mais prováveis dos parâmetros e de que forma devem ser agrupados no planejamento de experimentos.

Outra forma de estimar os parâmetros de um controlador é usando o **método de colocação dos polos**. Esse método é muito simples e resulta em bons resultados sempre que puder ser aplicado. Os polos da função de transferência de malha fechada são determinados com base nas propriedades de regime transiente que se deseja para o mesmo. Isto é, por meio do tempo de assentamento e da amplitude máxima desejada para o sistema no regime transiente, é possível determinar quais valores de polos são mais adequados para que os objetivos de controle sejam cumpridos. Visto que os polos foram escolhidos, utilizam-se os mesmos para estimar os parâmetros de controle dadas as restrições nas propriedades de interesse. O método consiste das seguintes etapas:

- **Determinação dos valores da máxima amplitude e tempo de ajuste e cálculo dos valores dos polos**

O processo começa com a escolha das características comportamentais que se espera que o

¹¹<http://www.gnu.org/software/octave>

¹²<http://www.mathworks.com/products/matlab>

sistema de malha fechada presente durante o regime transiente, isto é, consideram-se a máxima amplitude que será suportada e qual o tempo de assentamento que o sistema apresentará até que um novo regime estacionário seja atingido. Com base nesses valores, determinam-se os valores dos polos que atingem suficientemente esses comportamentos e que não ferem a estabilidade do sistema, garantindo que a amplitude de todos os polos está dentro do círculo unitário. É interessante que a existência de um polo dominante seja considerada, já que o polo dominante é responsável pelos principais comportamentos do sistema. Nesse momento já é possível verificar, pelos valores dos polos encontrados, se o comportamento desejado é alcançável ou se alguma consideração (*e. g.* um relaxamento das características almejadas com o aumento do tempo de ajuste) é necessária para tornar o controle possível.

- **Construção do polinômio característico com os polos encontrados**

Tendo os polos é possível construir um polinômio característico com a ordem desejada no domínio Z . Esse polinômio pode ser obtido pela expansão da multiplicação dos fatores $(z - p_i)$, onde p_i é i -ésimo polo calculado, com $i = 1, \dots, n$ e n é a ordem do polinômio característico.

- **Construção da função de transferência do sistema modelado**

Visto que o diagrama de blocos que representa o sistema já deve estar disponível nessa etapa, é possível construir a função de transferência do sistema de controle de malha fechada em função dos parâmetros do controlador. A função de transferência da malha fechada pode ser obtida pelas funções de transferência da malha aberta, como mostra a equação 3.46, em que F_{ff} é a função de transferência do sistema *feedforward* e F_{fb} é a função de transferência do sistema *feedback*. Sendo assim, o polinômio característico (*i. e.* o denominador da função de transferência da malha fechada) pode ser obtido pela equação 3.47. Uma vez que o polinômio característico envolve ambas as funções de transferência da malha aberta, é fácil perceber que os parâmetros que representam os ganhos do controlador (*e. g.* K_p , K_d e K_i) fazem parte do polinômio característico modelado.

$$F_r(z) = \frac{N(z)}{D(z)} = \frac{F_{ff}}{1 + F_{ff}F_{fb}} \quad (3.46)$$

$$D(z) = 1 + F_{ff}F_{fb} \quad (3.47)$$

- **Estimação dos ganhos do controlador**

Tendo que os polinômios característicos com os polos desejados e modelado foram construídos, a comparação entre esses dois polinômios permite estimar o valor dos ganhos desde que os polinômios tenham a mesma ordem. Os ganhos estimados são então introduzidos no controlador para que o mesmo provoque o efeito planejado inicialmente quando foram escolhidos o tempo de ajuste e a máxima amplitude em regime transiente.

Parte IV

Desenvolvimento, Experimentos e Resultados

Arquitetura de Broker de Consumidor com Gerenciamento de Desempenho Inspirado em Controle Automático para Ambientes de Computação em Nuvem

Este capítulo descreve a arquitetura proposta neste trabalho e explica com detalhes os principais módulos que a compõem. O objetivo desta arquitetura é fornecer um *framework* para implementação de mecanismos automáticos de gerenciamento de desempenho de um conjunto de recursos físicos, contratados junto a um provedor de nuvem, por um consumidor de *IaaS*, que utiliza a infraestrutura contratada para entregar algum tipo de serviço (*i. e PaaS* ou *SaaS*) a seus clientes ou usuários finais.

No final deste capítulo há uma seção onde é apresentado um método de projeto de controle desenvolvido durante este projeto de pesquisa para sistemas computacionais lineares e invariantes no tempo que consideram a suavização da resposta por média móvel. Esse método reduz a complexidade das funções de transferência a um sistema de equações lineares que relaciona os parâmetros do projeto.

4.1 A Arquitetura

O objetivo desta arquitetura é fornecer uma estrutura para implementação de mecanismos automáticos de gerenciamento de desempenho de um conjunto de recursos físicos, contratados junto a um provedor de nuvem, por um consumidor de *IaaS*, que utiliza a infraestrutura contratada para entregar algum tipo de serviço (*i. e PaaS* ou *SaaS*) a seus clientes ou usuários finais.

Nesse formato de negócio, o provedor de nuvem detém a propriedade de recursos computacionais altamente escaláveis dispostos em *data centers* geograficamente dispersos acessíveis pela Internet. Comumente, o provedor aluga servidores virtualizados nessa infraestrutura física para seus consumidores em um modelo de pagamento por uso. Nesse modelo os consumidores são onerados apenas quando os recursos, frequentemente representados por máquinas virtuais, estão sendo utilizados. Por sua vez, o consumidor dos serviços de nuvem aplica as máquinas virtuais contratadas para oferecer algum tipo de serviço *Web* para usuários finais, eventualmente, cobrando esses usuários pelo acesso ao serviço dando em troca a garantia da qualidade do serviço prestado.

Há diferenças substanciais entre os valores cobrados pelo provedor para máquinas virtuais ligadas e em execução, que utilizam processamento, memória e rede; e desligadas, quando utilizam apenas espaço em disco para armazenamento. Usualmente, os custos relativos ao armazenamento são mais baixos que para os demais recursos que podem ser adquiridos junto ao provedor, sendo dispendioso manter máquinas virtuais ligadas quando as mesmas não são necessárias no atendimento da demanda gerada pelas requisições enviadas pelos usuários do serviço.

Para ter um rendimento adequado, o consumidor deve manter, ou mesmo otimizar, um ponto de equilíbrio entre a quantidade de recursos consumidos pelos quais está pagando e a demanda gerada por seus clientes pela qual está cobrando, além de que o descumprimento dos contratos junto aos usuários finais pode ocasionar a redução dos ganhos. Para isso, é necessário gerenciar o desempenho dos recursos disponíveis para que haja um mínimo perdas por descumprimentos de contratos. A figura 4.1 ilustra em alto nível os principais componentes da arquitetura proposta.

A arquitetura considera um ambiente usual de computação em nuvem como mostrado na figura 1.1, com duas formas principais de interação entre as partes envolvidas: **negócio** e **comunicação**. A interação que compõe a linha do negócio refere-se ao que é estabelecido por contrato entre os envolvidos e motiva a orientação a mercado dada para arquitetura, já que usualmente os serviços em nuvem são pagos. Como a arquitetura proposta é orientada ao mercado, alguns módulos são comuns aos propostos por Buyya et al. (2008) para um controle de admissão em *data centers*.

A linha da comunicação envolve o envio de requisições pelos usuários e de respostas pelas máquinas virtuais no *data center*. O consumidor não participa dessa interação que é feita diretamente entre os clientes que acessam os serviços hospedados na infraestrutura contratada.

Nesse ambiente estão envolvidas as seguintes entidades:

- **Usuários finais:** Usualmente, os usuários finais são consumidores de um serviço do tipo *PaaS* ou *SaaS*, dessa forma, são clientes do consumidor de *IaaS* e enviam requisições dire-

Figura 4.1 — Arquitetura proposta em alto nível.

tamente às máquinas virtuais do *data center* para que sejam processadas pelos serviços lá hospedados.

- **Broker:** O *Broker* é o ator principal do ambiente e integra o mecanismo automático de gerenciamento de desempenho. O *Broker* é um sistema computacional autônomo que, uma vez disposto, representa os interesses de um consumidor de nuvem que contrata um serviço do tipo *IaaS*, com o objetivo de garantir que os parâmetros de desempenho configurados serão cumpridos e, conseqüentemente, o rendimento será adequado.
- **Máquinas virtuais:** As máquinas virtuais compõem o recurso de granularidade grossa que será provido dinamicamente pelo *data center*. Constituem o principal recurso entregue em serviços do tipo *IaaS*. As máquinas são ligadas e desligadas, em número, conforme julga necessário o *Broker* para atingir os objetivos de gerenciamento de desempenho. Diversas métricas podem ser analisadas para verificar o desempenho das máquinas virtuais como o tempo de inicialização, a taxa de utilização, o tempo médio de resposta e o *throughput*. Usualmente, as máquinas virtuais executam servidores de aplicação que hospedam algum tipo de serviço de interesse e respondem a uma carga de trabalho imposta por usuários através de requisições. A alocação e distribuição dessas máquinas virtuais entre os recursos físicos disponíveis é de responsabilidade do *data center*. Abordagens globais de gerenciamento de desempenho podem considerar o balanceamento de carga entre as máquinas físicas e virtuais, mas está fora do escopo deste trabalho.

- **Máquinas físicas:** As máquinas físicas constituem o conjunto de recursos físicos compartilhados que serão atribuídos a instâncias de máquinas virtuais pelo *data center*.

A arquitetura proposta não prevê, mas pode ser implementada em conjunto com um controle de admissão, como o proposto por Wu et al. (2012), caso seja necessário para manter a quantidade de requisições ou usuários conectados ao sistema dentro dos limites suportados para atingir cumprir com os contratos estabelecidos.

O *Broker* é composto por um conjunto de módulos com funções distintas e complementares. As próximas seções descrevem os módulos da arquitetura proposta e suas principais funcionalidades.

4.1.1 Monitor de máquinas virtuais

A quantidade de máquinas virtuais ligadas é fundamental no rendimento percebido pelo consumidor, visto que esse número é variável da função custo que dá o quanto o consumidor deve pagar ao provedor de nuvem. Quanto maior o número de máquinas virtuais ligadas, maior é a quantidade de recursos contratados e sujeitos a cobrança. Em contrapartida, quando for insuficiente para atender a demanda de requisições enviadas, a quantidade de máquinas virtuais ligadas influenciará diretamente no tempo médio de resposta que, segundo Xiong et al. (2010), é a principal medida de desempenho sentida pelos usuários. Com menos instâncias de máquinas virtuais que o necessário para atender a demanda, haverá uma contenção maior pelos recursos que incorrerá na formação de filas de requisições nos servidores. O aumento das filas têm como consequência o atraso no atendimento das requisições que culmina no aumento do tempo médio de resposta.

Para aplicações cujo padrão de utilização de seus usuários é bem definido e previsível pode-se considerar que a aplicação permanece em constante regime estacionário. Uma vez nessa condição, o correto planejamento de capacidade pode levantar um número de instâncias suficientemente próximo do ideal para atender a demanda da aplicação. Nem todas as aplicações apresentam comportamento previsível, para aplicações que não possuem um padrão de utilização previsível, fixar uma quantidade de instâncias quase sempre provocará uma subestimação ou superestimação da capacidade dos recursos. Com base no padrão de utilização e na previsibilidade do comportamento dos usuários, Klems et al. (2009) classificam diversas aplicações como previsíveis ou não. Para casos em que a carga de trabalho das aplicações não é previsível, a estratégia usada neste trabalho apresenta resultados significativos quanto ao gerenciamento de desempenho.

Atualmente, grandes provedores de nuvem fornecem na *API* de acesso funções que possibilitam o gerenciamento do ciclo de vida das máquinas virtuais. Essas funções permitem que novas instâncias de máquinas virtuais sejam criadas, removidas, ligadas e desligadas de acordo com a configuração pretendida e os serviços desejados.

O monitor de máquinas virtuais deve manter o controle sobre o ciclo de vidas das instâncias de máquinas virtuais de responsabilidade do consumidor no *data center*, em outras palavras, o *Broker*. A figura 4.2 ilustra o ciclo de vida que a arquitetura proposta considera para manipulação das instâncias de máquinas virtuais, com o intuito de gerenciar o desempenho do sistema.

Figura 4.2 — Diagrama de estados que representa o ciclo de vida de uma máquina virtual

Conforme ilustra a figura 4.2, existem, basicamente, quatro possíveis estados em que uma máquina virtual pode se encontrar durante todo o seu ciclo de vida no provedor:

- **Desligada:** A instância de máquina virtual existe, está desligada e não consome recursos físicos do *data center*, exceto espaço de armazenamento. Uma máquina virtual desligada pode ser ligada tão logo o *Broker* considere adequado.
- **Inicializando:** Uma máquina virtual está inicializando quando a instância já foi ligada e encontra-se em processo de inicialização do sistema operacional e dos serviços necessários. Neste estado a máquina virtual já consome recursos, mas ainda não está disponível para atender às requisições do usuário.
- **Ligada:** A máquina virtual está ligada e pronta para processar as requisições que forem recebidas. Nesse momento, o *Broker* considera que a instância faz parte do seu conjunto de recursos e começa a enviar requisições para a mesma.

- **Desligando:** Neste estado se encontram as máquinas virtuais que foram consideradas sobressalentes pelo *Broker* e foram marcadas para desligar tão logo seja possível. As máquinas neste estado não são desligadas imediatamente porque ainda possuem processamento ou possuem requisições em fila. Essas máquinas permanecem consumindo recursos da mesma forma que as máquinas ligadas, mas deixam de receber carga.

Enquanto no estado **Desligada** o consumidor não é cobrado pela instância da máquina virtual, exceto, é claro, pela espaço em disco que a mesma ocupa. Já nos estados **Inicializando**, **Desligando** e **Ligada** a instância já consome recursos físicos do *data center* e, por isso, o consumidor já é cobrado pelo processamento, memória e, quando aplicável, taxa de transferência, independente das máquinas virtuais estarem ocupadas com o processamento de requisições ou simplesmente ociosas.

A figura 4.2 também ilustra sete possíveis mudanças de estado representadas por setas numeradas. As setas numeradas de 1 a 3 refletem as mudanças de estado de um ciclo de vida padrão de uma máquina virtual ilustrando o fluxo normal de estados que uma instância percorre, enquanto as setas numeradas de 4 a 7 mostram as mudanças de estado alternativas ao fluxo normal. Essas mudanças foram adicionadas ao ciclo de vida padrão para facilitar o gerenciamento de desempenho de muitas instâncias.

As alterações nos estados das máquinas virtuais são provocadas pelas seguintes mudanças:

- **Desligada para Inicializando (1):** Ocorre quando o *Broker* entende que a quantidade de máquinas virtuais disponíveis é insuficiente para atender a demanda atual e então inicia novas instâncias para corrigir esse déficit.
- **Inicializando para Ligada (2):** Uma vez ligada, a máquina virtual passa por um processo de inicialização. Enquanto nesse estado, o consumidor já é cobrado pelos recursos ocupados pela instância, mas a mesma ainda não se encontra disponível para atender a demanda emergente que ocasionou o seu ligamento. Nesse período, o sistema operacional escolhido e os serviços essenciais são inicializados. Com as tecnologias atuais de virtualização o processo de inicialização de uma máquina virtual assemelha-se em tempo ao de uma máquina física de mesma configuração. Esse tempo é muito variável e dentro de uma infraestrutura de nuvem podendo ultrapassar sessenta segundos (Fito et al., 2010). Para casos em que as máquinas virtuais são inicializadas por meio de estados salvos anteriormente, esse tempo de inicialização pode ser reduzido.
- **Ligada para Desligada (3):** Essa mudança ocorre quando o *Broker* identifica que a quantidade de instâncias ligadas é superior à demanda e ajusta essa diferença mandando que

algumas máquinas virtuais ociosas sejam desligadas. Ao contrário de quando são ligadas, não há atraso para que uma máquina ociosa seja desligada, a menos que a mesma tenha que executar procedimentos internos para tal. Caso contrário, o fato de terem seus sistemas extraídos de imagens, permite que o desligamento seja instantâneo.

- **Inicializando para Desligada (4):** Quando é necessário reduzir a quantidade de instâncias, o *Broker* começa encerrando as máquinas que ainda não estão completamente disponíveis e evita que máquinas ocupadas sejam marcadas para desligar. Isso garante que as primeiras máquinas a serem desligadas serão as ociosas, caso existam.
- **Ligada para Desligando (5):** Visto que é necessário reduzir a quantidade de instâncias e não há mais máquinas ociosas para serem desligadas, o *Broker* seleciona aquelas com menor carga e marca as mesmas para que sejam desligadas tão logo seja possível. Uma vez no estado **Desligando** a instância deixa de receber novas requisições.
- **Desligando para Ligada (6):** Quando é necessário aumentar a quantidade de máquinas virtuais ligadas, antes que novas máquinas sejam ligadas, o *Broker* primeiro verifica se há instâncias no estado **Desligando**, ou seja, máquinas virtuais que seriam desligadas mas ainda não o foram. Em caso afirmativo, o *Broker* cancela o desligamento dessas máquinas e altera seu estado para Ligada, fazendo com que elas voltem a receber carga. Esse procedimento permite o aumento do número de instâncias sem atraso já que evita o ligamento de novas máquinas que passariam obrigatoriamente pelo processo de inicialização do sistema.
- **Desligando para Desligada (7):** Uma vez que uma máquina virtual se encontra no estado Desligando e acabou de processar as requisições que estavam pendentes, a mesma é desligada e deixa de consumir recursos.

O monitor de máquinas virtuais deve manter listas sempre atualizadas sobre as instâncias em cada um dos estados ilustrados, já que o estado influi diretamente na disposição ou não em receber carga de trabalho e no consumo das capacidades computacionais. O monitor prevê a implementação de algoritmos para ligamento e desligamento das instâncias que otimizem esses processos por meio das instâncias que se encontram nos estados Inicializando e Desligando. Esses algoritmos devem considerar que uma nova instância só será ligada se, e somente se, não houver máquinas no estado Desligando que podem ser passadas ao estado Ligada. Da mesma forma, nenhuma máquina virtual ocupada deve ser Desligada, a menos que não existam mais instâncias no estado Inicializando que possam ser desligadas.

Além de gerenciar o ciclo de vidas dos recursos, o monitor é responsável por manter algumas informações de interesse disponíveis aos outros módulos da arquitetura, como a taxa de utilização do sistema e a quantidade de tempo que cada instância permaneceu nos estados que cobram o consumidor.

4.1.2 Monitor de requisições

A carga de trabalho é imposta ao sistema por meio de requisições enviadas pelos usuários finais aos serviços (*e. g.* um serviço *Web*) que executam nos servidores de aplicação das máquinas virtuais instanciadas no *data center*. Essas requisições exigem que as máquinas virtuais se ocupem pelo tempo necessário para processá-las, alterando o desempenho experimentado pelo sistema.

Dois fatores que envolvem uma requisição afetam diretamente o desempenho do sistema: o tempo de processamento e a quantidade de carga imposta pelas requisições, dados pelo tempo de processamento e pela taxa de chegada de novas requisições, respectivamente. É esperado que com requisições de tamanho e intervalo entre as chegadas aproximadamente fixos, ou distribuídos em torno de uma média, o sistema apresente desempenho aproximadamente constante, que constitui o caso típico de análise de desempenho usando teoria de filas.

Com o tempo, a quantidade e o tamanho das requisições podem se alterar, dependendo do perfil de utilização dos usuários que utilizam o serviço naquele momento. Havendo um aumento em algum desses fatores é possível que o desempenho do sistema sofra degradação, podendo, em casos extremos, entrar em colapso, descumprir com os requisitos de *QoS* e não atingir os objetivos de desempenho previstos.

O monitor de requisições tem por objetivo analisar as requisições que chegam, que são processadas e que deixam o sistema. Dessa forma, o monitor de requisições consegue manter informações precisas sobre a taxa em que novas requisições chegam, o tempo de processamento exigido e o quanto as requisições permanecem no sistema, inclusive esperando em fila, até que sejam respondidas. Além de informações individuais sobre as requisições, este monitor também pode manter informações sumarizadas em variáveis de interesse do desempenho como o *throughput*, o tempo médio de resposta, a quantidade de requisições que ocupam o sistema ou que já foram processadas, o número de requisições que não atenderam os parâmetros de *QoS*, entre outras.

4.1.3 Repositório de SLA

O *SLA* explicita todos os parâmetros acordados entre partes envolvidas em uma negociação. Para cada tipo de negociação, informações que atribuem direitos, responsabilidades, penalidades e custos podem fazer parte do contrato. Neste trabalho foram utilizados dois tipos diferentes de contratos de prestação de serviço para o ambiente de negócio proposto.

O primeiro tipo de contato contém os parâmetros estabelecidos entre o consumidor e provedor de nuvem, e modera a disponibilização e o acesso à infraestrutura requerida. Esse contrato deve conter informações sobre a quantidade, a discriminação dos recursos virtualizados adquiridos, o custo e forma de cobrança. Além disso esse contrato pode prever a existência de elasticidade dos recursos em tempo de execução e quais são as regras (por exemplo, quanto tempo uma nova máquina virtual leva até estar totalmente disponível para uso, qual o número máximo e mínimo de instâncias que podem estar ligadas) para esses recursos que possam ser dinamicamente alocados.

Na outra ponta do negócio está o segundo tipo de contrato considerado. Esse contrato regula a interação entre os usuários finais e o consumidor, e deve deixar explícito os parâmetros de *QoS* com os quais o usuário pretende ser atendido. Outrossim, o contrato deve prever os custos e penalizações comportados quando há descumprimento, bem como as situações que levam a uma penalização.

O repositório de *SLA* tem como função básica manter uma base dos contratos e disponibilizar as informações lá contidas para que outros módulos da arquitetura tenham ciência das regras estabelecidas e possam pautar suas ações dentro do que foi combinado entre as partes envolvidas. A princípio o módulo não é capaz de detectar a violação dos contratos que, na arquitetura proposta, é realizada pelos monitores, todavia, é possível implementar o monitoramento e as verificações de quebra de contrato neste módulo. Para isso, uma alternativa viável é implementar a arquitetura proposta por Emeakaroha et al. (2012), que faria do módulo um monitor de *SLA*.

4.1.4 Monitor de rendimento

O termo rendimento, nesta arquitetura, tem foco na entidade consumidor. O rendimento pode ser entendido como o lucro, o montante residual da diferença entre o quanto o consumidor paga ao provedor de nuvem por uma infraestrutura descrita no *SLA*, e o quanto o consumidor recebe de seus clientes pelo acesso ao serviço sempre que os parâmetros de *QoS* estabelecidos em contrato são obedecidos, além do quanto o consumidor deixa de arrecadar quando os parâmetros de *QoS* não são obedecidos. Para fins de cálculo do rendimento é interessante que seja definida e implementada uma função lucro com base nas informações de contrato, como em Xiong et al. (2010). A equação 4.1 mostra um exemplo simples de cálculo de rendimento considerando apenas as requisições atendidas e a quantidade de recursos utilizados.

$$\begin{aligned}
 R &= Wc - Mp \\
 P &= \sum_{i=1}^n T(i)C(i) \\
 L &= R - D
 \end{aligned}$$

$$L = \overbrace{Wc - Mp}^R - \underbrace{\sum_{i=1}^n T(i)C(i)}_P \quad (4.1)$$

Onde:

- L é o lucro para o consumidor.
- R é o quanto foi recebido dos usuários finais.
- D é o quanto foi gasto com os recursos contratados.
- n é o total de recursos utilizados independente da granularidade com que são providos.
- $T(i)$ é a quantidade em unidades de tempo (i . e. a unidade de tempo e a carência combinada) que o recurso i permaneceu ativo e sujeito a cobrança.
- $C(i)$ é o quanto o consumidor paga ao provedor por unidade de tempo que o recurso permaneceu ligado. Esse custo está definido no *SLA*.
- W é o número de requisições bem atendidas.
- c é o valor pago por requisição bem atendida.
- M é a quantidade de requisições penalizadas.
- p é o valor por requisição penalizada.

Para executar os cálculos, o monitor de rendimento precisa de informações atualizadas sobre os recursos que foram consumidos até então, que podem ser obtidas diretamente do monitor de máquinas virtuais. Além do que foi gasto, também são necessárias informações sobre a receita que podem ser obtidas pela quantidade de requisições bem e mal atendidas que vem do monitor de requisições. Qualquer outro fator que possa influenciar o rendimento do deve estar representado na função lucro.

Definir se uma requisição deve ou não ser penalizada é uma tarefa simples, por isso, a definição de como uma requisição é penalizada deve ser bem clara para que o rendimento seja calculado corretamente. Exemplos de condições que levam a penalização de requisições são: se há atraso no atendimento da requisição, o quanto além do contratado com que uma requisição é atendida e a satisfação do usuário.

O monitor de rendimento é essencial para o viés de mercado da arquitetura, deve calcular e manter disponíveis informações atualizadas sobre as variáveis econômicas do sistema que são consequência do gerenciamento adequado de desempenho.

4.1.5 Despachador

A cada chegada de uma nova requisição ao sistema, é necessário escolher entre as instâncias de máquina virtual disponíveis aquela que será responsável por processá-la. O despachador atua como um escalonador global de requisições no conjunto de capacidades disponíveis. Algoritmos clássicos de escalonamento (*e. g. Round-robin*) podem ser utilizados para distribuir as requisições nesse módulo, no entanto, a aplicação de soluções que considerem o balanceamento da carga podem trazer melhores resultados.

Para executar corretamente suas funções, o despachador precisa ter conhecimento de quantas e quais são as máquinas virtuais disponíveis e, dependendo do algoritmo utilizado, informações sobre o desempenho das mesmas. Essas informações são obtidas junto ao monitor de máquinas virtuais.

A implementação real de um bom despachador deve garantir não só o escalonamento de todas as requisições como o balanceamento da carga de trabalho entre as instâncias disponíveis, mantendo um equilíbrio entre a quantidade de requisições submetidas a cada servidor. Não havendo balanceamento, algumas instâncias podem ficar sobrecarregadas, aumentando o tempo médio de resposta sem que a taxa de utilização seja prejudicada.

Caso a política de escalonamento utilizada permita a redistribuição das requisições que ainda não foram processadas entre as instâncias, o despachador deve implementar mecanismos que permitam reenviar essas requisições para outras máquinas virtuais.

4.1.6 Controlador e provisor de recursos

Sem dúvida, de todos os módulos presentes na arquitetura, o controlador e provisor de recursos é o mais importante e o mais complexo de ser projetado e implementado. Implementar cada um dos módulos do *Broker* envolve uma série de dificuldades e escolhas de projeto que vão de como o rendimento será calculado aos parâmetros que compõem os contratos. Cada módulo deve ser corretamente planejado e implementado até que todos estejam prontos para trabalhar em conjunto.

O objetivo principal do *Broker* é tornar o sistema alvo auto-adaptativo para que, uma vez projetado, o mesmo tenha a propriedade de adaptar-se sempre que houver uma variação de carga ou alteração de desempenho do sistema por qualquer motivo, submetendo o sistema a situações transientes. Enquanto os outros módulos mantêm basicamente informações estatísticas sobre o sistema, o controlador e provisor de recursos é responsável pela adaptação da quantidade de recursos segundo as informações de desempenho disponíveis nos outros módulos. A figura 4.3 ilustra os módulos da arquitetura que interagem diretamente com a planta.

Basicamente, os monitores colhem informações de desempenho do sistema objetivo e sintetizam da forma mais adequada para que sejam usadas a posteriori na tomada de decisão do *Broker*. Já o Despachador escolhe e envia, segundo a política de balanceamento de carga, para uma das

Figura 4.3 — Fluxo da interação entre módulos da arquitetura e *Data center*.

instâncias disponíveis as novas requisições que chegam ao sistema. A cada intervalo de amostra o Controlador e provisor de recursos analisa as informações do comportamento do sistema para a carga atual e decide se é necessária uma modificação na quantidade de recursos.

Este módulo utiliza a *API* fornecida pelo provedor para gerenciar a quantidade de recursos que atenderá a demanda. A cada intervalo de amostra, o controlador e provisor de recursos analisa a resposta do sistema para a variável de desempenho de interesse e reage adequadamente. Como a arquitetura trata de sistemas computacionais que são naturalmente discretos, é importante que o intervalo de amostra escolhido seja representativo para o sistema que se quer controlar. Isso significa que o tamanho do intervalo de amostra deve ser suficientemente grande quanto o necessário para capturar a real dinâmica do sistema.

São muitas as formas possíveis de implementar o módulo. Este trabalho propõe que o controlador e provisor de recursos seja implementado como um sistema de controle de malha fechada como ilustrado na figura 3.4.

Constantemente, o módulo verifica se as capacidades computacionais atuais são suficientes para manter os objetivos de desempenho dentro dos limites aceitáveis. Essa verificação é realizada medindo-se a resposta do sistema objetivo para a entrada de controle atual. Se a quantidade de máquinas virtuais for insuficiente, o módulo deve identificar a necessidade e corrigi-la imediatamente.

Para executar sua função, o controlador e provisor de recursos solicita aos monitores de máquinas virtuais de requisições o valor atual da resposta do sistema para as variáveis de desempenho de interesse. Os monitores são responsáveis por preparar os valores da maneira mais adequada para o controlador, o que envolve a suavização das saídas ou mesmo um atraso proposital dos valores

em intervalos de amostra. As informações obtidas são comparadas com aquelas dos contratos e, se necessário, uma nova quantidade de recursos é provida.

A arquitetura pressupõe a provisão de recursos de granularidade grossa (*i. e.* máquinas virtuais), o que não é necessariamente obrigatório, mas conveniente aos módulos previstos. Para prover os recursos, uma nova entrada de controle que representa um déficit ou superávit de recursos é enviado ao *data center* para que a adaptação a nova realidade experimentada seja imediata.

A implementação do controlador do módulo pode ser realizada seguindo as etapas definidas em 3.6. Tendo o controlador definido, todos os demais módulos devem ser implementados tendo como foco este módulo.

4.2 Projeto de sistema de controle proporcional-integral de terceira ordem com suavização de resposta por média móvel

Esta seção explica um método desenvolvido neste trabalho que facilita o desenvolvimento de sistemas de controle de malha fechada de terceira ordem com suavização de resposta do sistema por média móvel para um sistema estável. O método a seguir reduz todas as funções de transferência a um sistema de equações lineares que relaciona os parâmetros dos polos da malha fechada, do controlador e o transdutor.

O método contempla um sistema de malha fechada como o mostrado na figura 4.4 onde estão presentes um controlador com a política *PI* e um transdutor que é uma média móvel. Por conveniência, todos os sistemas envolvidos são discretos e consideram o mesmo intervalo de amostra.

Figura 4.4 — Sistema de controle *PI* de malha fechada com média móvel.

O método considera que o sistema objetivo G é modelado por uma equação de diferença de primeira ordem, ou pode ser aproximado para uma equação desse tipo, como mostrado na equação 4.2. Tendo a representação do sistema G no domínio do tempo, a função de transferência

do sistema pode ser obtida transformando os sinais envolvidos para o domínio Z e executando manipulações simples, como mostra a equação 4.3.

$$y(k+1) = ay(k) + bu(k) \quad (4.2)$$

$$G(z) = \frac{Y(z)}{U(z) + D(z)} = \frac{b}{z - a} \quad (4.3)$$

Os parâmetros a e b definem a dinâmica do sistema objetivo e devem ter sido estimados na modelagem do sistema, sendo a o polo de malha aberta para o sistema G .

O bloco controlador, ora denominado C , contempla como política um controlador proporcional-integral, cujo modelo no domínio do tempo é obtido pela soma das influências de cada ganho no erro, como mostra a equação 4.4. Da mesma forma que na equação 4.3, a função de transferência $C(z)$ que representa a transformação do erro em entrada de controle no domínio Z é dada pela equação 4.5.

$$\begin{aligned} u_p(k) &= K_p e(k) \\ u_i(k) &= u_i e(k-1) + K_i e(k) \\ u(k) &= u_p(k) + u_i(k) \end{aligned}$$

$$u(k) = \overbrace{K_p e(k)}^{u_p(k)} + \underbrace{u_i e(k-1) + K_i e(k)}_{u_i(k)} \quad (4.4)$$

$$C(z) = \frac{U(z)}{E(z)} = K_i \frac{z}{z-1} + K_p \quad (4.5)$$

O bloco controlador também contribui com um polo de malha aberta $z = 1$.

Assim como o sistema objetivo, o transdutor H , que transforma a saída medida em uma média móvel, pode ser representado por uma equação de diferença que considera a influência das saídas anteriores na média atual, como mostra a equação 4.6. Aplicando a transformada Z para os sinais da equação de diferença 4.6 obtém-se a função de transferência do transdutor dada pela equação 4.7.

$$w(k+1) = cw(k) + (1-c)y(k) \quad (4.6)$$

$$H(z) = \frac{W(z)}{T(z)} = \frac{1-c}{z-c} \quad (4.7)$$

O transdutor também contribui ao sistema com um polo de malha aberta $z = c$, fazendo com que o sistema de malha fechada seja de terceira ordem.

Considerando todas as entradas nulas, exceto a referência $R(z)$, para o sistema exposto na figura 4.4, e combinando as funções de transferência dos sistemas $C(z)$, $G(z)$ e $H(z)$ como mostrado na equação 3.46, a função de transferência do sistema de controle de malha fechada $F_r(z)$ é dada pela equação 4.8.

$$F_r(z) = \frac{T(z)}{R(z)} = \frac{C(z)G(z)}{1 + C(z)G(z)H(z)}$$

$$F_r(z) = \frac{\overbrace{\left(\frac{[(K_i + K_p)z - K_p]}{(z - 1)}\right)}^{C(z)} \overbrace{\left(\frac{b}{(z - a)}\right)}^{G(z)}}{1 + \underbrace{\left(\frac{[(K_i + K_p)z - K_p]}{(z - 1)}\right)}_{C(z)} \underbrace{\left(\frac{b}{(z - a)}\right)}_{G(z)} \underbrace{\left(\frac{(1 - c)}{(z - c)}\right)}_{H(z)}}$$

$$F_r(z) = \frac{(K_i + K_p)bz^2 - (K_i bc + K_p bc - K_p b)z + K_p bc}{z^3 - (1 + a + c)z^2 + [a + c + ac - K_i b(1 - c) + K_p b(1 - c)]z - (ac + K_p b + K_p bc)} \quad (4.8)$$

Como o sistema considerado aqui é *LTI*, caso as entradas $D(z)$ e $N(z)$ não sejam nulas, é viável combinar as funções de transferência para essas entradas utilizando a propriedade de superposição.

A função de transferência do sistema modelado $F_r(z)$ pode colocada em termos de um numerador $N_m(z)$, como mostra a equação 4.9, e um denominador $D_m(z)$, como mostra a equação 4.10, que representa seu polinômio característico.

$$F_r(z) = \frac{N_m(z)}{D_m(z)}$$

$$N_m(z) = (K_i + K_p)bz^2 - (K_i bc + K_p bc - K_p b)z + K_p bc \quad (4.9)$$

$$D_m(z) = z^3 - (1 + a + c)z^2 + [a + c + ac - K_i b(1 - c) + K_p b(1 - c)]z - (ac + K_p b + K_p bc) \quad (4.10)$$

O polinômio característico $D_m(z)$ é obtido com base nos parâmetros a e b do sistema objetivo, K_i e K_p que são os ganhos do controlador e c que a influência dos saídas anteriores na média

do transdutor. Como a e b são constantes conhecidas pelo modelo de primeira ordem do sistema objetivo, apenas os parâmetros K_i , K_p e c são desconhecidos no polinômio característico.

Tendo que o polinômio característico do sistema de controle de malha fechada modelado $D_m(z)$ foi obtido, assim como no método de colocação dos polos, as propriedades de interesse, ou seja, o comportamento que se espera que o sistema apresente, podem ser utilizadas para obter polos de malha fechada e, conseqüentemente, um polinômio característico que represente o comportamento desejado $D_d(z)$ seja alcançado.

Cada um dos sistemas envolvidos (*i. e.* controlador, planta e transdutor) é de primeira ordem, contribuindo com um polo na função de transferência do sistema de controle de malha fechada. Como são três os envolvidos, são três polos que resultam em um sistema de terceira ordem. Os passos seguintes consideram o método de colocação de polos para ajustar os polos do sistema de malha fechada com o comportamento do controlador que se deseja.

Considerando que dois dos três polos do sistema de malha fechada, ora chamados de p_1 e p_2 , constituem um par conjugado de polos complexos e que esses polos apresentam comportamento dominante, e p_3 como um polo de livre escolha que dá um grau de liberdade, é possível encontrar as coordenadas polares desse par adequando-os ao tempo de ajuste e amplitude máxima desejados. Como considera-se a premissa de que o sistema é estável, todos os três polos do sistema têm magnitude menor que 1.

$$\begin{cases} |p_1| < 1 \\ |p_2| < 1 \\ |p_3| < 1 \end{cases}$$

Para os polos complexos p_1 e p_2 , duas representações são especialmente úteis para o método aqui demonstrado, tomando a representação exponencial dada pelas coordenadas polares r (magnitude) e θ (ângulo) e sua representação dada pelas coordenadas cartesianas α (real) e β (imaginária), como mostram as equações 4.11 e 4.12.

$$p_1 = \underbrace{r}_{\text{Magnitude}} e^{j \underbrace{\theta}_{\text{Ângulo}}} = \underbrace{\alpha}_{\text{Real}} + \underbrace{\beta j}_{\text{Imaginário}} \quad (4.11)$$

$$p_2 = r e^{-j\theta} = \alpha - \beta j \quad (4.12)$$

As representações dos polos complexos são equivalentes e usando manipulações trigonométricas simples é possível encontrar relações entre as coordenadas polares e cartesianas como mostram as equações de 4.13 a 4.16.

$$r = \sqrt{\alpha^2 + \beta^2} \quad (4.13)$$

$$\theta = \tan^{-1} \frac{\beta}{\alpha} \quad (4.14)$$

$$\alpha = r \cos \theta \quad (4.15)$$

$$\beta = r \sin \theta \quad (4.16)$$

Como exposto na seção 3.2, para um sistema de controle de malha fechada exposto em regime transiente é interessante que o tempo de ajuste do controlador esteja dentro de um limite aceitável. Logo, cabe a quem projeta o controlador escolher um tempo de ajuste que seja adequado para os objetivos de controle propostos. Sabendo que o tempo de ajuste K_s depende sempre da magnitude do maior polo e pode ser obtido pela equação 3.34, dado o tempo de ajuste desejado é possível encontrar a magnitude dos polos p_1 e p_2 pela equação 4.17.

$$|p_1| = |p_2| = r = e^{\left(\frac{-4}{K_s}\right)} \quad (4.17)$$

Na seção 3.5 é definido que a amplitude máxima de um sistema de malha fechada que tem um polo dominante pode ser obtida pela equação 3.36, que relaciona o ângulo, a magnitude e amplitude máxima. Tendo encontrado o valor da magnitude e sabendo que, assim como o tempo de ajuste, a amplitude máxima também é um comportamento definido no projeto do controlador, para o polo complexo, a coordenada polar θ do polo dominante é dada pela equação 4.18.

$$\theta = \pi \frac{\log r}{\log M_p} \quad (4.18)$$

Visto que os valores das propriedades de interesse permitiram estimar os valores das coordenadas polares ideais, é possível obter a representação exponencial dos polos p_1 e p_2 . Das coordenadas polares também é possível encontrar a representação cartesiana de ambos pelas equações 4.15 e 4.16. A representação cartesiana é mais simples de ser manipulada algebricamente já que não envolve funções trigonométricas, todavia, o método aqui desenvolvido apenas sugere a representação cartesiana, mas não a obriga.

O terceiro polo p_3 dá um grau de liberdade para o controlador e é tratado com uma incógnita até que os outros parâmetros do sistema sejam estimados. A única consideração necessária sobre p_3 é que a sua magnitude deve ser menor que a magnitude de p_1 e p_2 , o que necessariamente também faz com que p_3 não altere a estabilidade do sistema. Essa condição garante que p_3 não é dominante e seu comportamento não é tão decisivo no objetivo e nas propriedades de interesse do sistema de controle.

Uma vez que dois polos foram estimados e o terceiro é uma incógnita, o polinômio característico que apresenta o comportamento desejado para as propriedades escolhidas $D_d(z)$ é dado pela

equação 4.19. Como α e β são coordenadas do polos p_1 e p_2 , apenas o polo p_3 é desconhecido em $D_d(z)$.

$$F_r(z) = \frac{N_d(z)}{D_d(z)}$$

$$D_d(z) = (z - p_1)(z - p_2)(z - p_3)$$

$$D_d(z) = [z - (\alpha + \beta j)][z - (\alpha - \beta j)](z - p_3)$$

$$D_d(z) = z^3 - (2\alpha + p_3)z^2 + (\alpha^2 + 2\alpha p_3 + \beta^2)z - (\alpha^2 + \beta^2)p_3 \quad (4.19)$$

Tendo que os polinômios característicos modelado, representado pela equação 4.10, e o polinômio característico desejado, representado pela equação 4.19, representam o mesmo sistema de malha fechada, é possível compará-los, como mostra a equação a equação 4.20, de onde é possível obter o sistema de equações lineares 4.21.

$$D_m(z) = D_d(z)$$

$$\overbrace{z^3 - (1 + a + c)z^2 + [a + c + ac + K_i b(1 - c) + K_p b(1 - c)]z - (ac + K_p b - K_p bc)}^{\text{Polinômio modelado}} = \underbrace{z^3 - (2\alpha + p_3)z^2 + (\alpha^2 + 2\alpha p_3 + \beta^2)z - (\alpha^2 + \beta^2)p_3}_{\text{Polinômio desejado}} \quad (4.20)$$

$$\begin{cases} 2\alpha + p_3 = 1 + a + c \\ \alpha^2 + 2\alpha p_3 + \beta^2 = c + a(1 + c) + K_i b(1 - c) + K_p b(1 - c) \\ (\alpha^2 + \beta^2)p_3 = ac + K_p b - K_p bc \end{cases} \quad (4.21)$$

Pela primeira equação do sistema de equações lineares 4.21 percebe-se que o terceiro polo do sistema da malha fechada p_3 depende apenas do valor do parâmetro c escolhido para a média móvel do transdutor. Logo, p_3 pode ser obtido de acordo com a influência desejada das respostas anteriores na resposta atual do sistema, desde que isso não faça de p_3 dominante. Assim, é possível obter o intervalo que limita os possíveis valores do parâmetro de média móvel c baseado nos polos p_1 e p_2 e no parâmetro a do sistema objetivo, como mostra a inequação 4.22, para que o sistema apresente continue apresentando as propriedades definidas em projeto.

$$\begin{aligned}
|p_3| &\leq |p_1| \\
|1 + a + c - 2\alpha| &\leq |p_1| \\
-|p_1| &\leq 1 + a + c - 2\alpha \leq |p_1| \\
2\alpha - 1 - a - |p_1| &\leq c \leq |p_1| + 2\alpha - 1 - a
\end{aligned}$$

$$\mathbf{c} \in [2\alpha - |p_1| - a - 1, 2\alpha + |p_1| - a - 1] \quad (4.22)$$

É importante notar que, por definição, o parâmetro c não pode assumir valores negativos ou maiores que 1 e, uma vez que a função de transferência do componente de média móvel, dada pela equação 4.7, tem um zero em $c = 1$, o parâmetro também não pode ser igual a 1. Logo, os possíveis valores desse parâmetro devem pertencer ao intervalo 4.23.

$$\mathbf{c} \in [2\alpha - |p_1| - a - 1, 2\alpha + |p_1| - a - 1] \cap [0, 1[\quad (4.23)$$

Tendo escolhido um valor adequado para c dentro do intervalo e encontrado um valor para p_3 correspondente, os parâmetros desconhecidos do controlador K_i e K_p são obtidos resolvendo as demais equações do sistema linear.

Alternativamente, $D_d(z)$ pode ser descrito usando a representação exponencial de p_1 e p_2 com coordenadas polares. A equação 4.24 mostra o polinômio característico desejado representado pelas coordenadas polares.

$$D_d(z) = z^3 - (2r \cos \theta + p_3)z^2 + (r^2 + 2r \cos \theta p_3)z - r^2 p_3 \quad (4.24)$$

A equação 4.25 dá a comparação entre os polinômios usando as coordenadas polares.

$$\begin{aligned}
z^3 - (1 + a + c)z^2 + [a + c + ac + K_i b(1 - c) + K_p b(1 - c)]z - (ac + K_p b - K_p bc) = \\
z^3 - (2r \cos \theta + p_3)z^2 + (r^2 + 2r \cos \theta p_3)z - r^2 p_3
\end{aligned} \quad (4.25)$$

Para a representação exponencial dos polos, o sistema 4.26 mostra as equações lineares obtidas.

$$\begin{cases} 2r \cos \theta + p_3 = 1 + a + c \\ r^2 + 2r \cos \theta p_3 = c + a(1 + c) + K_i b(1 - c) + K_p b(1 - c) \\ r^2 p_3 = ac + K_p b - K_p bc \end{cases} \quad (4.26)$$

Da mesma forma como foi obtido para as coordenadas cartesianas, é possível encontrar o intervalo ideal para o parâmetro da média móvel c em função das coordenadas polares, como mostra a inequação 4.27, que da mesma forma resulta no intervalo 4.28 de possíveis valores do parâmetro.

$$\begin{aligned} |p_3| &\leq |p_1| \\ |1 + a + c - 2r \cos \theta| &\leq |p_1| \\ -|p_1| &\leq 1 + a + c - 2r \cos \theta \leq |p_1| \\ 2r \cos \theta - 1 - a - |p_1| &\leq c \leq |p_1| + 2r \cos \theta - 1 - a \\ 2r \cos \theta - r - a - 1 &\leq c \leq 2r \cos \theta + r - a - 1 \end{aligned}$$

$$c \in [2r \cos \theta - r - a - 1, 2r \cos \theta + r - a - 1] \quad (4.27)$$

$$c \in [2r \cos \theta - r - a - 1, 2r \cos \theta + r - a - 1] \cap [0, 1] \quad (4.28)$$

O método também permite verificar se as características comportamentais desejadas em regime transiente são alcançáveis. Por exemplo, sabendo que a magnitude do polo dominante é dada pela equação 4.17 e que para garantir a estabilidade do sistema de malha fechada o polo dominante deve ter magnitude menor que 1, verifica-se facilmente que não é possível que o sistema tenha um tempo de assentamento nulo, como pode ser comprovado pela equação 4.29, uma vez que isso faria com que a magnitude do polo dominante fosse muito grande e faria com que o sistema ficasse instável. Tendo em vista que K_s é uma medida de tempo e, por isso, não assume valores não positivos, não é necessário calcular o limite quando K_s tende a zero pela esquerda.

$$\lim_{K_s \rightarrow 0^+} e^{-4/K_s} = \infty \quad (4.29)$$

O encolhimento do valor absoluto do polo dominante tem como consequência uma limitação na magnitude do polo p_3 e, em consequência, no intervalo para o parâmetro de média móvel c . É possível que, em casos extremos, o intervalo calculado para o parâmetro c seja irrealizável, isto é,

um intervalo que contenha somente valores negativos ou maiores ou iguais a 1, resultando em um intervalo vazio que indica que as escolhas de projeto escolhidas são impraticáveis. Um exemplo dessa conclusão é dado na seção 5.4.2.

Experimentos

Com o intuito de demonstrar a validade da arquitetura de *broker* de consumidor proposta foram planejados e executados uma série de experimentos. Este capítulo começa com as considerações acerca do tempo de inicialização de máquinas virtuais num ambiente de recursos físicos compartilhados para um ambiente simulado.

A seguir um conjunto de experimentos considera que considera a alternância de um sistema entre períodos estacionários e transitório chama a atenção para a importância da avaliação de desempenho em regime transiente e como a correta atuação nesses períodos podem ser determinantes para que os objetivos de desempenho de um sistema sejam alcançados.

Este capítulo também detalha a implementação de um protótipo da arquitetura proposta para um ambiente de simulação usando a biblioteca CloudSim, além de fazer algumas considerações sobre a biblioteca que levaram a implementação de uma extensão da mesma para que fosse possível o desenvolvimento do protótipo. O capítulo também traz uma descrição pormenorizada do planejamento de experimentos e enumera as configurações do ambiente de simulação consideradas na experimentação. Foram conduzidos experimentos com quatro variações do módulo principal da arquitetura, cujo desenvolvimento considerou o método desenvolvido neste trabalho.

5.1 Tempo de inicialização de máquinas virtuais

O tempo de inicialização de uma instância de máquina virtual é atributo do *data center* e das tecnologias empregadas para viabilizar a virtualização de sistemas sobre as capacidades computacionais disponíveis e está além da autoridade do consumidor. Entretanto, levando em conta a

aplicação de tecnologias atuais em sistemas reais nos quais as instâncias de máquinas virtuais têm tempos de inicialização equivalentes ao de uma máquina física, o tempo de inicialização pode influenciar diretamente o gerenciamento de desempenho e a forma como o sistema objetivo responde a uma modificação da configuração frente a uma alteração da demanda.

O intervalo de tempo compreendido entre a ordem do gerenciador de desempenho para que uma instância seja iniciada e o instante no qual a máquina virtual está disponível para processar carga de trabalho pode gerar um atraso entre o desempenho que se espera do sistema e o desempenho percebido de fato. Além do que, sob diferentes condições de carga do servidor físico, o tempo de inicialização de uma nova instância de máquina virtual pode variar. Num sistema ideal, as instâncias de máquinas virtuais estariam disponíveis tão logo fossem necessárias, mas não é o que se observa em *data centers* reais. Uma solução praticável seria manter um conjunto de instâncias previamente inicializadas e prontas para receber carga de trabalho caso necessário, todavia, essa solução não é oportuna se considerar que a demanda pode variar imprevisivelmente, dificultando a determinação de uma quantidade precisa de instâncias previamente ligadas, e que, mesmo ociosas, essas máquinas são cobradas por terem recursos físicos alocados.

Havendo variação intensa da demanda, adaptações como a proposta na seção 4.1.1, que gerencia o ciclo de vida das instâncias de máquinas virtuais com o intuito de diminuir os efeitos do tempo de inicialização, são factíveis e podem diminuir a influência do tempo de inicialização no gerenciamento do desempenho.

Para entender como os tempos de inicialização de máquinas virtuais é influenciado pelas condições de carga do servidor físico e como esses tempos se distribuem foram realizados experimentos levando em conta dois fatores: **a carga do servidor e a quantidade de instâncias que se deseja ligar**. Foram definidos dois níveis para a quantidade de instâncias que serão ligadas: **6 instâncias e 9 instâncias**. Essas máquinas virtuais foram ligadas todas ao mesmo tempo e o intervalo de tempo entre a ordem de inicialização até que a máquina virtual estivesse pronta e disponível foi medido e armazenado pelo hipervisor. Já a carga do servidor foi dividida em 8 níveis que representam a quantidade de instâncias com carga de trabalho que já estão em execução no servidor quando foi dada a ordem de ligamento para as demais. Esse níveis são: **2, 4, 6, 8, 10, 12, 14 e 16** instâncias previamente prontas.

Os experimentos foram executados segundo um arranjo fatorial completo e cada experimento foi repetido por 6 vezes, resultando num total de **96 execuções** e um tempo total (*i. e.* do planejamento de experimentos, configuração do ambientes, execução dos experimentos e análise dos resultados) em torno de 10 dias.

Para carregar o servidor, cada uma das instâncias previamente ligadas executava uma aplicação de teste **Smallpt**¹, do tipo *CPU-bound*, disponível no pacote de *benchmark Phoronix Test Suite*². O objetivo era que as instâncias que já estavam em execução pudessem gerar uma carga sintética no servidor físico e no hipervisor, ocupando todo tempo de processador que fosse destinado a cada

¹<http://openbenchmarking.org/test/pts/smallpt>

²<http://www.phoronix-test-suite.com/>

uma dessas instâncias, de modo que a inicialização de novas máquinas virtuais não acontecesse com as capacidades computacionais em condições ociosas. Uma vez inicializada e disponível a nova instância também passava a executar a mesma aplicação de teste. Isso fazia com que as novas máquinas também gerassem carga sobre o servidor, exigindo a utilização de outros recursos que não só a ocupação de memória.

O ambiente de experimentação foi composto de um servidor **Dell PowerEdge T410**³, com processador **Intel Xeon X5660**⁴, dotado de **6 núcleos físicos de 2.8 GHz** e **12 MB de cache por núcleo**. Além disso, o servidor dispunha de **12 GB de memória principal** e **1.5 TB de armazenamento** em disco rígido. O sistema operacional instalado no servidor dos experimentos foi um **GNU/Linux**, com a versão do **kernel 3.2.0-31** e **sistema de arquivos ext4**. Para dar suporte a virtualização de sistemas computacionais, o servidor ainda dispunha da biblioteca de virtualização **libvirt 0.9.8-2**⁵ e emulador de processadores **QEMU-KVM 1.0**⁶.

Tendo executado todos os experimentos e colhido os intervalos de inicialização, foi possível calcular as medidas de localização e dispersão de interesse e, com base nessas informações, inferir sobre a influência desses tempos no desempenho do sistema. A Figura 5.1 apresenta o tempo médio de inicialização para 6 e 9 instâncias sob diferentes condições de carga do servidor, bem como o intervalo de confiança de 95%.

Figura 5.1 — Tempo de inicialização para diferentes condições de carga do servidor

Quando 6 instâncias foram ligadas, é possível observar que o tempo de inicialização se manteve estacionário com uma média praticamente constante independente da quantidade de máquinas

³<http://www.dell.com/br/empresa/p/poweredge-t410/pd>

⁴http://ark.intel.com/products/47921/Intel-Xeon-Processor-X5660-12M-Cache-2_80-GHz-6_40-GTs-Intel-QPI

⁵<http://libvirt.org/>

⁶http://www.linux-kvm.org/page/Main_Page

virtuais que já eram executadas pelo hipervisor. Para esse caso, a quantidade total de máquinas virtuais no servidor físico variou entre 8 e 22 instâncias após todas estarem disponíveis. Verifica-se também que os intervalos de confiança foram pequenos, o que demonstra que os tempos de inicialização variaram em torno de uma média com pouca dispersão.

Para os experimentos em que foram ligadas 9 instâncias, verifica-se que houve um aumento no tempo médio de inicialização e que num primeiro momento (*i. e.* entre um total de 11 e 19 instâncias) essa média teve comportamento estacionário. Embora, quando a quantidade de máquinas virtuais ligadas vai além de um total de 23 instâncias, há um aumento do tempo médio causado pela sobrecarga gerada pela quantidade de máquinas virtuais ligadas ao mesmo tempo. Novamente, mesmo com um aumento crescente significativo para uma grande quantidade de máquinas virtuais, os intervalos de confiança permaneceram pequenos.

É possível observar também que o tempo médio de inicialização para 9 instâncias é maior que o tempo médio para 6 instâncias, independente da quantidade de máquinas que já estavam ligadas *a priori*, exceto para o caso de 2 instâncias iniciais. Isso mostra que a quantidade de máquinas em contenção pela inicialização influencia mais o tempo médio que a quantidade de recursos que já estão sendo ocupados. Além disso, a carga do servidor imposta por outras máquinas começa a fazer maior diferença no tempo de inicialização somente quando o número de máquinas ligadas e ligando é muito grande e provoca a falta de recursos físicos para novas instâncias.

Uma vez que os contratos de prestação de serviço estabelecem parâmetros de desempenho e para a qualidade do serviço prestado pelos provedores aos seus consumidores, não se espera que situações de sobrecarga aconteçam com frequência e conjectura-se que o provedor será capaz de gerenciar a distribuição das instâncias entre os servidores físicos de seu *data center* com capacidades suficientes para suportá-las.

Exceto para situações de sobrecarga, o que se observa é um estacionamento do tempo médio de inicialização e dessa forma, é plausível que, em ambientes simulados onde há a variação da quantidade de máquinas virtuais, a aproximação do tempo de inicialização para um valor constante ou distribuído em torno de uma média seja bastante representativo e consinta com o que intercorre em sistemas reais. Este trabalho considera que os tempos de inicialização de máquinas virtuais são constantes e que, após uma quantidade determinística de intervalos de simulação, as instâncias já estarão inicializadas e disponíveis.

5.2 Configuração do *data center* e das máquinas virtuais utilizados na simulação

É possível que na falta de recursos para atender seus consumidores, um provedor de nuvem adquira infraestrutura junto a outro provedor para expandir os recursos do seu *data center* temporariamente. Dessa forma, é possível que instâncias de máquinas virtuais de uma mesma aplicação e

de um mesmo consumidor sejam espalhadas entre diferentes *data centers*. A esse tipo de interação entre *data centers* dá-se o nome de **nuvens confederadas**⁷. A distribuição das máquinas virtuais seja entre *data centers* ou entre os servidores físicos de um mesmo provedor está fora do escopo da arquitetura proposta que parte do princípio que os problemas de alocação das máquinas virtuais são resolvidos a contento por outras soluções adotadas pelo provedor de nuvem e que as *APIs* disponíveis aos consumidores não suportam a localização de uma instância virtual, que envolveria o acesso a informações internas do *data center* e teria como consequência a possibilidade de um *broker* de consumidor influenciar diretamente no desempenho da aplicação de outro consumidor.

Para os experimentos realizados neste trabalho, considerou-se que o *data center* utilizado contém recursos suficientes e capazes de prover sempre a quantidade de recursos requisitada pelo *broker* do consumidor. Para todas as execuções, o *data center* foi composto de **200 servidores** físicos independentes, sendo que a configuração física de cada unidade de servidor dispunha de **2 entidades de processamento** ou núcleos de arquitetura **x86**, com capacidade de processar **1000 Millions of Instructions Per Second (MIPS)**. Cada servidor dispunha de **4096 unidades de memória primária** e **1000000 unidades de armazenamento**. Além disso, essas máquinas estão conectadas em rede com uma taxa de transmissão de **10000 unidades de transmissão**. O sistema operacional instalado em cada um dos servidores era **GNU/Linux** e o gerenciador de máquinas virtuais era o **Xen**⁸.

A política de alocação de máquinas virtuais adotadas foi **Space-shared**, assim, uma vez que máquina virtual ocupava um núcleo, ali permanecia até que fosse desligada, dedicando a infraestrutura física envolvida em uma execução a apenas uma máquina virtual. Da mesma forma como as máquinas físicas, foram consideradas máquinas virtuais homogêneas. Em todos os experimentos foram ligadas **20 máquinas virtuais** inicialmente, sendo que o número mínimo de instâncias ligadas não deveria ser menor que 1 máquina virtual durante toda a simulação, enquanto que o número máximo de instâncias era limitado pelas capacidades dos recursos físicos disponíveis no *data center*. Cada máquina virtual dispunha de **1 unidade de processamento** de arquitetura **x86** com capacidade para processar **1000 MIPS**, consumindo **2048 unidades memória primária**. Uma vez ligadas as máquinas virtuais eram conectadas à rede a **1000 unidades de transferência**. Além disso, quando desligadas, a imagem de cada instância de máquina virtual ocupava **10000 unidades de armazenamento**. Para alocar as cargas de trabalho, cada máquina virtual utilizada a política **Space-shared**, isto é, uma vez que uma requisição começava a ser processada, não havia possibilidade de preempção.

Apesar de permitir, a princípio não foram fixados preços e moeda relativos a cobranças pela utilização dos recursos na arquitetura de *broker*. É possível ter uma dimensão do rendimento obtido de acordo com a quantidade de recursos utilizados para atingir os objetivos de desempenho e pela quantidade de requisições que não foram bem atendidas.

⁷tradução do termo em inglês *federated clouds*.

⁸<http://www.xen.org/>

5.3 Avaliação de desempenho em regime transiente

Este conjunto de experimentos tem por objetivo mostrar a importância da avaliação de desempenho quando um sistema se encontra no regime transiente e como a implementação de soluções adequadas para adaptação dinâmica dos recursos disponíveis podem influenciar diretamente os objetivos de desempenho e, conseqüentemente, o rendimento atribuídos ao mesmo.

É muito comum que o comportamento verificado para um sistema quando no regime transiente seja desprezado e considerado irrelevante na avaliação do desempenho, principalmente quando essa avaliação se concentra em analisar apenas o comportamento do sistema quando sujeito a uma solução para o regime estacionário, o que nem sempre condiz com efetivamente com o comportamento de um sistema real ao longo de seu ciclo de vida.

Na análise de regime estacionário, o início de uma experimentação, chamado de tempo de aquecimento⁹, frequentemente é descartado por não representar o comportamento habitual que se deseja avaliar. No entanto, o *warm-up* nada mais é do que um período em que o sistema se encontra em regime transiente antes de convergir para um valor de interesse em regime estacionário, que é o foco dessa análise.

Há que se considerar que, ao longo do seu ciclo de vida, um sistema está sujeito a alternar entre situações estacionárias e transitórias por muitas vezes e que, por isso, ambas devem ser ponderadas para que os objetivos de desempenho sejam alcançados. Além do que, ignorar os períodos transientes significa ignorar que um sistema pode apresentar comportamentos que não observados e não são capturados pelas métricas estacionárias e, esses comportamentos podem levar o sistema a situações que, se não forem previstas e devidamente planejadas, excedem suas capacidades. Sendo assim, medidas de interesse de regime transientes, como a amplitude máxima atingida pela resposta do sistema e o tempo de assentamento ou ajuste até que um novo regime estacionário seja atingido, são interessantes para que se possa medir e analisar o comportamento de um sistema durante os períodos transitórios.

Para mostrar a importância que esses períodos têm sobre os objetivos de desempenho de um sistema foram realizados diversos experimentos com três variações de um mesmo controlador integral, para uma mesma configuração de ambiente, em que os gerenciadores de desempenho com esses controladores foram submetidos às mesmas condições de carga. A variável de controle escolhida e utilizada em todos os experimentos foi a taxa de utilização do sistema controlado, baseada na ocupação das instâncias de máquinas virtuais disponíveis. Conforme as requisições chegavam ao sistema, as mesmas eram enviadas seguindo a ordem crescente de ocupação das máquinas virtuais, onde eram escalonadas por ordem de chegada.

O primeiro controlador experimentado, ora denominado **controlador 1**, é um controlador integral convencional sem modificações cuja entrada de referência (*i. e.* a taxa de utilização do sistema) foi fixada em 70%; o segundo controlador, ora denominado **controlador 2**, possui a mesma

⁹É muito comum que o termo em inglês *warm-up* seja utilizado para designar esse período.

entrada de referência do primeiro e estabelece um intervalo de desempenho aceitável com limite mínimo de 60% e máximo de 80%, atuando somente quando a saída do sistema encontra-se fora do intervalo; o terceiro controlador, ora denominado **controlador 3**, possui a mesma entrada de referência e os mesmos limites mínimo e máximo do segundo controlador, mas atua obedecendo a uma histerese até a entrada de referência (*i. e.* uma vez que a resposta do sistema esteja acima do limite superior, o controlador passa a atuar até que o erro medido seja menor ou igual a 0, da mesma forma, caso a taxa de utilização do sistema seja menor que o limite inferior, o controlador passa a atuar até que a taxa de utilização seja maior ou igual à referência).

O intuito desses experimentos é analisar o comportamento dos diferentes controladores quando o sistema objetivo for sujeito às seguintes situações transitórias: **warm-up**, regime transiente que começa no instante inicial da simulação e cuja duração depende da atuação do controlador; **perturbação**, uma perturbação proposital inserida no sistema forçando-o a entrar novamente em regime transiente; **regime estacionário**, o número de regimes estacionários, bem como seus limites também dependem da atuação dos controladores, visto que um regime estacionário só se inicia após o fim de um período transitório.

Para os experimentos a seguir, o tempo de inicialização das máquinas virtuais foi fixado em 70 *Unidades de Tempo de Simulação (UTS)*¹⁰ e foram submetidas ao sistema 2000 requisições de diferentes usuários. As requisições eram enviadas ao sistema com um intervalo médio de 5 *UTS*, utilizando uma função de distribuição de probabilidade exponencial negativa. A escolha de uma distribuição tal qual um processo de Poisson repousa sobre o fato da distribuição não manter memória de seus valores anteriores e, dessa forma, não influencia na aleatoriedade dos valores futuros. Quando o tempo de simulação foi igual 1500 *UTS*, uma perturbação proposital foi inserida na média do intervalo entre as chegadas de requisições fazendo com que a taxa de chegada de requisições fosse aumentada em 50%.

Os experimentos foram divididos segundo os tempos médios de processamento das requisições, que também foram distribuídos segundo uma função de distribuição de probabilidade exponencial negativa, com médias iguais a 35, 70 e 350 *UTS*. Esses valores foram escolhidos para que se pudesse analisar a resposta do sistema, segundo a atuação dos controladores, para requisições que impõem diferentes cargas ao sistema alvo. As Figuras 5.2(a), 5.2(b) e 5.2(c) ilustram as taxas de utilização observada com os três controladores para os tempos médios de processamento de 35, 70 e 350 *UTS*, respectivamente.

Para as requisições com tempo médio de processamento em torno de 35 *UTS*, é possível observar que os três sistemas apresentaram comportamentos bem parecidos durante grande parte da simulação, com respostas muito próximas umas das outras. Todos mantiveram a taxa de utilização do sistema em torno da referência de 70% no regime estacionário. No entanto, uma observação mais meticulosa permite notar que tanto no *warm-up* quanto na inserção da perturbação (*i. e.* a partir do instante 1500 *UTS*), as respostas às atuações de todos os controladores apresentaram um

¹⁰Um *UTS* equivale a um intervalo de amostra na simulação.

(a)

(b)

aumento na taxa de utilização, caracterizado por um *overshoot* de pequena intensidade e curta duração, que desaparece logo após alguns instantes. Durante o período de aquecimento, o sistema com o controlador 1 apresentou uma atuação sensivelmente distinta dos demais, com um menor tempo de assentamento (80 *UTS* contra 119 *UTS* do controlador 2 e 120 *UTS* do controlador 3), sob o custo de apresentar uma amplitude levemente superior, o que fez com que o mesmo convergisse para o valor de regime estacionário antes dos sistemas com os outros controladores. Além do mais, durante a perturbação, mesmo não havendo diferença significativa entre os tempos de assentamento, há novamente uma pequena superioridade do sistema do controlador 1 que apresentou uma amplitude máxima de 18.489%, enquanto os controladores 2 e 3 tiveram um *overshoot* de 24.789% e 25.364%, respectivamente.

Com requisições um pouco maiores (*i. e.* em média 70 *UTS*), como mostra a Figura 5.2(b), os comportamentos ainda são muito parecidos no regime estacionário, mas no regime transiente há

Figura 5.2 — Taxa de utilização para os tempos médios de processamento 35, 70 e 350

um descolamento maior do sistema controlado pelo controlador 1, enquanto que as respostas aos controladores 2 e 3 ainda se comportam de maneira muito parecida. No período de aquecimento, o controlador 1 fez com que o sistema objetivo convergisse para o valor de regime estacionário após 127 *UTS*, contra exatas 216 *UTS* para os demais. Durante a ação da perturbação, começa a ficar evidente a melhor atuação do sistema do controlador 1 que reflete em uma amplitude máxima de 20.047%, enquanto o sistema controlado por 2 teve um *overshoot* de 27.647% e o sistema controlado por 3 de 27.677%. Não bastasse um menor valor da amplitude máxima, a atuação do controlador 1 fez com que o sistema objetivo convergisse novamente para a referência após um tempo de ajuste de 124 *UTS*, contra 368 *UTS* e 372 *UTS* dos controladores 2 e 3, respectivamente.

Requisições ainda maiores (*i. e.* 350 *UTS* em média) mostram que os comportamentos dos controladores fica ainda mais disjunto e as diferenças de atuação existente entre eles ainda mais perceptíveis. Durante o aquecimento todos os sistemas experimentados atingem a taxa máxima de utilização até que as instâncias de máquinas virtuais necessárias para atender a demanda estejam disponíveis. Nesse período não é possível identificar o sistema de atuação mais justa. No entanto, mesmo com tempo de inicialização constante, é possível perceber que o controlador 1 reage melhor ao atendimento da demanda e produz um tempo de assentamento menor (628 *UTS*), cerca de 30% inferior aos demais (981 *UTS* para o controlador 2 e 989 *UTS* para o controlador 3). Observa-se também que enquanto o controlador 1 faz com que a resposta do sistema objetivo convirja para o valor da entrada de referência, os controladores 2 e 3 fazem com que taxa de utilização estacione sobre a borda inferior do intervalo (*i. e.* 60%). Quando na inserção da perturbação os três controladores apresentaram amplitudes máximas muito próximas com os valores de 16.54%, 17.921% e 17.594%, respectivamente, entretanto, o controlador 1 é o único que teve o comportamento esperado e fez com que a taxa de utilização do sistema objetivo voltasse ao valor da entrada de referência após um tempo de assentamento de 206 *UTS*. A ação dos controladores 2 e 3 fize-

ram com que a resposta do sistema passasse agora ao limite superior do intervalo apresentando uma lenta tendência de decaimento para a referência, o que mostra que esses controladores ainda mantêm o sistema em regime transiente.

A taxa de utilização está diretamente ligada ao número de instâncias de máquinas virtuais disponíveis para atender as requisições submetidas, uma vez que a quantidade de máquinas virtuais foi a entrada de controle de todos os controladores experimentados. Quanto mais próximo do ideal estiver o número de instâncias, mais próximo da entrada de referência a resposta do sistema vai permanecer. As Figuras 5.3(a), 5.3(b) e 5.3(c) ilustram o número de instâncias de máquinas virtuais ligadas para os experimentos com requisições de tamanhos médios de 35, 70 e 350 *UTS*, respectivamente.

(a)

(b)

Os resultados para requisições com tempo médio de processamento de 35 *UTS*, Figura 5.3(a), mostram que o número de instâncias gerados pelos três controladores é muito similar ao longo de

Figura 5.3 — Número de instâncias de máquinas virtuais para os tempos médios de processamento 35, 70 e 350

toda simulação. Todavia, com uma verificação sistemática do gráfico percebe-se que o número instâncias para o primeiro controlador é levemente menor que aquele observado para os demais no regime estacionário. Também é possível verificar que o controlador 1 começa a atuar e a instanciar as máquinas virtuais antes dos outros controladores nos dois períodos de regime transiente.

Com a média do tempo de processamento em 70 *UTS*, Figura 5.3(b), fica mais claro que controlador 1 reage mais rápido e converge para um número médio de máquinas virtuais inferior aos outros dois controladores. Na primeira parte da simulação, do início até o envio da requisição 750, os controladores 2 e 3 instanciam uma quantidade similar de máquinas virtuais. No entanto, a partir da introdução da perturbação, o controlador 2 converge antes do controlador 3 para uma quantidade estacionária de instâncias. Pode-se observar também que a nova quantidade de instâncias no regime estacionário para o controlador 2 é superior a quantidade instanciada pelo controlador 3, isso significa que, apesar de mais lento, nesse caso o controlador 3 convergiu para um número menor de máquinas virtuais que o controlador 2.

Quanto maior o tamanho médio das requisições mais clara fica a diferença entre a quantidade de máquinas virtuais que cada controlador instanciam, como mostra a Figura 5.3(c). É nítido como o controlador 1 reage mais rápido que os demais durante o *warm-up*, sob o preço de ligar mais máquinas que o necessário. O primeiro controlador é levado a ligar 122 instâncias e depois corrige essa quantidade diminuindo para 112 máquinas virtuais no regime estacionário. Da mesma forma, durante o período de aquecimento é possível verificar que os controladores 2 e 3 tiveram comportamento muito parecidos até a requisição 750 quando a perturbação na taxa de chegadas de requisições é inserida e, a partir de então, o controlador 2 convergiu para 136 instâncias, contra 141 do terceiro controlador. Verifica-se também que, mesmo com uma forte perturbação, a reação de ambos foi lenta, pouco perceptível e teve menor intensidade que a reação do primeiro

controlador. Após a perturbação, o controlador 1 passa de 112 para 150 máquinas virtuais ligadas, uma quantidade bem superior a dos outros controladores. A quantidade de instâncias após a perturbação explica a taxa de utilização em torno do limite superior na Figura 5.2(c), evidenciando que as quantidades para as quais convergiram os controladores 2 e 3 são insuficientes para levar a utilização do sistema ao valor de referência. Por outro lado, apesar de alcançar um número maior de instâncias, o controlador 1 consegue com isso manter a taxa de utilização muito próxima do valor esperado.

O número de instâncias de máquinas virtuais ligadas tem efeito direto na principal métrica de desempenho sentida pelo usuário, o tempo médio de resposta. Em uma situação ideal, o tempo de resposta de uma requisição deve ser influenciado apenas pelo tamanho da mesma, evitando o reflexo de outras variáveis como, por exemplo, o tempo de espera em fila. Se houver máquinas virtuais ligadas em número suficiente para atender as requisições, é provável que o tempo médio de resposta pouco seja influenciado pelo tempo de espera em fila. As Figuras 5.4(a), 5.4(b), 5.4(c) ilustram o tempo médio de resposta para cada um dos experimentos realizados.

(a)

A Figura 5.4(a) mostra que os controladores mantiveram o tempo médio de resposta do sistema entre 30 e 40 *UTS* a maior parte da simulação e absorveram bem tanto o período de aquecimento, quanto a perturbação proposital. Isso fez com que esses períodos transitórios fossem quase imperceptíveis no tempo médio de resposta. Todavia, novamente com um olhar criterioso percebe-se que a atuação do primeiro controlador fez com que o sistema objetivo apresentasse um tempo médio de resposta sensivelmente menor que os demais nos dois períodos transitórios. Essa diferença fica mais clara com requisições de tamanho maior, como mostra a Figura 5.4(b).

Assim como na Figura 5.4(a), os controladores apresentaram comportamentos muito semelhantes no regime estacionário, mesmo com o aumento do tempo médio de resposta que se concentraram entre 60 e 80 *UTS*. Mais uma vez, é possível notar que nos períodos transitórios, o controlador 1 teve os melhores resultados.

(b)

(c)

Figura 5.4 — Tempo médio de resposta para os tempos médios de processamento 35, 70 e 350

Para requisições muito grandes, como na Figura 5.4(c), fica nítido como a atuação dos controladores pode influir nos objetivos de desempenho de um sistema. A atuação do controlador 1 no período de aquecimento fez com que a média do tempo de resposta ficasse muito próxima daquela em regime estacionário, enquanto os outros controladores fizeram com que os tempos médios de resposta aumentassem substancialmente. Os reflexos da tardia atuação dos controladores 2 e 3 no *warm-up* só deixaram de existir após o distúrbio proposital.

Quanto maior o tempo de espera, maior será o tempo médio de resposta. Uma possível consequência do aumento do tempo médio de resposta é a aparição de requisições mal atendidas (*i. e.* sem que os requisitos de *QoS* sejam respeitados) e como consequência o surgimento de requisições penalizadas. Para esses experimentos foi considerada penalizada toda requisição que teve o tempo de resposta maior que 50% do seu tempo de processamento exigido (*e. g.* uma requisição

que levaria 100 *UTS* para ser completamente processada tem um *deadline* de 150 *UTS* até que seja considerada penalizada). As Figuras 5.5(a), 5.5(b) e 5.5(c) mostram a quantidade de requisições penalizadas usando cada um dos controladores nos experimentos conduzidos, bem como o intervalo de confiança calculado para 95%. Esses dados são sumarizados na Tabela 5.1, mostrando a média (μ) das requisições mal atendidas levando em conta todas as repetições, bem como o valor mínimo (min) e máximo (max) do intervalo de confiança.

Na Figura 5.5(a), muito embora as médias de requisições penalizadas sejam bem diferentes, há uma sobreposição dos intervalos de confiança encontrados que torna a análise dos resultados inconclusiva e não é possível afirmar que um controlador teve comportamento melhor que os demais para requisições com tamanho médio de 35 *UTS* no que tange a quantidade de requisições penalizadas. Para os controladores 2 e 3 é possível afirmar que o comportamento foi equivalente tendo em vista que o intervalo de confiança de cada controlador abrange a média do outro. Todavia, os

Figura 5.5 — Número médio de requisições penalizadas para os tempos médios de processamento 35, 70 e 350 *UTS*

resultados para as outras variáveis de saída (*e. g.* taxa de utilização, tempo médio de resposta e *throughput*) ainda podem fornecer um comparação contundente sobre a melhor possibilidade. Já a Figura 5.5(b) mostra que o controlador 1 conseguiu manter o desempenho do sistema em níveis que penalizaram um número menor de requisições que os outros sistemas. Ademais, não é possível afirmar nesse caso qual dos controladores, 2 ou 3, teve melhor desempenho, pois, novamente, há uma sobreposição dos intervalos de confiança.

Para requisições com tempo médio de processamento de 70 *UTS*, como exposto na Figura 5.5(b), os controladores 2 e 3 outra vez apresentam funcionamento equivalente e revelam quantidades de requisições penalizadas muito semelhantes, ao contrário do controlador 1 que teve uma quantidade média de requisições penalizadas bem menor. Isso mostra que, mesmo que muito sutil, o desempenho do controlador 1 em relação aos outros nos períodos transientes para a quantidade de instâncias, a taxa de utilização e o tempo médio de resposta resultou numa quantidade menor de requisições penalizadas, que pode implicar no aumento do ganho percebido pelo consumidor. Mais que isso, o controlador 1 conseguiu uma quantidade menor de penalizações com menos instâncias de máquinas virtuais em regime estacionário, aumentando a quantidade de instâncias somente em regime transiente, como ilustrado na Figura 5.3(b).

A mesma conduta pode ser vista na Figura 5.5(c) para requisições com tempo médio de processamento de 350 *UTS*. Mais uma vez, os controladores 2 e 3 apresentam atuações indistinguíveis, enquanto o controlador 1 obteve o menor número de requisições penalizadas.

Embora os controladores apresentem desempenhos similares quando o sistema se encontra em regime estacionário, o mesmo não acontece nos períodos transitórios onde a atuação dos controladores é mais incisiva. Além disso, como os controladores demonstraram o mesmo comportamento estacionário, pode-se concluir que a diferença observada na quantidade de requisições mal atendi-

Tabela 5.1 — Resumo das requisições penalizadas para 35, 70 e 350 UTS

Controlador	Tempo médio de processamento								
	35 UTS			70 UTS			350 UTS		
	min	μ	max	min	μ	max	min	μ	max
1	46.64	51.94	57.24	18.60	20.56	22.53	112.00	118.19	265.59
2	56.18	62.44	68.70	33.31	37.62	41.94	251.74	259.25	266.76
3	55.96	60.37	64.79	33.70	37.50	41.29	247.78	256.69	265.59

das é fruto exclusivamente da atuação do controlador no regime transiente, o que evidencia ainda mais a importância desses períodos no comportamento do sistema e no alcance dos objetivos, sejam de rendimento ou de desempenho. A Figura 5.6(a) e 5.6(b) ilustram o crescimento do número de requisições penalizadas com base no tipo de controlador utilizado e no tempo médio de processamento, respectivamente.

Pela Figura 5.6(a) percebe-se que enquanto o aumento no tempo médio de processamento para os controladores 2 e 3 faz com que o número de requisições penalizadas aumente exponencialmente, o mesmo não acontece com o controlador 1 que apresenta um crescimento linear.

A Figura 5.6(b) traz uma comparação do espalhamento da quantidade de requisições penalizadas segundo o aumento do tempo médio de processamento. É possível perceber que o aumento do tempo médio de processamento não provoca uma dispersão muito grande de requisições penalizadas. Também é possível verificar que o aumento no tempo médio de processamento não causa o mesmo impacto no sistema com o controlador 1 e que é causado nos demais.

Os resultados dos experimentos mostram que a justa atuação do controlador no regime transiente, mesmo que de forma tênue pode influenciar sobremaneira o comportamento do sistema como um todo e essa consequência pode ser observada nas variáveis de saída do sistema. De mais a mais, o correto projeto de um controlador pode ser determinante no atendimento de requisitos de desempenho.

Além de influenciar o desempenho, alguns comportamentos não podem ser capturados pelas métricas estacionárias, como a estabilidade e o valor estacionário, e só são observáveis no regime transiente. É preciso considerar que o comportamento do sistema em períodos transitórios pode levá-lo a condições além da sua capacidade, fazendo com que o mesmo deixe atender aos objetivos propostos.

5.4 Protótipo da arquitetura do *broker* de consumidor

Para validar a arquitetura de *broker* de consumidor proposta foi desenvolvido um protótipo da mesma usando as bibliotecas de simulação do CloudSim. Adicionalmente, um ambiente configurável de simulação foi criado para dar suporte a experimentação da arquitetura sob diferentes condições.

(a)

(b)

Figura 5.6 — Curva de aumento de requisições penalizadas

Apesar de permitir a construção de simuladores de ambientes de nuvens robustos, o CloudSim possui algumas limitações na sua versão original que dificultam a representação de uma série de cenários e situações de interesse de quem pretende desenvolver uma aplicação ou solução baseada em computação em nuvem.

As classes disponíveis na *API* original não permitem a criação de cenários de simulação nos quais a submissão de um conjunto de tarefas seja realizado durante o tempo de simulação, essa limitação implica que toda a carga de trabalho que se pretende experimentar seja submetida, em lote, logo no início da simulação, o que impossibilita, por exemplo, a simulação de um ambiente em que há a alteração da carga de trabalho introduzida no sistema ao longo da simulação, como é o foco deste trabalho. Para distribuir a submissão da carga de trabalho ao longo da simulação, uma estratégia utilizada é criar um topologia de rede que especifique os nós, os atrasos entre as comunicações entre nós e *data center* e a quantidade de carga submetida por cada um em um arquivo separado do simulador, usando a estratégia proposta por Medina et al. (2001). Entretanto, essa alternativa ainda não resolve o problema de submissão da carga de trabalho segundo uma função de distribuição de probabilidade, além de que, essa solução obriga que toda uma topologia de rede seja projetada segundo os tempos de submissão que se espera, o que pode não fazer parte do objeto de estudo, e que todos os tempos de submissão de carga, bem como seus tamanhos, sejam estabelecidos antes do início da simulação. Essa limitação aumenta a curva de aprendizado, o tempo de projeto de um simulador e pode inviabilizar a simulação como uma alternativa à execução de experimentos de ambientes de computação em nuvem.

O CloudSim também não suporta nativamente a alocação e a deslocação dos recursos de uma máquina virtual em tempo de simulação, apesar de não impossibilitar, sendo que todo o conjunto de capacidades computacionais que compõem os *data centers* e as máquinas virtuais devem ser criados e alocados antes do início da simulação.

Para amenizar as influências dessas limitações, uma série de implementações foram necessárias para estender a *API* original da biblioteca. Além de implementar a arquitetura proposta, o protótipo da arquitetura estende as classes já existentes no CloudSim para permitir a criação de simuladores mais dinâmicos que permitam a elasticidade de recursos e a submissão de cargas de trabalho a qualquer tempo com a utilização de processos estocásticos, caso necessário. A Figura 5.7 mostra um diagrama de classes simplificado do protótipo da arquitetura de *broker* de consumidor e de algumas classes da biblioteca de simulação envolvidas. No diagrama as classes em destaque fazem parte da arquitetura e foram implementadas como uma extensão da biblioteca CloudSim durante este trabalho.

A classe **Monitors** corresponde a uma classe abstrata estendida e implementada pelas classes **RequestMonitor**, **VMMonitor** e **RevenueMonitor**. Enquanto a classe **VMMonitor** monitora e gerencia o ciclo de vida das instâncias de máquinas virtuais, a classe **RequestMonitor** mantém informações sobre as requisições que foram submetidas pelos usuários.

Figura 5.7 — Diagrama simplificado de classes do protótipo do *Broker*

O *VMMonitor* mantém um conjunto de listas e contadores que permitem fornecer estatísticas atualizadas sobre cada uma das instâncias de máquinas virtuais que fazem parte do conjunto de recursos computacionais utilizados. Dentre os principais componentes dessa classe estão:

- **vmsList**: mantém uma lista de todas as instâncias que já foram requisitadas pelo **broker** ao *data center*. Cada máquina virtual possui um identificador único e nessa lista também são mantidos os identificadores das máquinas criadas antes e em tempo de simulação. Para

facilitar as análises posteriores, o VMMonitor estabelece um deslocamento entre os identificadores das instâncias iniciais e criadas em tempo de simulação.

- **vmsCreatedList:** mantém uma lista de todas as máquinas disponíveis para o processamento de requisições. As informações da *vmCreatedList* são usadas para aplicação da política de escalonamento de requisições utilizada. O VMMonitor mantém nessa lista apenas as máquinas que estão no estado Inicializada.
- **vmsToBeIdleList:** contém as máquinas virtuais que ainda processam requisições, mas que serão finalizadas tão logo terminem o processamento das requisições que já foram submetidas. Nessa lista estão todas as máquinas no estado Desligando.
- **bootingVmsList:** guarda as instâncias em inicialização que estarão disponíveis em breve. As listas *vmsToBeIdleList* e *bootingVmsList* são usadas para otimizar o processo de ligamento e desligamento das máquinas virtuais.
- **vmsIdleList:** armazena as instâncias que já foram utilizadas em algum momento, mas que agora se encontram no estado Desligada. Além de fornecer informações sobre a utilização das máquinas virtuais, essas listas são usadas para o gerenciamento do ciclo de vida das diversas instâncias que são criadas e destruídas ao longo de uma simulação.

Além das listas, o VMMonitor mantém alguns contadores para guardar a quantidade de instâncias requisitadas ao *data center* e a quantidade de máquinas virtuais que o *data center* conseguiu alocar na infraestrutura física. A implementação do VMMonitor também prevê a disposição de instâncias em mais de um *data center* usando mapeamentos. Essa classe é capaz de fornecer informações sobre o histórico e a taxa de utilização de cada máquina virtual utilizada pela aplicação do consumidor, bem como o tempo que cada uma permaneceu ligada, quando foram ligadas e desligadas e a quantidade de requisições processadas. Uma outra importante função do VMMonitor é manter o correto mapeamento das instâncias para as suas filas de requisições pendentes.

Análogo ao que o VMMonitor faz pelas máquinas virtuais, o RequestMonitor faz pelas requisições, que no CloudSim são referenciadas pelo termo *Cloudlets*. Os principais elementos que foram a estrutura da classe Request Monitor são:

- **cloudletSubmittedList:** guarda uma lista com todas as características de todas as requisições que já foram submetidas ao sistema. Cada requisição possui um identificador que é associado a instância responsável pelo processamento.
- **cloudletReceivedList:** mantém informações sobre todas as requisições que já foram processadas. Essas informações são importantes e podem ser consultadas por outros módulos para, por exemplo, calcular o rendimento obtido pelo *broker* ao longo do ciclo de vida da aplicação.

- **cloudletPenalizedList**: guarda uma lista com informações das requisições que foram mal atendidas e foram consideradas penalizadas. É o RequestMonitor o módulo responsável por determinar se uma requisição atendida deve ser penalizada ou não com base em informações fornecidas antes do início da simulação.

O RequestMonitor também mantém uma série de variáveis que são replicações de informações que podem ser obtidas dinamicamente como a quantidade de requisições penalizadas que pode ser obtida contando os elementos que fazem parte da cloudletPenalizedList, entretanto, para diminuir a disputa por recursos entre os módulos da arquitetura e o sistema objetivo, essas informações são replicadas para facilitar o fornecimento das mesmas aos módulos que possam, porventura, requisitá-las (e. g. o RevenueMonitor pode requisitar a quantidade de requisições penalizadas desde o último instante em que o rendimento foi calculado). O RequestMonitor também mantém as estatísticas sobre cada requisição submetida como o tempo de submissão, tempo de espera em fila, tempo de processamento e tempo de resposta, além de manter um mapeamento sobre qual instância foi responsável por qual requisição, o tempo médio de resposta e o *throughput* do sistema.

As classes RequestMonitor e VMMonitor estão diretamente ligadas a duas extensões da biblioteca CloudSim que foram implementadas neste trabalho. A primeira permite a criação e submissão de requisições em tempo de simulação usando uma função de distribuição de probabilidade, caso necessário; e a segunda permite o gerenciamento de ciclo de vida de instâncias de máquinas virtuais em tempo de simulação. Para implementação dessas extensões foi necessário modificar o código original da biblioteca, inserindo novos eventos de baixo nível que fossem reconhecidos pelo núcleo de simulação e enviados ao manipulador correto. Três principais eventos foram criados: **InsertCloudletEvent**, **StartupVmBootEvent** e **SampleTimeInterruptEvent**. Este terceiro evento tem a função de permitir que intervalos de amostra com tamanhos fixos sejam usados em um simulador. Sempre que um evento do tipo SampleTimeInterruptEvent acontece, todos os valores das variáveis de interesse são atualizados.

O InsertCloudletEvent dispara a criação de novas requisições em tempo de simulação, e pode ser espaçado segundo um conjunto de funções de distribuição de probabilidade. Além do RequestMonitor, este evento afeta diretamente o módulo *Dispatcher*. Um StartupVmBootEvent é programado sempre que uma nova máquina precisa ser ligada, e não há máquinas que estejam no estado Desligando. Esse evento faz com que uma série de atualizações sejam procedidas no VMMonitor para que uma nova instância seja criada e disponibilizada para o atendimento de requisições. O acontecimento de um evento desse tipo deve promover o agendamento do próximo para que as requisições sejam criadas e submetidas indefinidamente durante todo o tempo de simulação.

A classe RevenueMonitor dá sentido a orientação a mercado da arquitetura arquitetura proposta e é responsável por consultar os outros monitores sobre a quantidade de recursos utilizados e quão bem as requisições tem sido atendidas. A partir da configuração de um conjunto de constantes como **DatacenterCost**, **DatacenterCostPerMemory**, **DatacenterCostPerStorage**, **DatacenterCostPerBW**, **CostPerPenalized** e da implementação de uma função custo que relaciona a

quantidade de recursos utilizados e os custos envolvidos, o *RevenueMonitor* é capaz de calcular e manter informações sobre os rendimentos atuais e passados da aplicação do cliente de acordo com o desempenho conseguido pelo *broker*.

A classe **Dispatcher** é uma classe abstrata que deve ser estendida pela classe que implementa a política de balanceamento de carga de interesse. O estudo de políticas de balanceamento de carga está fora do escopo deste projeto de pesquisa e, sendo assim, apenas alguns algoritmos muito simples de balanceamento foram implementados. Para os experimentos executados, o balanceamento de carga era realizado distribuindo as requisições sempre para as máquinas virtuais com menor carga, informação essa obtida junto ao *VMMonitor*. Uma vez escolhida a instância que processa a requisição, o *VMMonitor* utiliza informações do *RequestMonitor* para atualizar as estatísticas da instância escolhida. A ação de distribuição do *Dispatcher* é solicitada pelo *broker* sempre que um evento do tipo *InsertCloudletEvent* acontece.

Sem dúvida, o principal módulo da arquitetura reside na classe abstrata *Controller*. Apesar de muitas das funções do módulo Controlador e Provisor de recursos estarem espalhadas em diversas classes do protótipo, toda a dinâmica do controlador é implementada numa única classe que deve herdar as funcionalidades de *Controller*, como a classe *PIDController*. Mesmo com o foco do trabalho no desenvolvimento de controladores inspirados na teoria clássica de controle, o protótipo permite que controladores criados por outros métodos, por exemplo, lógica *fuzzy*, possam ser implementados e adicionados arquitetura.

Enquanto os outros módulos desempenham, na sua maioria, funções simples, mas fundamentais ao objetivo da arquitetura, o poder de decisão do *broker* de consumidor reside nas classes que estendem *Controller*. As próximas seções dão detalhes de como o sistema objetivo foi modelado até que o controlador utilizado fosse obtido.

5.4.1 Modelagem do sistema objetivo

Para que seja possível projetar um controlador adequado para um determinado sistema objetivo, é necessário que se conheça de que forma o sistema reage às entradas de controle e como isso reflete na variável de resposta escolhida. Neste projeto, o sistema objetivo foi modelado como um sistema de malha aberta, como o ilustrado na Figura 3.3, usando a técnica para identificação de sistema proposta por Parekh et al. (2002). Nessa técnica o sistema objetivo é analisado como uma caixa preta da qual se pretende extrair a dinâmica de comportamento sem que seja necessário de fato ter conhecimento específico sobre o seu funcionamento interno. Para isso, o sistema de malha aberta é submetido a um sinal de entrada e a resposta de interesse correspondente é medida para que se possa extrair um modelo matemático que o represente.

Como visto na seção 3.4, um sistema pode ser modelado por um modelo *AR* ou *ARMA*, tendo por base uma série de dados que relacionam a sua entrada e a sua saída. Para modelagem do sistema objetivo neste projeto a entrada do sistema foi representada pela a entrada de controle do

mesmo (*i. e.* a quantidade de instâncias de máquinas virtuais que deveriam estar ligadas para atender a demanda), enquanto que a saída escolhida foi a variável de controle tida como a resposta do sistema (*i. e.* a taxa de utilização dos recursos dessas máquinas virtuais).

Assim, para analisar a resposta à variação da entrada de controle, o sistema foi submetido a uma variação dentro de um intervalo da quantidade de máquinas virtuais, conforme mostra a Figura 5.8 e suas saídas foram medidas para a modelagem, como mostra a Figura 5.9. O sistema objetivo foi configurado inicialmente para que 26 máquinas virtuais fossem ligadas até que o mesmo entrasse no ponto de operação planejado, ou seja, a taxa de utilização deveria estar em torno de 54%, o que correspondeu aos primeiros 49 intervalos do experimento. Os primeiros intervalos foram considerados negativos para que a mudança prevista na entrada de controle acontecesse exatamente no instante 0, quando então a quantidade de máquinas é diminuída de uma única vez, ou seja, em degrau, para 14 instâncias. A partir de então a dinâmica do sistema eleva a taxa de utilização para aproximadamente 99% após alguns instantes.

Figura 5.8 — Entrada de controle (número de máquinas virtuais) para modelagem do sistema objetivo

Pela Figura 5.9 é possível concluir que a resposta medida para uma entrada em degrau apresenta um comportamento de sistema de primeira ordem, uma vez que a intensidade das oscilações apresentadas é pequena e pode ser atribuída às estocasticidades envolvidas no experimento. Dessa forma, a dinâmica do sistema pode ser representada por um modelo autorregressivo de primeira ordem como o mostrado pela equação 5.1, em que $\hat{y}(k+1)$ é o valor da saída previsto no instante $k+1$ com base nos valores da entrada aplicada ao sistema em k , $u(k)$, e na saída medida ou prevista em k , $y(k)$ ou $\hat{y}(k)$.

Figura 5.9 — Resposta do sistema objetivo (taxa de utilização) à variação na entrada de controle

$$\hat{y}(k+1) = ay(k) + bu(k) \quad (5.1)$$

Para facilitar a manipulação numérica na modelagem, os valores de entrada e saída foram normalizados em torno dos valores médios encontrados para essas medidas (*i. e.* 16.955 instâncias de máquinas virtuais para a entrada e 84.406% de utilização para a saída), como mostram as equações 5.2 e 5.3, respectivamente. Nessas equações $\tilde{u}(k)$ e $\tilde{y}(k)$ correspondem aos valores absolutos observados para a entrada e a saída, enquanto que $u(k)$ e $y(k)$ correspondem aos mesmos valores normalizados. A lista completa com todos os valores normalizados utilizados na modelagem do sistema objetivo pode ser encontrada no Anexo A deste documento.

$$u(k) = \tilde{u}(k) - 16.955 \quad (5.2)$$

$$y(k) = \tilde{y}(k) - 0.84406 \quad (5.3)$$

Visto que os valores da entrada e saída foram desviados com relação às suas médias, foi aplicado o método dos mínimos quadrados para que os parâmetros a e b da equação 5.1 fossem estimados. Os valores encontrados para os parâmetros a e b foram $a = 0.57961$ e $b = -0.016263$, que resultou no modelo matemático do sistema objetivo dado pela equação 5.4.

$$y(k+1) = 0.57961y(k) - 0.016263u(k) \quad (5.4)$$

Para comprovar que o modelo encontrado é válido e verificar a adequação do mesmo ao sistema objetivo, os valores da saída previstos para o modelo, $\hat{y}(k)$, foram confrontados com os valores reais medidos, $y(k)$, em dois gráficos, como mostram as Figuras 5.10(a) e 5.10(b). A Figura 5.10(a) ilustra num mesmo gráfico as saídas reais e previstas ao longo da simulação, é possível perceber que as linhas praticamente se sobrepõem, o que sinaliza para um bom ajuste do modelo.

Figura 5.10 — Comparação da resposta real medida com a resposta prevista pelo modelo

Na Figura 5.10(b), as saídas são representadas utilizando como coordenadas o valor da saída real e prevista num mesmo instante de tempo. A figura também ilustra um segmento de reta diagonal onde $\hat{y}(k) = y(k)$ que serve de referência à comparação dos valores. Para que o modelo esteja bem ajustado é necessário que os pontos estejam próximos ou sobre a diagonal traçada, como foi o caso da Figura 5.10(b) que comprova, visualmente, a pertinência do modelo encontrado para representar o sistema objetivo.

Além da comprovação visual dada pelo gráfico, também foram calculados os valores para os testes da raiz quadrada da média do erro, da variabilidade R^2 e da correlação entre a entrada e o erro residual, os quais foram sumarizados na Tabela 5.2.

Tabela 5.2 — Valores de R^2 , $RMSE$ e correlação entre entrada e erro residual para o modelo estimado

Teste	Valor calculado
R^2	0.98873
$RMSE$	0.021981
Correlação	0.19890

A análise dos valores calculados para os testes reforça a hipótese de que o ajuste do modelo estimado é adequado ao sistema objetivo, uma vez que os valores para a raiz quadrada da média do erro e a correlação entre a entrada e o erro residual foram baixos, e o valor da variabilidade R^2 é muito próximo de 1.

Aplicando a transformada Z nos sinais envolvidos na equação 5.4 e realizando as devidas manipulações, é possível encontrar a função de transferência do sistema objetivo que é dada pela equação 5.5. Tendo a função de transferência que representa a planta a ser controlada é possível seguir com o projeto de um controlador que mantenha o comportamento dessa planta dentro dos limites apropriados.

$$G(z) = \frac{Y(z)}{U(z)} = \frac{-0.016263}{z - 0.57961} \quad (5.5)$$

5.4.2 Projeto do módulo controlador e provisor de recursos

O Controlador e provisor de recursos é o módulo de arquitetura responsável pelas tomadas de decisão do *broker*. Esse módulo deve ser projetado segundo os interesses do consumidor e as características do sistema objetivo. Toda ação do controlador e provisor de recursos gira em torno de um controlador, cuja entrada de controle é a quantidade de recursos necessários para atender a demanda atual. Essa entrada de controle é usada para readequar as capacidades computacionais e disponibilizá-las ao VMMonitor que, por sua vez, informá-la-as aos demais módulos do *broker*.

Para implementar esse módulo foi desenvolvido um controlador do tipo proporcional-integral com suavização da resposta por uma média móvel que leva em consideração o sistema objetivo modelado na seção 5.4.1. Esse tipo de controlador foi escolhido por ser apropriado ao controle de plantas que apresentam o comportamento definido pela interação de muitos processos estocásticos que acabam refletindo numa resposta de muita variação. Essa adequação deve-se à exploração de um filtro de média móvel que reduz a influência da variação da resposta no erro calculado, e de um ganho integral, que absorve as variações de curta duração e reflete na entrada de controle apenas aquelas que foram acumuladas e tiveram duração significativa na saída.

O projeto do controlador desenvolvido utiliza o método definido na seção 4.2. Como critério de projeto foi admitido que, em regime transiente, o sistema objetivo não deveria apresentar uma amplitude máxima superior a 50%, $M_p = 0.5$, e que o tempo de assentamento não deveria ser maior que 50 intervalos de amostra, $K_s = 50$. Tendo em vista essas escolhas e o método definido neste projeto, foi possível encontrar os valores para a magnitude, equação 5.6, e o ângulo no plano complexo, equação 5.7, dos polos dominantes p_1 e p_2 . De posse dessas informações, foi possível calcular as representações exponenciais e cartesianas dos polos p_1 e p_2 dadas pelas equações 5.8 e 5.9, respectivamente.

$$\begin{aligned} r &= |p_1| \\ r &= e^{(-4/K_s)} \\ r &= e^{(-4/50)} \end{aligned}$$

$$r = 0.92312 \tag{5.6}$$

$$\begin{aligned} \theta &= \pi \frac{\log(r)}{\log(M_p)} \\ \theta &= \pi \frac{\log(0.92312)}{\log(0.5)} \end{aligned}$$

$$\theta = 0.36257 \tag{5.7}$$

$$p_1 = 0.92312e^{0.36257j} = 0.86311 + 0.32741j \tag{5.8}$$

$$p_2 = 0.92312e^{-0.36257j} = 0.86311 - 0.32741j \tag{5.9}$$

Substituindo os parâmetros do modelo do sistema objetivo e as coordenadas polares ou cartesianas dos polos encontrados nos sistemas de equações lineares 4.21 ou 4.26, é possível obter os ganhos do controlador proporcional K_p e integral K_i resolvendo as equações e tendo o parâmetro da média móvel c como um grau de liberdade do projeto. Optando por trabalhar com as coordenadas polares e substituindo a magnitude r e o ângulo θ do polo p_1 , bem como os parâmetros a e b do modelo, no sistema de equações lineares 4.26, foi obtido o sistema de equações lineares 5.10.

$$\begin{cases} 1.7262 + p_3 = 1.57961 + c \\ 0.85216 + 1.7262p_3 = 0.57961 + 1.57961c + (-0.016263)(1 - c)(K_i + K_p) \\ 0.85216p_3 = 0.57961c + (-0.016263)(1 - c)K_p \end{cases} \quad (5.10)$$

Para esse sistema de controle de malha fechada, é possível substituir os valores encontrados nas inequações 4.22 e 4.27 e verificar que os valores possíveis do parâmetro de média móvel c que não infringem os critérios do projeto estão dentro do intervalo 5.11, mas como os valores de c só fazem sentido se estiverem entre 0 e 1, a interseção desses intervalos mostra que os valores factíveis desse parâmetro são dados, na verdade, pelo intervalo 5.12.

$$r[2 \cos \theta - 1] - a - 1 \leq c \leq r[2 \cos \theta + 1] - a - 1$$

$$0.92312[2 \cos(0.36257) - 1] - 1.57961 \leq c \leq 0.92312[2 \cos(0.36257) + 1] - 1.57961$$

$$-0.77652 \leq c \leq 1.0697$$

$$c \in [-0.77652, 1.0697] \quad (5.11)$$

$$c \in [-0.77652, 1.0697] \cap [0, 1[\implies c \in [0, 1[\quad (5.12)$$

Consequentemente, o terceiro polo da malha fechada p_3 tem seus valores possíveis dados pelo intervalo 5.13.

$$-0.14659 \leq p_3 < 0.85341$$

$$p_3 \in [-0.14569, 0.85341[\quad (5.13)$$

A partir das coordenadas dos polos, é possível encontrar o polinômio característico que representa o comportamento desejado para o sistema de controle de malha fechada. Tratando p_3 como um polo desconhecido, já que o mesmo pode ser definido em função do parâmetro de média

móvel c como pode ser observado na primeira equação do sistema linear 5.10, e usando as coordenadas dos polos p_1 e p_2 , a equação 5.14 exhibe o polinômio característico para o sistema com as características desejadas.

$$D_d(z) = (z - p_1)(z - p_2)(z - p_3)$$

$$D_d(z) = z^3 - (p_1 + p_2 + p_3)z^2 + (p_1p_2 + p_1p_3 + p_2p_3)z - p_1p_2p_3$$

$$D_d(z) = z^3 - (1.7262 + p_3)z^2 + (0.85215 + 1.7262p_3)z - 0.85215p_3 \quad (5.14)$$

Com a função de transferência do sistema objetivo, dada pela equação 5.5, é possível compor o sistema de malha fechada, como mostrado pela Figura 5.11 em que estão explícitas as funções de transferência do controlador ($C(z)$), da planta ($G(z)$) e o transdutor ($H(z)$) envolvidos na malha de controle.

Figura 5.11 — Diagrama de blocos do sistema de malha fechada com a função de transferência do sistema objetivo

Partindo desse diagrama é possível encontrar a função de transferência do sistema de controle de malha fechada, como mostra a equação 5.15.

$$F_r(z) = \frac{C(z)G(z)}{1 + C(z)G(z)H(z)}$$

$$F_r(z) = \frac{\left(K_i \frac{z}{z-1} + K_p\right) \left(\frac{-0.016263}{z-0.57961}\right)}{1 + \left(K_i \frac{z}{z-1} + K_p\right) \left(\frac{-0.016263}{z-0.57961}\right) \left(\frac{1-c}{z-c}\right)}$$

$$F_r(z) = \frac{(K_i z + K_p z - K_p)(-0.016263)}{(z-1)(z-0.57961)(z-c) + (K_i z + K_p z + K_p)(-0.016263)(1-c)}$$

$$F_r(z) = \frac{(K_i z + K_p z - K_p)(-0.016263)(z - c)}{(z - 1)(z - 0.57961)(z - c) + (K_i z + K_p z + K_p)(-0.016263)(1 - c)} \quad (5.15)$$

Como é do interesse desta análise que o polinômio característico seja encontrado, já que dele é caracterizado o comportamento do sistema de controle, é possível dividir $F_r(z)$ conforme a equação 3.26, tendo como resultado o polinômio característico dado pela equação 5.16.

$$D_m(z) = z^3 - (c + 1.57961)z^2 + (1.57961c + 0.57961)z - 0.57961c + (K_i z + K_p z - K_p)(-0.016263)(1 - c)$$

$$D_m(z) = z^3 - (c + 1.57961)z^2 + (1.57961c + 0.57961)z - 0.57961c + (K_i + K_p)(-0.016263)(1 - c)z - (-0.016263)(1 - c)K_p$$

$$D_m(z) = z^3 - (c + 1.57961)z^2 + [1.57961c + 0.57961 + (K_i + K_p)(-0.016263)(1 - c)]z - [0.57961c + (-0.016263)(1 - c)K_p] \quad (5.16)$$

Perceba que a comparação termo a termo dos polinômios característicos $D_d(z)$, equação 5.14, e $D_m(z)$, equação 5.16, resulta num sistema de equações lineares equivalente ao 5.10, que comprova que a utilização do método desenvolvido na seção 4.2 facilita o desenvolvimento de um controlador desse tipo e obtém o mesmo resultado que o desenvolvimento dos polinômios característicos.

Uma vez que um sistema de equações lineares como o 5.10 foi desenvolvido, é possível estimar os ganhos de controle para diferentes valores do parâmetro c do filtro de média móvel. A Tabela 5.3 associa os valores encontrados para o terceiro polo e o ganho dos controladores para diferentes valores de c .

Tabela 5.3 — Diferentes valores de c , p_3 , K_p e K_i

c	p_3	K_p	K_i
0	-0.14660	7.6817	-8.8791
0.25	0.10340	4.6562	-9.2576
0.5	0.35340	-1.3949	-10.014
0.75	0.60340	-19.548	-12.285

Diferentes valores de c resultam em diferentes localizações do terceiro polo no plano complexo. No entanto, desde que os valores escolhidos para c estejam dentro intervalo especificado, p_3 não

será dominante e não será determinante no comportamento do sistema, como ilustrado pela Figura 5.12 onde diferentes localizações do terceiro polo são mostradas no plano complexo. É interessante notar que todos os polos têm magnitude menor que p_1 e p_2 e encontram-se dentro do círculo de raio igual a r , que é a magnitude do polo dominante. Ademais, o método desenvolvido faz com que todos os polos que não são o par dominante sejam reais, embora não seja possível encontrar um terceiro polo complexo desde que tenha a mesma magnitude do polo real.

Figura 5.12 — Diferentes localizações para o polo p_3 de acordo com os valores do parâmetro do filtro de média móvel c

Para cada localização diferente do polo p_3 , os ganhos de controle K_p e K_i variam para manter o comportamento do sistema dentro do que se espera. Para verificar que isso acontece de fato foram realizados experimentos alterando os parâmetros do controlador do *broker* para os valores da Tabela 5.3. Os experimentos foram repetidos 16 vezes, com diferentes chaves para os geradores de números randômicos e cada experimento durou 1500 *UTS*. Foram inseridas duas perturbações no sistema além do tempo de aquecimento, a primeira foi inserida no tempo de simulação igual a 550 *UTS* e dobrava a taxa entre as chegadas de requisições, forçando a um aumento no número de máquinas. No instante 1050 *UTS* da simulação outra perturbação era inserida agora diminuindo taxa de chegadas de requisições forçando a uma redução na quantidade de recursos alocados. A Figura 5.13 ilustra as taxas de utilização para os experimentos realizados com base nos valores da Tabela 5.3.

É possível perceber que, apesar da alternância de valores para os ganhos de controle para diferentes valores do parâmetro de média móvel c , independente do grau de influência das respostas

Figura 5.13 — Taxa de utilização para $c = 0, 0.25, 0.50, 0.75$

anteriores na saída medida, os resultados obtidos para taxa de utilização são muito próximos para todos os 4 controladores. Verifica-se também que todos os controladores conseguiram manter os requisitos de projeto do sistema de controle dentro daquilo que se esperava, como o tempo de assentamento e amplitude máxima. Além do que, os controladores mantiveram a taxa de utilização estacionária sobre a entrada de referência de 70% da taxa de utilização durante o regime estacionário, havendo uma pequena variação nos períodos transientes. Observa-se nitidamente que há três períodos distintos de regime transiente, o *warm-up*, a partir do instante 550 UTS e a partir de 1050 UTS. Embora esses períodos levem o sistema a um comportamento bem diferente daquele observado nos períodos de regime estacionário, todos os controladores mantiveram a atuação dentro das capacidades e, após alguns instantes, acabaram convergindo para o valor da entrada de referência no regime estacionário. Da mesma forma, a quantidade de máquinas virtuais foi muito parecida para todos os casos, como mostra a Figura 5.14.

A Figura 5.14 mostra claramente que a quantidade de máquinas virtuais para os quais o sistema converge é muito similar no regime estacionário. Em um primeiro momento, antes da inserção da primeira perturbação, os sistemas convergiram para um número médio de 20 máquinas virtuais após o período de aquecimento. O período de aquecimento apresentou a diferença mais perceptível na atuação de cada um dos controladores. Após a primeira perturbação há um salto no número de instâncias caracterizado pela presença de um *overshoot* de aproximadamente 20% (equivalente a 10 instâncias) muito semelhante em todos os casos e, após esse período transiente, estacionaram sobre uma quantidade média de 50 máquinas virtuais. Quando a segunda perturbação foi inserida, novamente, com um comportamento muito similar, todos os *brokers* voltaram a flutuar sobre uma média de 20 instâncias.

Figura 5.14 — Número de máquinas virtuais para $c = 0, 0.25, 0.50, 0.75$

Quanto maior for o valor do parâmetro c , mais lenta será a reação do controlador. Isso porque a resposta do sistema passa a depender mais das respostas anteriores do que da resposta atual e é necessário que uma mudança maior seja observada na resposta do sistema para que isso influencie a saída do transdutor. Os gráficos das Figuras 5.13 e 5.14 mostram de forma muito sutil como o *broker* atua mais lentamente às variações quando $c = 0.75$ do que para os demais valores observados na Tabela 5.3, no entanto, essas diferenças são mais fáceis de serem observadas na Figura 5.15 que ilustra o tempo médio de resposta.

É notável pela Figura 5.15 que o atraso para perceber a mudança na resposta do *broker* que utiliza o sistema de controle quando $c = 0.75$ gerou um impacto no tempo médio de resposta que durou até aproximadamente até o instante 600 UTS. Quanto menor a influência das respostas anteriores na saída medida, mais rápida foi a reação do controlador para que novas instâncias de máquinas virtuais fossem ligadas e então diminuíssem o tempo médio de resposta. Para o *broker* que não considerou o filtro de média móvel, $c = 0$, o valor médio foi alcançada a partir do instante 360 UTS, contra 400 UTS para o *broker* que considerou $c = 0.25$ e 450 UTS quando foi considerado $c = 0.5$. Embora as variações a partir do *warm-up* tenham sido intensas, após a primeira perturbação na taxa de chegada de requisições todos os controladores reagiram bem e não foi observado aumento no tempo médio de resposta. Analogamente, na segunda perturbação no intervalo entre as chegadas, há uma variação sutil no tempo médio de resposta, mas não há distanciamento suficiente para que se possa afirmar algo sobre o desempenho sentido. Novamente, após essa perturbação, o tempo médio de resposta convergiu para o valor médio e permaneceu estacionado.

Figura 5.15 — Tempo médio de resposta para $c = 0, 0.25, 0.50, 0.75$

Um aumento no tempo médio de resposta de grande intensidade e longa duração pode provocar um aumento na quantidade de requisições penalizadas e, conseqüentemente, diminuição no rendimento do consumidor. A ideia é que o *broker* seja capaz de reagir a essas variações e evitar que muitas requisições sejam penalizadas. A média da quantidade de requisições penalizadas, bem como um intervalo de confiança de 95% são mostrados pela Figura 5.16.

Figura 5.16 — Quantidade de requisições penalizadas para $c = 0, 0.25, 0.50, 0.75$

Percebe-se que, mesmo com o aumento do tempo médio de resposta quando $c = 0.75$, esse aumento não foi suficientemente grande no experimento para afirmar categoricamente que essa implementação teve piores resultados. Apesar de apresentar a maior média, a sobreposição dos intervalos de confiança mostra que essas informações são inconclusivos para determinar uma melhor ou pior implementação. Outro resultado interessante é que a média apresentada pelo *broker* que não considera a média móvel foi maior que aquelas apresentadas para os *brokers* que consideraram $c = 0.25$ e $c = 0.5$. Isso se deve ao fato de que, já que a saída medida não é suavizada, a reação do controlador é mais rápida que os demais, o que pode fazer com ele reaja além do necessário, ligando ou desligando mais máquinas que os demais, fazendo com que seja necessária uma correção *a posteriori*. Cabe ressaltar ainda que a quantidade média de requisições submetidas para cada *broker* foi de 4510 requisições e mesmo que não seja possível afirmar qual deles teve o melhor desempenho, pode-se considerar que todos tiveram bons resultados.

Uma verificação interessante é que apesar de todos os controladores utilizados terem seu comportamento ditado pelos polos dominantes p_1 e p_2 foram observadas pequenas variações na taxa de utilização, no número de máquinas virtuais e na quantidade de requisições penalizadas, e uma diferença mais aparente no tempo médio de resposta. Essas variações devem-se principalmente aos diferentes valores do polo p_3 que influenciam de maneira diferente o comportamento do sistema. Quanto mais próxima da magnitude do polos dominantes estiver a magnitude de p_3 , maior será a sua influência no comportamento do sistema.

Há casos também em que os critérios de controle desejados tornam inviável encontrar um valor para o terceiro polo. É possível verificar que com escolhas muito rígidas quanto ao comportamento do sistema de controle de malha fechada em regime transiente, o sistema de equações lineares 5.10 pode não ter solução. Isso acontece porque com o aumento da rigidez do comportamento que se espera do sistema, os polos dominantes tendem a aproximar-se da origem no plano complexo, e como a magnitude do terceiro polo não pode exceder a do polo dominante, o espaço possível para o terceiro polo fica muito restrito e, algumas vezes, acaba sendo inviável encontrar um intervalo do parâmetro de média móvel que respeite essas condições, tornando as escolhas impraticáveis. Por exemplo, optando por um tempo de assentamento de 1 intervalo de amostra, $K_s = 1$ e amplitude máxima de 10%, $M_p = 0.1$, a magnitude e o ângulo dos polos dominantes encontrados são dados pelas equações 5.17 e 5.18.

$$r = e^{-4/K_s}$$

$$r = e^{-4/1}$$

$$r = 0.018316 \tag{5.17}$$

$$\begin{aligned}
\theta &= \pi \frac{\log(r)}{\log(M_p)} \\
\theta &= \pi \frac{\log(0.018316)}{\log(0.1)} \\
\theta &= 5.4575
\end{aligned} \tag{5.18}$$

Tendo a magnitude e o ângulo é possível encontrar as coordenadas dos polos dominantes, como mostram as equações 5.19 e 5.20, respectivamente. Verifica-se que a magnitude desses polos é bem pequena e muito próxima de zero.

$$p_1 = 0.018316 e^{5.4575j} = 0.012419 + 0.013462j \tag{5.19}$$

$$p_1 = 0.018316 e^{-5.4575j} = 0.012419 - 0.013462j \tag{5.20}$$

Como os polos p_1 e p_2 são muito pequenos, para respeitar as escolhas de projeto p_3 deveria ser menor ainda. As restrições rígidas do projeto de controle fazem com que não existam valores viáveis para o parâmetro c , como pode ser observado em 5.21, e consequentemente não há como encontrar um terceiro polo e os parâmetros de controle, demonstrando que essas escolhas são impraticáveis.

$$c \in [2r \cos \theta - r - a - 1, 2r \cos \theta + r - a - 1] \cap [0, 1[$$

$$c \in [2(0.018316) \cos(5.4575) - 0.018316 - 0.57961 - 1, 2(0.018316) \cos(5.4575) + 0.018316 - 0.57961 - 1] \cap [0, 1[$$

$$c \in [-1.5731, -1.5365] \cap [0, 1[$$

$$c \in \emptyset \tag{5.21}$$

Parte V

Conclusão e Trabalhos Futuros

Conclusões e trabalhos futuros

Este capítulo traz as conclusões sobre os resultados e contribuições obtidos com os experimentos realizados e com o desenvolvimento deste projeto de pesquisa. Algumas conclusões são facilmente verificáveis por meio da análise dos gráficos de resultados, enquanto outras envolveram a experiência obtida ao longo da pesquisa.

O capítulo também faz um panorama sobre os rumos futuros deste projeto e aponta para a direção da continuidade, uma vez que, mesmo com a obtenção de resultados satisfatórios, o tema ainda permite que muito seja explorado e desenvolvido.

6.1 Conclusões

A computação em nuvem tem sido temática de destaque nos últimos anos tanto na indústria como na comunidade científica. Os provedores de nuvem disponibilizam uma nova maneira de comercializar capacidades computacionais na qual infraestrutura, plataforma e *software* são oferecidos aos seus consumidores como serviços. Além de versátil, esse novo modelo de comércio vem se firmando como um modelo muito atraente e custo-efetivo de adquirir recursos computacionais configuráveis amparados por um contrato de prestação de serviços.

Esse novo conceito traz consigo uma nova visão de capacidade computacional em que os recursos são elásticos, acessíveis de qualquer lugar, de forma ubíqua e sob demanda. Junto com a possibilidade de elasticidade dos recursos sob demanda, vem a necessidade de mecanismos automáticos que possam gerenciar o desempenho dos recursos instanciados de forma a garantir que contratos e parâmetros de *QoS* sejam obedecidos.

Este projeto de pesquisa foi ao encontro do desafio de buscar soluções adaptativas para esse novo modelo de aquisição de recursos e disponibilização de aplicações que é a computação em nuvem. O trabalho optou por aplicar técnicas de adaptação ainda pouco exploradas nas áreas da Computação, mas que são reconhecidas com sucesso há décadas por outros campos da ciência e engenharia, tendo como foco a aplicação para provisão dinâmica de recursos e gerenciamento de desempenho em ambientes de computação em nuvem. Em adição, esta pesquisa traz uma análise da aplicação da teoria clássica de controle em sistemas computacionais, listando características peculiares aos sistemas de computação que devem ser consideradas no projeto de controladores para esse domínio.

A teoria de controle é estabelecida na engenharia e algumas ciências naturais, e conta como uma rica coleção de ferramentas de linguagem matemática. Essas ferramentas auxiliam na descrição do comportamento de sistemas dinâmicos em termos de como eles respondem a diferentes estímulos em regime estacionário e transiente, e também na verificação de propriedades de interesse. A menos explorada aplicação de teoria de controle em computação tem como resultado o menor desenvolvimento de métodos e técnicas específicos que consideram, na sua prática, as particularidades intrínsecas dos sistemas computacionais. Em busca do mesmo êxito da teoria clássica de controle em outras áreas de conhecimento este trabalho buscou direcionar técnicas que trouxessem os mesmos benefícios para a aplicação de controle realimentado em sistemas adaptativos voltados ao gerenciamento de recursos em ambientes de computação em nuvem.

Os resultados da presente pesquisa foram bastante satisfatórios, na medida em que respondem positivamente às hipóteses formuladas na introdução, sendo é possível enumerar uma série de contribuições diretas na resolução do problema proposto. Como contribuições diretas na solução do problema de provisão dinâmica e automática de recursos em ambientes de computação em nuvem, a principal contribuição é a arquitetura de *broker* de consumidor com gerenciamento de desempenho e com orientação a mercado que permite que soluções próprias à ambientes de computação em nuvem sejam desenvolvidas com a implementação dos módulos propostos pela mesma. Além disso, o desenvolvimento de um mecanismo realimentado que permite a provisão dinâmica de recursos com base na medida do desempenho medido consiste em uma importante contribuição, uma vez que pode ser generalizado para problemas de mesma natureza ou estendido para outros desafios da computação.

Como contribuições para área de pesquisa explorada, este trabalho promoveu o avanço da aplicação de teoria de controle clássica em sistemas computacionais enumerando uma série de práticas que podem guiar a identificação de um modelo que representa o sistema objetivo e projeto de um controlador para esse sistema. Especificamente, o método para projeto de um sistema de controle proporcional-integral de malha fechada com suavização da resposta do sistema por média móvel, desenvolvido neste trabalho, permite que mesmo aqueles que possuem pouco conhecimento da área de controle consigam projetar controladores eficientes sem que seja necessário conhecimento específico para tal. Esse método de controle de malha fechada foi resultado da investigação de vários métodos de controle diferentes e mostrou-se adequado para situações em que a variação na

carga de trabalho foi intensa, com reações que condiziam com a variação verificada em grande parte dos experimentos.

Outro resultado importante é a versatilidade das técnicas desenvolvidas que não se limitam a aplicação na arquitetura proposta cujo foco está principalmente no desempenho. Essas técnicas podem ser estendidas e aplicadas a modelos de provisão dinâmica de recursos elásticos cujo objetivo de regulação seja outro como, por exemplo, na construção de modelos de alocação dinâmica de recursos para aplicações em nuvem cientes de *QoS* e *SLA*, ou focados diretamente na otimização do rendimento.

Além disso, como consequência da validação da arquitetura e métodos propostos, outras contribuições complementares podem ser identificadas como a extensão da biblioteca de simulação original do CloudSim para permitir que submissão de carga de trabalho e a alocação dinâmica de recursos possam ser realizadas em tempo de simulação. Outra contribuição complementar é a implementação do protótipo da arquitetura que, uma vez não se restrito exclusivamente a solução do problema proposto, pode ser usado para adaptação da arquitetura à outros problemas que não o gerenciamento de desempenho.

Verificou-se ainda que a presença de outros processos estocásticos fora da ação do *broker* (e.g. o intervalo entre as chegadas e o tempo de processamento das requisições) quando abordados na modelagem não invalidaram a aproximação por um sistema linear invariante no tempo, considerando-se os valores médios, a ponto de impossibilitar o projeto que atenda às especificações de desempenho dadas.

Nesta pesquisa, uma importante constatação foi obtida com relação a avaliação de desempenho de um sistema em regime transiente. Os experimentos conduzidos mostraram que o comportamento do sistema no regime transiente também influencia no alcance ou não dos objetivos propostos e o simples descarte do comportamento do sistema nesse regime, durante a avaliação de desempenho meramente em regime estacionário, pode levar a interpretações incorretas do desempenho ao longo do ciclo de vida de uma aplicação. Os resultados também mostraram que, mesmo quando duas soluções apresentam comportamentos equivalentes em regime estacionário, a diferença em regime transiente pode ser decisiva na escolha da solução de melhor aderência às especificações, especialmente no caso de cargas sujeitas a intensas variações (perturbações).

A eficácia da arquitetura ficou demonstrada por meio da análise dos resultados do protótipo que mostram que os objetivos de desempenho propostos foram atingidos. A modelagem correta do sistema objetivo e o projeto ajustado de um controlador levaram a criação de um mecanismo realimentado que combinado com outros módulos com funções sensoriais criaram uma solução integrada e autônoma de gerenciamento de desempenho com alocação dinâmica de recursos elásticos. Quando testado, o protótipo conseguiu regular as variáveis de resposta do sistema objetivo dentro daquilo que se esperava com recursos suficientes.

O método de projeto de controlador desenvolvido também se mostrou efetivo visto que os controladores projetados com o método foram capazes de atender aos requisitos do projeto quando em situações de aumento e diminuição da demandas de recursos.

O gerenciamento de desempenho dos recursos computacionais alocados em uma infraestrutura física compartilhada com base num controlador realimentado mostrou-se suficiente para que cumprir com os parâmetros dos contratos entre as partes envolvidas, visto que a quantidade de requisições mal atendidas foi reduzida quando comparada ao total, mesmo sob variações abruptas das condições de submissão sem que uma quantidade maior que o necessário de máquinas virtuais fosse utilizada. Como consequência, foi verificado que, na maior parte dos experimentos, os requisitos de *QoS* foram respeitados. Mesmo sujeito às estocasticidades das camadas que gerenciam a infraestrutura do *data center*, o sistema com controlador realimentado obteve bons resultados.

Os resultados da aplicação da teoria de controle clássica na solução de problemas de gerenciamento de desempenho e provisão dinâmica de recursos resultou no esperado e mostrou-se uma ferramenta promissora a ser explorada na construção de sistemas computacionais autônômicos. Pode-se considerar que os resultados obtidos atingiram os objetivos na solução do problema proposto. Contudo, o aproveitamento amplo dessa ferramenta em outros problemas de sistemas computacionais requer a dedicação de pesquisas no desenvolvimento de novas técnicas e na adaptação de vários outros métodos para sistemas computacionais, o que chama a atenção para esforços sejam direcionados a esse campo de pesquisa.

6.2 Trabalhos futuros

Este trabalho constituiu um passo inicial na investigação de estratégias para aplicação das técnicas da teoria clássica de controle em sistemas computacionais, principalmente voltados a computação em nuvem. Todavia, ainda há um horizonte distante a ser perseguido até que essas técnicas sejam amplamente empregadas em computação, como é o caso de outras ciências e engenharias.

Apesar dos bons resultados, a aplicação de controle realimentado neste trabalho ficou restrita à aplicação da teoria clássica de controle que limita-se ao desenvolvimento de controladores regulatórios com uma única entrada e uma única saída. Esse tipo de controle obedece a uma entrada de referência fixa e tem o objetivo de diminuir o erro entre a referência e saída do sistema, caso exista. Entretanto, essa entrada de referência previamente estabelecida pode não ser a mais adequado o tempo todo. Da mesma forma, uma vez estimados os ganhos do controlador do sistema de malha fechada, os mesmos permanecem sempre constantes.

Espera-se que com o desenvolvimento de pesquisas em aplicação da teoria de controle em sistemas computacionais, essas técnicas possam ser extendidas, como uma evolução natural, para outros tipos de controle como, por exemplo, o controle ótimo, no qual o valor de entrada da referência é automaticamente ajustado para atender a um objetivo de otimização; controle adaptativo, no qual os ganhos do controlador podem ser reajustados dinamicamente a fim de aprimorar o desempenho em diferentes circunstâncias. Técnicas de controle moderno também podem ser exploradas

para que sistemas *MIMO* possam ser modelados e controlados adequadamente usando espaço de estados.

Embora a adaptação de técnicas conhecidas seja interessante numa primeira exploração, almeja-se o desenvolvimento de técnicas que sejam específicas ao problemas de computação, inclusive com a integração de abordagens e algoritmos que foram desenvolvidos em computação e outras áreas de conhecimento e que já são usados com sucesso em problemas de adaptação dinâmica, por exemplo, o uso de estratégias preditivas para determinar o tamanho mais adequado de um conjunto de máquinas virtuais previamente instanciadas e disponíveis.

A análise de desempenho em regime transiente também permite que informações importantes do sistema, como amplitude máxima e o tempo de assentamento sejam calculados e, com isso, é possível ter uma visão ampla do comportamento do sistema não só no regime estacionário. Entende-se que essas informações podem ser muito úteis no desenvolvimento de novas metodologia de planejamento de capacidade para políticas de controle de admissão em sistemas computacionais sob cargas de trabalho variáveis.

Parte VI

Referências

Referências

- ABDELZAHER, T. F.; STANKOVIC, J. A.; LU, C.; ZHANG, R.; LU, Y. Feedback performance control in software services. *IEEE Control Systems Magazine*, v. 23, n. June, p. 74–90, 2003.
- AGESEN, O.; GARTHWAITE, A.; SHELDON, J.; SUBRAHMANYAM, P. The evolution of an x86 virtual machine monitor. *ACM SIGOPS Operating Systems Review*, v. 44, n. 4, p. 3, 2010.
Disponível em: <http://portal.acm.org/citation.cfm?doid=1899928.1899930>
- AN, B.; LESSER, V. Automated negotiation with decommitment for dynamic resource allocation in cloud computing. *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems*, v. 1, p. 981–988, 2010.
Disponível em: <http://dl.acm.org/citation.cfm?id=1838338>
- ARMBRUST, M.; STOICA, I.; ZAHARIA, M.; FOX, A.; GRIFFITH, R.; JOSEPH, A. D.; KATZ, R.; KONWINSKI, A.; LEE, G.; PATTERSON, D.; RABKIN, A. A view of cloud computing. *Communications of the ACM*, v. 53, n. 4, p. 50, 2010.
Disponível em: <http://portal.acm.org/citation.cfm?doid=1721654.1721672>
- AUYOUNG, A.; CHUN, B. N.; SNOEREN, A. C.; VAHDAT, A. Resource Allocation in Federated Distributed Computing Infrastructures. *Proceedings of the 1st Workshop on Operating System and Architectural Support for the On-demand IT InfraStructure*, v. 1, p. 1–10, 2004.
- BADGER, L.; PATT-CORNER, R.; VOAS, J. DRAFT Cloud Computing Synopsis and Recommendations Recommendations of the National Institute of Standards and Technology. *NIST Special Publication*, v. 800, p. 146, 2011.
- BARHAM, P.; DRAGOVIC, B.; FRASER, K.; HAND, S.; HARRIS, T.; HO, A.; NEUGEBAUER, R.; PRATT, I.; WARFIELD, A. Xen and the Art of Virtualization Categories and Subject

- Descriptors. *ACM SIGOPS Operating Systems Review - SOSP '03*, v. 37, n. 5, p. 164–177, 2003.
- BÉGIN, M. E.; JONES, B.; CASEY, J.; LAURE, E.; GREY, F.; LOOMIS, C.; KUBLI, R. An EGEE comparative study: Grids and Clouds-evolution or revolution. *EGEE III project Report*, v. 30, 2008.
- BOGGIA, G.; CAMARDA, P.; GRIECO, L. A.; MASCOLO, S. Feedback-Based Control for Providing Real-Time Services With the 802.11e MAC. *IEEE/ACM Transactions on Networking*, v. 15, n. 2, p. 323–333, 2007.
Disponível em: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4154743>
- BOX, G. E. P.; JENKINS, G. M. *Times series analysis: forecasting and control*. Wiley, 734 p., 2011.
- BUYYA, R.; RANJAN, R.; CALHEIROS, R. N. Modeling and simulation of scalable Cloud computing environments and the CloudSim toolkit: Challenges and opportunities. In: *2009 International Conference on High Performance Computing & Simulation*, IEEE, 2009a, p. 1–11.
Disponível em: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5192685>
- BUYYA, R.; RANJAN, R.; CALHEIROS, R. N. InterCloud : Utility-Oriented Federation of Cloud Computing Environments for Scaling of. *ALGORITHMS AND ARCHITECTURES FOR PARALLEL PROCESSING*, v. 6081, p. 13–31, 2010.
- BUYYA, R.; YEO, C. S.; VENUGOPAL, S. Market-Oriented Cloud Computing: Vision, Hype, and Reality for Delivering IT Services as Computing Utilities. In: *2008 10th IEEE International Conference on High Performance Computing and Communications*, IEEE, 2008, p. 5–13.
Disponível em: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4637675>
- BUYYA, R.; YEO, C. S.; VENUGOPAL, S.; BROBERG, J.; BRANDIC, I. Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation Computer Systems*, v. 25, n. 6, p. 599–616, 2009b.
Disponível em: <http://linkinghub.elsevier.com/retrieve/pii/S01677339X08001957>
- CALHEIROS, R. N.; RANJAN, R.; BELOGLAZOV, A.; DE ROSE, C. A. F.; BUYYA, R. CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Software: Practice and Experience*, v. 41, n. 1, p. 23–50,

2011.

Disponível em: <http://dx.doi.org/10.1002/spe.995><http://doi.wiley.com/10.1002/spe.995>

CHAMPRASERT, P.; SUZUKI, J. SymbioticSphere : A Biologically-Inspired Autonomic Architecture for Self-Adaptive and Self-Healing Server Farms. *WOWMOM '06 Proceedings of the 2006 International Symposium on on World of Wireless, Mobile and Multimedia Networks*, p. 469 – 474, 2006.

CHAMPRASERT, P.; SUZUKI, J.; LEE, C. Exploring self-optimization and self-stabilization properties in bio-inspired autonomic cloud applications. *Concurrency and Computation: Practice and Experience*, v. 24, n. 9, p. 1015–1034, 2012.

Disponível em: <http://doi.wiley.com/10.1002/cpe.1906>

CHANDRA, A.; GONG, W.; SHENOY, P. Dynamic resource allocation for shared data centers using online measurements. In: *Proceedings of the 11th international conference on Quality of service*, Springer-Verlag, 2003, p. 381–398.

CHEN, Y.; DAS, A.; QIN, W.; SIVASUBRAMANIAM, A.; WANG, Q.; GAUTAM, N. Managing server energy and operational costs in hosting centers. *ACM SIGMETRICS Performance Evaluation Review*, v. 33, n. 1, p. 303, 2005.

Disponível em: <http://portal.acm.org/citation.cfm?doid=1071690.1064253>

DIAO, Y.; HELLERSTEIN, J. L.; PAREKH, S. Optimizing Quality of Service Using Fuzzy Control. *Lecture Notes in Computer Science*, v. 2506/2002, p. 42–53, 2002a.

DIAO, Y.; HELLERSTEIN, J. L.; PAREKH, S. Using fuzzy control to maximize profits in service level management. *IBM Systems Journal*, v. 41, n. 3, p. 403–420, 2002b.

Disponível em: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5386869>

EMEAKAROHA, V. C.; NETTO, M.; CALHEIROS, R. N.; BRANDIC, I.; BUYYA, R.; DE ROSE, C. Towards autonomic detection of SLA violations in Cloud infrastructures. *Future Generation Computer Systems*, v. 28, n. 7, p. 1017–1029, 2012.

Disponível em: <http://linkinghub.elsevier.com/retrieve/pii/S0167739X11002184>

FIGUEIREDO, R.; DINDA, P.; FORTES, J. A case for grid computing on virtual machines. In: *23rd International Conference on Distributed Computing Systems, 2003. Proceedings.*, IEEE, 2003, p. 550–559.

Disponível em: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1203506>

- FITO, J. O.; GOIRI, I.; GUITART, J. SLA-driven Elastic Cloud Hosting Provider. In: *Parallel, Distributed and Network-Based Processing (PDP), 2010 18th Euromicro International Conference on*, 2010, p. 111–118.
- FOSTER, I.; KESSELMAN, C. *The grid 2: Blueprint for a new computing infrastructure*. 2 ed. San Francisco: Morgan Kaufmann, 748 p., 2004.
- FOSTER, I.; ZHAO, Y.; RAICU, I.; LU, S. Cloud Computing and Grid Computing 360-Degree Compared. In: *2008 Grid Computing Environments Workshop*, IEEE, 2008, p. 1–10.
Disponível em: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4738445>
- HELLERSTEIN, J.; PAREKH, S. A first-principles approach to constructing transfer functions for admission control in computing systems. In: *Proceedings of the 41st IEEE Conference on Decision and Control*, 2002., IEEE, 2002, p. 2906–2912.
Disponível em: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1184291>
- HELLERSTEIN, J.; PAREKH, S.; GRIFFITH, R.; KAISER, G.; PHUNG, D. A control theory foundation for self-managing computing systems. *IEEE Journal on Selected Areas in Communications*, v. 23, n. 12, p. 2213–2222, 2005.
Disponível em: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1546093>
- HELLERSTEIN, J.; SINGHAL, S. Research challenges in control engineering of computing systems. *IEEE Transactions on Network and Service Management*, v. 6, n. 4, p. 206–211, 2009.
Disponível em: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5374029>
- HELLERSTEIN, J. L. Why Feedback Implementations Fail: The Importance of Systematic Testing. In: *Proceedings of the Fifth International Workshop on Feedback Control Implementation and Design in Computing Systems and Networks - FeBiD '10*, New York, New York, USA: ACM Press, 2010, p. 25–26.
Disponível em: <http://portal.acm.org/citation.cfm?doid=1791204.1791209>
- HELLERSTEIN, J. L.; DIAO, Y.; PAREKH, S.; TILBURY, D. M. *Feedback Control of Computing Systems*. 1 ed. Hoboken, NJ, USA: John Wiley & Sons, Inc., 429 p., 2004.
Disponível em: <http://doi.wiley.com/10.1002/047166880X>
- HORN, P. Autonomic computing: IBM's Perspective on the State of Information Technology. 2001.

- INOMATA, A.; MORIKAWA, T.; IKEBE, M.; OKAMOTO, Y.; NOGUCHI, S.; FUJIKAWA, K.; SUNAHARA, H.; RAHMAN, S. M. M. Proposal and Evaluation of a Dynamic Resource Allocation Method Based on the Load of VMs on IaaS. In: *2011 4th IFIP International Conference on New Technologies, Mobility and Security*, IEEE, 2011, p. 1–6.
Disponível em: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5720603>
- IQBAL, W.; DAILEY, M. N.; CARRERA, D. Black-box approach to capacity identification for multi-tier applications hosted on virtualized platforms. In: *2011 International Conference on Cloud and Service Computing*, IEEE, 2011, p. 111–117.
Disponível em: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6138506>http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6138506
- JIANG, F.; FRATER, M.; HU, J. A Bio-inspired Host-Based Multi-engine Detection System with Sequential Pattern Recognition. In: *2011 IEEE Ninth International Conference on Dependable, Autonomic and Secure Computing*, IEEE, 2011, p. 145–150.
Disponível em: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6118343>
- JIANG, W.-W.; CUI, H.-Y.; CHEN, J.-Y. A fuzzy modeling based dynamic resource allocation strategy in service grid. *The Journal of China Universities of Posts and Telecommunications*, v. 16, n. September, p. 108–113, 2009.
Disponível em: <http://linkinghub.elsevier.com/retrieve/pii/S1005888508603465>
- JURY, E. *Theory and Application of the z-Transform Method*. Wiley New York., 1964.
- KLEMS, M.; NIMIS, J.; TAI, S. Do Clouds Compute? A Framework for Estimating the Value of Cloud Computing. *Designing E-Business Systems. Markets, Services, and Networks*, v. 22, p. 110–123, 2009.
Disponível em: <http://www.springerlink.com/index/10.1007/978-3-642-01256-3>
- KUSIC, D.; KANDASAMY, N. Risk-aware limited lookahead control for dynamic resource provisioning in enterprise computing systems. *Cluster Computing*, v. 10, n. 4, p. 395–408, 2007.
Disponível em: <http://dx.doi.org/10.1007/s10586-007-0022-y>
- LAI, K.; RASMUSSEN, L.; ADAR, E.; ZHANG, L.; HUBERMAN, B. A. Tycoon: An implementation of a distributed, market-based resource allocation system. *Multiagent and Grid Systems*, v. 1, n. 3, p. 169–182, 2005.
Disponível em: <http://iospress.metapress.com/index/9DB9JL1N8TVL9U2K.pdf>

- LEVY, R.; NAGARAJARAO, J.; PACIFICI, G.; SPREITZER, A.; TANTAWI, A.; YOUSSEF, A. Performance management for cluster based Web services. In: *IFIP/IEEE Eighth International Symposium on Integrated Network Management, 2003.*, n. i, Kluwer Academic Publishers, 2003, p. 247–261.
Disponível em: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1194184>
- LEWIS, P. R.; MARROW, P.; YAO, X. Resource allocation in decentralised computational systems: an evolutionary market-based approach. *Autonomous Agents and Multi-Agent Systems*, v. 21, n. 2, p. 143–171, 2009.
Disponível em: <http://www.springerlink.com/index/10.1007/s10458-009-9113-x>
- LIM, H. C.; BABU, S.; CHASE, J. S.; PAREKH, S. S. Automated control in cloud computing. In: *Proceedings of the 1st workshop on Automated control for datacenters and clouds - ACDC '09*, New York, New York, USA: ACM Press, 2009, p. 13.
Disponível em: <http://dl.acm.org/citation.cfm?id=1555275http://portal.acm.org/citation.cfm?doid=1555271.1555275>
- LU, C.; ABDELZAHER, T. F.; STANKOVIC, J. J. A.; SON, S. S. H.; ABDELZABER, T. A feedback control approach for guaranteeing relative delays in Web servers. In: *Proceedings Seventh IEEE Real-Time Technology and Applications Symposium*, IEEE Comput. Soc, 2001, p. 51–62.
Disponível em: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=929865>
- LU, C.; LU, Y.; ABDELZAHER, T. F.; STANKOVIC, J. A.; SON, S. H.; MEMBER, S. Feedback Control Architecture and Design Methodology for Service Delay Guarantees in Web Servers. *Parallel and Distributed Systems, IEEE Transactions on*, v. 17, n. 9, p. 1014–1027, 2006.
- LU, C.; STANKOVIC, J.; TAO, G.; SON, S. Design and evaluation of a feedback control EDF scheduling algorithm. In: *Proceedings 20th IEEE Real-Time Systems Symposium (Cat. No.99CB37054)*, IEEE Comput. Soc, 1999, p. 56–67.
Disponível em: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=818828>
- LU, C.; STANKOVIC, J. A.; SON, S. H.; TAO, G. Feedback Control Real-Time Scheduling : Framework , Modeling , and Algorithms. *Real-Time Systems*, v. 23, p. 85–126, 2002.
- MEDINA, A.; LAKHINA, A.; MATTA, I.; BYERS, J. BRITE: an approach to universal topology generation. *MASCOTS 2001, Proceedings Ninth International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, p. 346–353, 2001.

- Disponível em: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=948886>
- MELL, P.; GRANCE, T. The NIST Definition of Cloud Computing (Draft). *NIST special publication*, v. 800, p. 145, 2011.
- MOURA E SILVA, L.; BUYYA, R. Parallel Programming Models and Paradigms. In: *High Performance Cluster Computing: Architectures and Systems Systems*, 1 ed, cap. 1, Upper Saddle River, NJ, USA: Prentice Hall, p. 4–27, 1999.
- NOBILE, P. N.; LOPES, R. R. F.; MORON, C. E.; TREVELIN, L. C. QoS Proxy Architecture for Real Time RPC with Traffic Prediction. In: *11th IEEE International Symposium on Distributed Simulation and Real-Time Applications (DS-RT'07)*, IEEE, 2007, p. 261–267.
- Disponível em: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4384556>
- OGATA, K. *Modern Control Engineering*. 5th ed. Prentice-Hall, 912 p., 2009.
- PADALA, P.; HOU, K.-Y.; SHIN, K. G.; ZHU, X.; UYSAL, M.; WANG, Z.; SINGHAL, S.; MERCHANT, A. Automated control of multiple virtualized resources. In: *Proceedings of the fourth ACM european conference on Computer systems - EuroSys '09*, New York, New York, USA: ACM Press, 2009, p. 13.
- Disponível em: <http://portal.acm.org/citation.cfm?doid=1519065.1519068>
- PADALA, P.; SHIN, K. G.; ZHU, X.; UYSAL, M.; WANG, Z.; SINGHAL, S.; MERCHANT, A.; SALEM, K. Adaptive control of virtualized resources in utility computing environments. *ACM SIGOPS Operating Systems Review*, v. 41, n. 3, p. 289, 2007.
- Disponível em: <http://portal.acm.org/citation.cfm?doid=1272998.1273026>
- PARASKEVOPOULOS, P. *Modern Control Engineering*. Marcel Dekker Inc, 752 p., 2001.
- PAREKH, S.; GANDHI, N.; HELLERSTEIN, J. Using control theory to achieve service level objectives in performance management. *Real-Time Systems*, v. 23, p. 127–141, 2002.
- Disponível em: <http://www.springerlink.com/index/NKQN08V9P482PKXH.pdf>
- QUIROZ, A.; KIM, H.; PARASHAR, M.; GNANASAMBANDAM, N.; SHARMA, N. Towards autonomic workload provisioning for enterprise Grids and clouds. In: *Grid Computing, 2009 10th IEEE/ACM International Conference on*, 2009, p. 50–57.
- RAJ, H.; SCHWAN, K. High performance and scalable I/O virtualization via self-virtualized devices. *High Performance Distributed Computing HPDC 07 (2007)*, p. 179, 2007.

Disponível em: <http://portal.acm.org/citation.cfm?doid=1272366.1272390><http://wise.ajou.ac.kr:8080/pdfs/pdf/6464.pdf>

SALEHIE, M.; TAHVILDARI, L. Self-adaptive software: Landscape and research challenges. *ACM Transactions on Autonomous and Adaptive Systems*, v. 4, n. 2, p. 1–42, 2009.

Disponível em: <http://portal.acm.org/citation.cfm?doid=1516533.1516538>

STANKOVIC, J.; ABDELZAHER, T.; MARLEY, M. Feedback control scheduling in distributed real-time systems. In: *Proceedings 22nd IEEE Real-Time Systems Symposium (RTSS 2001) (Cat. No.01PR1420)*, IEEE Comput. Soc, 2001, p. 59–70.

Disponível em: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=990596>

STANKOVIC, J.; SON, S. The case for feedback control real-time scheduling. In: *Proceedings of 11th Euromicro Conference on Real-Time Systems. Euromicro RTS'99*, IEEE Comput. Soc, 1999, p. 11–20.

Disponível em: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=777445>

TULLOCH, M. *Understanding Microsoft Virtualization Solutions*. 2 ed. Redmond, Washington: Microsoft Press, 480 p., 2010.

Disponível em: <http://download.microsoft.com/download/5/B/4/5B46A838-67BB-4F7C-92CB-EABCA285DFDD/693821ebook.pdf>

URGAONKAR, B.; SHENOY, P.; CHANDRA, A.; GOYAL, P. Dynamic Provisioning of Multi-tier Internet Applications. In: *Second International Conference on Autonomic Computing (ICAC'05)*, IEEE, 2005, p. 217–228.

Disponível em: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1498066>

VAQUERO, L. M.; RODERO-MERINO, L.; CACERES, J.; LINDNER, M. A break in the clouds. *ACM SIGCOMM Computer Communication Review*, v. 39, n. 1, p. 50, 2008.

Disponível em: <http://portal.acm.org/citation.cfm?doid=1496091.1496100>

WANG, X.; CHEN, M. Cluster-level feedback power control for performance optimization. In: *2008 IEEE 14th International Symposium on High Performance Computer Architecture*, IEEE, 2008, p. 101–110.

Disponível em: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4658631>

- WANG, Z.; ZHU, X.; SINGHAL, S. Utilization and SLO-Based Control for Dynamic Sizing of Resource Partitions 2 A Case Study Using a Feedback Control Approach. *Lecture Notes in Computer Science*, v. 3775/2005, n. Ambient Networks, p. 133–144, 2005.
- WOLSKI, R. Analyzing Market-Based Resource Allocation Strategies for the Computational Grid. *International Journal of High Performance Computing Applications*, v. 15, n. 3, p. 258–281, 2001.
Disponível em: <http://hpc.sagepub.com/cgi/doi/10.1177/109434200101500305>
- WU, L.; KUMAR GARG, S.; BUYYA, R. SLA-based admission control for a Software-as-a-Service provider in Cloud computing environments. *Journal of Computer and System Sciences*, v. 78, n. 5, p. 1280–1299, 2012.
Disponível em: <http://linkinghub.elsevier.com/retrieve/pii/S0022000011001590>
- XIONG, P.; CHI, Y.; ZHU, S.; MOON, H. J.; PU, C.; HACIGUMUS, H. Intelligent management of virtualized resources for database systems in cloud environment. In: *2011 IEEE 27th International Conference on Data Engineering*, IEEE, 2011, p. 87–98.
Disponível em: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5767928
<http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5767928>
- XIONG, P.; WANG, Z.; JUNG, G.; PU, C. Study on performance management and application behavior in virtualized environment. In: *2010 IEEE Network Operations and Management Symposium - NOMS 2010*, IEEE, 2010, p. 841–844.
Disponível em: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5488362>
- XU, J.; ZHAO, M.; FORTES, J.; CARPENTER, R.; YOUSIF, M. Autonomic resource management in virtualized data centers using fuzzy logic-based approaches. *Cluster Computing*, v. 11, n. 3, p. 213–227, 2008.
- YING, L.; ABDELZAHER, T.; LU, C.; SHA, L.; LIU, X. Feedback control with queueing-theoretic prediction for relative delay guarantees in web servers. In: *The 9th IEEE Real-Time and Embedded Technology and Applications Symposium, 2003. Proceedings.*, IEEE, 2003, p. 208–217.
Disponível em: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1203053>
- ZHANG, J.; KIM, J.; YOUSIF, M.; CARPENTER, R.; FIGUEIREDO, R. J. System-level performance phase characterization for on-demand resource provisioning. *2007 IEEE International*

Conference on Cluster Computing, v. 0, p. 434–439, 2007.

Disponível em: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4629261>

ZHANG, Q.; CHENG, L.; BOUTABA, R. Cloud computing: state-of-the-art and research challenges. *Journal of Internet Services and Applications*, v. 1, n. 1, p. 7–18, 2010.

Disponível em: <http://www.springerlink.com/index/10.1007/s13174-010-0007-6>

ZHANG, R.; LU, C.; ABDELZAHER, T.; STANKOVIC, J. ControlWare: a middleware architecture for feedback control of software performance. In: *Proceedings 22nd International Conference on Distributed Computing Systems*, IEEE Comput. Soc, 2002, p. 301–310.

Disponível em: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1022267>

ZHANG, Z.; WANG, H.; XIAO, L.; RUAN, L. A statistical based resource allocation scheme in cloud. In: *2011 International Conference on Cloud and Service Computing*, IEEE, 2011, p. 266–273.

Disponível em: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6138531>

ZHU, Q.; AGRAWAL, G. Resource Provisioning with Budget Constraints for Adaptive Applications in Cloud Environments. *IEEE Transactions on Services Computing*, v. PP, n. 99, p. 1–13, 2012.

Disponível em: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6122011>

Parte VII

Anexos

Dados utilizados para modelagem do sistema

Tabela A.1 — Dados usados na simulação.

Intervalo de amostra (k)	Entrada $u(k)$	Saída $y(k)$
-49	26	0.13438
-48	26	0.24062
-47	26	0.32812
-46	26	0.38125
-45	26	0.41563
-44	26	0.4375
-43	26	0.45625
-42	26	0.375
-41	26	0.42067
-40	26	0.42067
-39	26	0.45192
-38	26	0.44471
-37	26	0.45913
-36	26	0.44712
-35	26	0.45673
-34	26	0.47356
-33	26	0.47115

Continua na próxima página

Intervalo de amostra (k)	Entrada $u(k)$	Saída $y(k)$
-32	26	0.49519
-31	26	0.50962
-30	26	0.52885
-29	26	0.53125
-28	26	0.52644
-27	26	0.53125
-26	26	0.51202
-25	26	0.51923
-24	26	0.53365
-23	26	0.52404
-22	26	0.4976
-21	26	0.51923
-20	26	0.52644
-19	26	0.51683
-18	26	0.4976
-17	26	0.51442
-16	26	0.53846
-15	26	0.51683
-14	26	0.48077
-13	26	0.48798
-12	26	0.52885
-11	26	0.55048
-10	26	0.54327
-9	26	0.53846
-8	26	0.52163
-7	26	0.52163
-6	26	0.53846
-5	26	0.54327
-4	26	0.5625
-3	26	0.57692
-2	26	0.60577
-1	26	0.58894
0	14	0.53846
1	14	0.89618
2	14	0.89504
3	14	0.89315
4	14	0.88036

Continua na próxima página

Intervalo de amostra (k)	Entrada $u(k)$	Saída $y(k)$
5	14	0.89792
6	14	0.91071
7	14	0.93363
8	14	0.9375
9	14	0.94643
10	14	0.93304
11	14	0.91071
12	14	0.94196
13	14	0.94643
14	14	0.95536
15	14	0.9375
16	14	0.96429
17	14	0.9375
18	14	0.92857
19	14	0.92857
20	14	0.94196
21	14	0.95089
22	14	0.9375
23	14	0.94643
24	14	0.95536
25	14	0.9375
26	14	0.94643
27	14	0.94643
28	14	0.93304
29	14	0.92411
30	14	0.94196
31	14	0.96429
32	14	0.95089
33	14	0.94643
34	14	0.9375
35	14	0.92857
36	14	0.95982
37	14	0.98661
38	14	0.96875
39	14	0.95089
40	14	0.96429
41	14	0.95982

Continua na próxima página

Intervalo de amostra (k)	Entrada $u(k)$	Saída $y(k)$
42	14	0.97768
43	14	0.95536
44	14	0.96429
45	14	0.94196
46	14	0.96875
47	14	0.96429
48	14	0.95536
49	14	0.96875
50	14	0.96875
51	14	0.98214
52	14	0.95536
53	14	0.95982
54	14	0.95982
55	14	0.95536
56	14	0.96429
57	14	0.94643
58	14	0.95536
59	14	0.95536
60	14	0.96429
61	14	0.96875
62	14	0.95536
63	14	0.9375
64	14	0.95982
65	14	0.97321
66	14	0.98214
67	14	0.97768
68	14	0.98214
69	14	0.97768
70	14	0.98661
71	14	0.98661
72	14	0.98661
73	14	0.97321
74	14	0.98661
75	14	0.97768
76	14	0.96429
77	14	0.96875
78	14	0.96875

Continua na próxima página

Intervalo de amostra (k)	Entrada $u(k)$	Saída $y(k)$
79	14	0.97768
80	14	0.97768
81	14	0.96429
82	14	0.96875
83	14	0.96429
84	14	0.96875
85	14	0.95536
86	14	0.98214
87	14	0.98214
88	14	0.97321
89	14	0.97321
90	14	0.97321
91	14	0.96429
92	14	0.97321
93	14	0.98661
94	14	0.97321
95	14	0.97321
96	14	0.96875
97	14	0.93304
98	14	0.95536
99	14	0.96875
100	14	0.97321
101	14	0.99554
102	14	0.97768
103	14	0.98661
104	14	0.95536
105	14	0.97321
106	14	0.98214
107	14	0.98214
108	14	0.99554
109	14	0.98214
110	14	0.98661
111	14	0.97321
112	14	0.97768
113	14	0.97321
114	14	0.99107
115	14	0.98214

Continua na próxima página

Intervalo de amostra (k)	Entrada $u(k)$	Saída $y(k)$
116	14	0.96875
117	14	0.96875
118	14	0.97768
119	14	0.99107
120	14	0.99554
121	14	0.98214
122	14	1
123	14	0.98214
124	14	0.98661
125	14	0.99107
126	14	0.97768
127	14	0.99554
128	14	0.98214
129	14	0.98661
130	14	0.98661
131	14	0.98661
132	14	0.98214
133	14	0.99107
134	14	0.98661
135	14	0.98661
136	14	0.99554
137	14	0.98661
138	14	0.97321
139	14	0.97768
140	14	0.99554
141	14	0.98661
142	14	0.96875
143	14	0.97321
144	14	0.97321
145	14	0.97321
146	14	0.96429
147	14	0.97321
148	14	0.98214
149	14	0.98214