Metodologia de benchmark para avaliação de
desempenho não-estacionária: um estudo de caso
baseado em aplicações de computação em nuvem

Edwin Luis Choquehuanca Mamani

SERVIÇO DE PÓS-GRADUAÇÃO DO ICMC-USP

Data de Depósito:

Assinatura: _____

Edwin Luis Choquehuanca Mamani

**Metodologia de benchmark para avaliação de desempenho
não-estacionária: um estudo de caso baseado em
aplicações de computação em nuvem**

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. *VERSÃO REVISADA*

Área de Concentração: Ciências de Computação e
Matemática Computacional

Orientador: Prof. Dr. Francisco José Monaco

USP – São Carlos
Abril de 2016

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

M263m

Mamani, Edwin Luis Choquehuanca

Metodologia de benchmark para avaliação de
desempenho não-estacionária: um estudo de caso
baseado em aplicações de computação em nuvem /
Edwin Luis Choquehuanca Mamani; orientador Francisco
José Monaco. - São Carlos - SP, 2016.

126 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, 2016.

1. Benchmark. 2. avaliação de desempenho.
3. análise transiente. 4. computação em nuvem. I.
Monaco, Francisco José, orient. II. Título.

Edwin Luis Choquehuanca Mamani

**Benchmark methodology for non-stationary performance
evaluation: a case study based on cloud computing
applications**

Doctoral dissertation submitted to the Instituto de
Ciências Matemáticas e de Computação – ICMC-
USP, in partial fulfillment of the requirements for the
degree of the Doctorate Program in Computer Science
and Computational Mathematics. *FINAL VERSION*

Concentration Area: Computer Science and
Computational Mathematics

Advisor: Prof. Dr. Francisco José Monaco

USP – São Carlos
April 2016

AGRADECIMENTOS

Os agradecimentos principais são direcionados a Deus. Sempre está ao meu lado.

A Universidade de São Paulo (USP) - Campus de São Carlos, pela oportunidade de realizar o curso de Pós-Graduação. Aos meus pais, Eduarda e José, em especial a minha mãe pelo apoio incondicional em todo momento.

Ao meu orientador e amigo, Francisco José Monaco, pela dedicação ao me orientar e por todo o apoio. A todos os colegas e amigos do LaSDPC,

Aos funcionários do ICMC-USP, em especial aos funcionários da secretária da Pós-Graduação, pelo convívio amigável e descontraído, e pela disposição em sempre atender bem.

A CAPES (Coordenação de Aperfeiçoamento de Pessoal de Nível Superior), pelo apoio financeiro.

“Obrigado Deus, aprendi.”

RESUMO

MAMANI, E. L. C.. **Metodologia de benchmark para avaliação de desempenho não-estacionária: um estudo de caso baseado em aplicações de computação em nuvem.** 2016. 126 f. Tese (Doutorado em Ciências – Ciências de Computação e Matemática Computacional) – Instituto de Ciências Matemáticas e de Computação (ICMC/USP), São Carlos – SP.

Este trabalho analisa os efeitos das propriedades dinâmicas de sistemas distribuídos de larga escala e seu impacto no desempenho, e introduz uma abordagem para o planejamento de experimentos de referência capazes de expor essa influência. Especialmente em aplicações complexas com múltiplas camadas de software (*multi-tier*), o efeito total de pequenos atrasos introduzidos por *buffers*, latência na comunicação e alocação de recursos, pode resultar gerando inércia significativa ao longo do funcionamento do sistema. A fim de detetar estas propriedade dinâmica, o experimento de execução do *benchmark* deve excitar o sistema com carga de trabalho não-estacionária sob um ambiente controlado. A presente pesquisa discorre sobre os elementos essenciais para este fim e ilustra a abordagem de desenvolvimento com um estudo de caso. O trabalho também descreve como a metodologia de instrumentação pode ser explorada em abordagens de modelagem dinâmica para sobrecargas transientes devido a distúrbios na carga de trabalho.

Palavras-chave: Benchmark, avaliação de desempenho, análise transiente, computação em nuvem.

ABSTRACT

MAMANI, E. L. C.. **Metodologia de benchmark para avaliação de desempenho não-estacionária: um estudo de caso baseado em aplicações de computação em nuvem.** 2016. 126 f. Tese (Doutorado em Ciências – Ciências de Computação e Matemática Computacional) – Instituto de Ciências Matemáticas e de Computação (ICMC/USP), São Carlos – SP.

This work examines the effects of dynamic properties of large-scale distributed systems and their impact on the delivered performance, and introduces an approach to the design of benchmark experiments capable of exposing this influence. Specially in complex, multi-tier applications, the net effect of small delays introduced by buffers, communication latency and resource instantiation, may result in significant inertia along the input-output path. In order to bring out these dynamic property, the benchmark experiment should excite the system with non-stationary workload under controlled conditions. The present research report elaborates on the essentials for this purpose and illustrates the design approach through a case study. The work also outlines how the instrumentation methodology can be exploited in dynamic modeling approaches to model transient overloads due to workload disturbances.

Key-words: Benchmark, performance evaluation, transient analysis, cloud computing.

LISTA DE ILUSTRAÇÕES

Figura 1 – Fluxo dos trabalhos relacionados.	29
Figura 2 – Arquitetura conceitual da estratégia DES não-estacionária.	32
Figura 3 – Arquitetura conceitual da estratégia DES não-estacionária.	36
Figura 4 – Análise transitória de um sistema IaaS.	39
Figura 5 – Estratégia de controle de retroalimentado dinâmico.	40
Figura 6 – Controle <i>feedback</i> do IBM Lotus Note TM	42
Figura 7 – Arquitetura da computação em nuvem.	48
Figura 8 – Modelo de referência da Computação em Nuvem.	49
Figura 9 – Modelo de uma nuvem privada.	51
Figura 10 – Modelo de uma nuvem privada fora da organização.	51
Figura 11 – Modelo de uma nuvem pública.	52
Figura 12 – Hipervisor <i>bare-metal</i>	54
Figura 13 – Hipervisor <i>hosted</i>	55
Figura 14 – Hipervisor monolítico.	55
Figura 15 – Hipervisor microkernel.	57
Figura 16 – Níveis de execução na arquitetura x86.	58
Figura 17 – Tradução binária.	59
Figura 18 – <i>Hypercalls</i> na paravirtualização.	59
Figura 19 – Esquema da virtualização assistida por <i>hardware</i>	60
Figura 20 – Arquitetura distribuída do Bench4Q.	62
Figura 21 – Interface gráfica do módulo Console usado pelo Bench4Q.	64
Figura 22 – Exemplo de CBMG.	65
Figura 23 – Geração de sessões definidas no TPC-W.	66
Figura 24 – Extensão do Bench4Q para o TPC-W na geração de sessões.	67
Figura 25 – Entidades do banco de dados, relacionamentos e características.	68
Figura 26 – Parâmetros de desempenho dinâmico.	70
Figura 27 – Intervalo linear na aproximação LTI.	72
Figura 28 – Representação no domínio do tempo.	72
Figura 29 – Representação no domínio da frequência.	73
Figura 30 – Arquitetura DES não-estacionária	75
Figura 31 – Uma malha fechada de controle integral.	81
Figura 32 – Diagrama de estados que representa o ciclo de vida de uma máquina virtual.	83
Figura 33 – OnlineBench4Q, extensão do <i>benchmark</i> Bench4Q.	84

Figura 34 – Casos de uso que representa a interação do desenvolvedor ou usuário do <i>benchmark</i> com os aspectos definidos pela MEDC.	88
Figura 35 – Modelo dos módulos no relacionados com o efetor.	88
Figura 36 – Esquema geral do modelo de demanda implementado no <i>Node balancer</i> . . .	91
Figura 37 – Modelos de estados das máquinas virtuais definidos pelo Efetor.	92
Figura 38 – Diagrama de classes do <i>Node balancer</i>	94
Figura 39 – Diagrama de classes do <i>middleware</i> gerenciador do Hipervisor.	95
Figura 40 – Tempo de inicialização das máquinas virtuais em condições diferentes de carga no servidor.	101
Figura 41 – Tempo de resposta das requisições nos clientes utilizando o banco de dados PostgreSQL.	102
Figura 42 – Tempo de resposta das requisições nos clientes utilizando o banco de dados DB2.	103
Figura 43 – Arquitetura do OnlineBench4Q com um único banco de dados.	104
Figura 44 – Arquitetura do OnlineBench4Q com o <i>pool</i> de bancos de dados.	105
Figura 45 – Comportamento do controlador PI ante uma mudança no contrato de 45% de utilização para 58%.	106
Figura 46 – Dados experimentais e função de transferência gerada.	108
Figura 47 – Lmites na aplicação de carga da trabalho.	109

LISTA DE CÓDIGOS-FONTE

Código-fonte 1 – Estrutura simplificada do simulador usando o <i>framework</i> CloudSim. .	35
Código-fonte 2 – Comportamento original do DataCenterBroker com o OnlineBroker .	36

LISTA DE TABELAS

Tabela 1 – Características de <i>software</i> e <i>hardware</i> do <i>Node balancer</i>	98
Tabela 2 – Número de elementos no banco de dados por tabela.	99

LISTA DE ABREVIATURAS E SIGLAS

API		<i>Application program interface</i>
ARX		auto-regressive with exogenous input
CBMG	...	Customer Behavior Model
DC		<i>Direct current gain</i>
LTI		<i>Linear Time Invariant</i>
MEDC	...	<i>Model, Effector, Demand, Capacity</i>
PI		<i>Proporcional Integral</i>
RPC		<i>Remote Procedure Call</i>
SISO		Single Input, Single Output
SLA		Service Level Agreement
SUT		Software Under Test
TPC-W	...	Transaction Processing Performance Council-Web

SUMÁRIO

1	INTRODUÇÃO	25
1.1	Avaliação de desempenho não-estacionária	26
1.2	O modelo conceitual MEDC	27
1.3	Contexto da pesquisa	28
1.4	Objetivo	29
1.5	Estrutura do documento	30
2	REVISÃO	31
2.1	O modelo conceitual MEDC	31
2.2	MEDC aplicada na simulação	33
2.2.1	<i>A extensão OnlineCloudsim</i>	34
2.2.2	<i>Análise transitória com a MEDC-DES</i>	38
2.3	Benchmarking não-estacionário	41
2.3.1	<i>Abordagens tradicionais</i>	41
2.4	Computação em nuvem	44
2.4.1	<i>Características da computação em nuvem</i>	44
2.4.2	<i>Tecnologias</i>	45
2.4.2.1	<i>Data center</i>	46
2.4.3	<i>Arquitetura da computação em nuvem</i>	47
2.4.4	<i>Modelos de serviço</i>	47
2.4.5	<i>Modelos de implantação</i>	50
2.5	Virtualização de sistemas operacionais	53
2.5.1	<i>Hipervisor tipo 1</i>	53
2.5.2	<i>Hipervisor tipo 2</i>	54
2.5.3	<i>Hipervisor Monolítico</i>	54
2.5.4	<i>Hipervisor Microkernel</i>	56
2.5.5	<i>Arquitetura x86 na virtualização</i>	57
2.5.6	<i>Entrada e saída na virtualização</i>	60
2.6	Bench4Q	62
3	METODOLOGIA	69
3.1	Fundamentação teórica e prática	69
3.1.1	<i>Modelagem dinâmica</i>	71

3.2	Modelo de capacidade dinâmica	74
3.3	Análise de resultados e avaliação	76
3.4	Materiais e métodos	76
4	ONLINEBENCH4Q	79
4.1	Problema-exemplo	79
4.2	Instanciação do modelo MEDC no OnlineBench4Q	84
4.3	Efeto	85
4.3.1	<i>Estados das máquinas virtuais no Efeto</i>	85
4.4	Monitor	86
4.5	Capacidade e Demanda	87
4.5.1	<i>Modelo de capacidade</i>	87
4.5.1.1	<i>Desligar máquinas</i>	89
4.5.1.2	<i>Ligando máquinas</i>	89
4.5.2	<i>Modelo de demanda</i>	90
4.5.2.1	<i>Gerando a perturbação</i>	91
4.6	Protótipo do gerenciador da demanda e capacidade	92
4.6.1	<i>Node balancer</i>	93
4.6.2	<i>VM Manager</i>	94
5	RESULTADOS E DISCUSSÕES	97
5.1	<i>Hardware e software</i>	97
5.2	Tempo de inicialização de máquinas virtuais	99
5.3	Comparação de PostgreSQL e DB2	101
5.3.1	<i>PostgreSQL</i>	102
5.3.2	<i>DB2</i>	103
5.3.3	<i>Mudança no banco de dados</i>	104
5.4	Adaptação às mudanças internas	105
5.5	Análise do degrau	107
5.6	Dificuldades e Desafios	109
5.6.1	<i>Definição da taxa de requisições</i>	109
5.6.2	<i>Configuração das camadas de aplicação e banco de dados</i>	110
6	CONCLUSÕES	111
6.1	Conclusões	111
6.2	Publicações	112
	REFERÊNCIAS	115
	APÊNDICE A CONFIGURAÇÃO DO AMBIENTE	123

A.1	Configuração da camada de aplicação	123
A.1.1	<i>Configurações do WebSphere</i>	124
A.2	Instalação e configuração do banco de dados	124
A.2.1	<i>Criação do banco de dados no DB2</i>	125

INTRODUÇÃO

Em sistemas computacionais, a avaliação de desempenho é comumente destinada a responder perguntas tais como quão adequado é o tempo de resposta do serviço sob uma determinada carga de trabalho. Uma estimativa quantitativa pode ser obtida experimentalmente por meio da aplicação de uma entrada e aquisição das métricas de desempenho de interesse, tais como taxa de transferência de dados, tempo de resposta, taxa de falhas ou qualquer parâmetro significativo. Nessa análise da saída após a aplicação de uma entrada, muitas vezes faz sentido descartar o intervalo de tempo da inicialização do sistema, e limitar a observação para o período após sua estabilização. Um procedimento prático para determinar a capacidade do sistema em estado estacionário é aumentar gradualmente a carga de trabalho, esperar os efeitos transitórios desaparecerem, e seguidamente medir o desempenho entregue. Esse processo pode ser repetido aumentando gradualmente o nível da carga de trabalho até encontrar um determinado nível que faz com que a saída se degrade além de um limite especificado. Em conclusão, este protocolo experimental de análise revela a “capacidade estacionária do sistema”.

A análise de estado estacionário é tão predominante na avaliação de desempenho de sistemas computacionais que muitas vezes é tacitamente assumida, tanto pela literatura técnica quanto pela científica, sem a necessidade de distingui-la explicitamente da análise complementar de capacidade não estacionária, ou seja, a avaliação de desempenho durante o regime transitório. Uma razão para a predominância da avaliação de desempenho de estado estacionário é em uma ampla classe de problemas de engenharia, programas, plataformas de computador, etc. comumente respondem rápido o suficiente às alterações operacionais, tal que é viável assumir que tais distúrbios de carga de trabalho afetam a saída na mesma resolução de tempo daquela escala de tempo do processo em análise. Para fins práticos, alterações na carga de trabalho afetam o desempenho de maneira instantânea. Quando este é o caso, frequentemente não é importante medir como as transições de desempenho evoluem temporalmente.

1.1 Avaliação de desempenho não-estacionária

Uma contribuição deste trabalho é ressaltar as condições técnicas em que as suposições mencionadas acerca de sistemas que apresentam saídas com dinâmicas muito pequenas, não necessariamente permanecem válidas. Isto pode ser o caso, por exemplo, de ambientes computacionais distribuídos em larga escala e aplicativos complexos compostos de várias camadas de *software*, em que efeitos combinados de atrasos intrínsecos, ainda que pequenos, e sua propagação por todo o subsistema interligado, podem gerar um comportamento dinâmico significativo.

Preenchimento de *buffers*, alocação de recursos em tempo de execução, sincronização de processos concorrentes e atrasos na resposta de dispositivos de *hardware*, para citar alguns casos, podem contribuir para o surgimento de propriedades dinâmicas em sistemas computacionais modernos. Por exemplo, isso significa que um aumento súbito no número de requisições por segundo que chega para um grupo de computadores dentro de um *data center* pode ser percebido como um aumento do tempo de resposta de maneira não instantânea. Em vez disso, um comportamento inercial pode ser observado no tempo de resposta das requisições. A mudança do desempenho pode se apresentar como um processo gradual, partindo de um nível estacionário inicial, até atingir um novo estado estacionário final. No decurso deste regime transitório, dependendo das propriedades intrínsecas do sistema, o que detém sua dinâmica, o desempenho medido pode variar com o tempo de maneira abrupta ou suave, monotonicamente, ou mesmo apresentando aspectos oscilatórios.

Contrastando com a identificação do sistema de estado estacionário, enquanto uma formulação analítica para modelagem de capacidade estacionária prevê desempenho em função da carga de trabalho imposta. Um modelo analítico da capacidade de desempenho não-estacionárias resulta em uma função da carga de trabalho e do tempo. Isso está de acordo com a noção formal de um sistema dinâmico, que, por definição, mantém a memória das entradas anteriores, armazenadas como estados internos.

Esta distinção aparentemente sutil é porém relevante. O escopo da avaliação de desempenho estacionária tem como objetivo informar se o sistema é capaz de entregar um determinado desempenho ante uma carga de trabalho, enquanto a avaliação de desempenho não-estacionárias o estende, informando se o sistema pode lidar com determinada mudança na carga de trabalho. A capacidade dinâmica (ao contrário de estática) do sistema pode dizer, em um determinado instante, se um sistema que tem os recursos suficientes para servir uma determinada carga de trabalho estacionária, poderia ser sobrecarregado durante o regime transitório resultante de uma perturbação abrupta, ainda que de baixa na amplitude, mesmo que o nível final estacionário da saída do sistema esteja dentro dos limites da capacidade estática.

Enquanto as ferramentas disponíveis para avaliação de desempenho de sistemas computacionais oferecem os recursos básicos para projetar experimentos usando *benchmarks* com

diferentes níveis de flexibilidade (NGAN *et al.*, 2012; ZHANG *et al.*, 2011a; GARCÍA; GARCÍA, 2003), muitos *frameworks* amplamente conhecidos para essa finalidade foram concebidos originalmente para realização de análise no regime estacionário (CASANOVA, 2001; CALHEIROS *et al.*, 2011). Módulos de *software* básicos são expressivos para implementar processos estacionários e instrumentação básica é oferecida pelos *frameworks* para coletar as medidas globais agregadas, tais como médias e outras estatísticas. O projeto de um experimento de avaliação de desempenho não-estacionária é destinado a obter o modelo de capacidade dinâmica, possui necessidades complementares.

1.2 O modelo conceitual MEDC

Como uma forma de abordar os requisitos metodológicos essenciais da avaliação de desempenho não-estacionária em sistemas computacionais, o modelo de aspectos *Model, Effector, Demand, Capacity* (MEDC), descrito por Júnior *et al.* (2015). O modelo foi proposto como resultado da linha de pesquisa em andamento no LaSDPC, ICMC-USP, no contexto do qual o presente trabalho de pesquisa tem sido desenvolvido. O modelo de referência deduz quatro aspectos básicos que podem ser resumidos nos módulos: **Monitor, Efetor, Demanda e Capacidade**. A lógica subjacente desta abordagem pode ser resumida como segue. Suponha um experimento destinado a medir como uma mudança abrupta de carga de trabalho impacta no desempenho entregue pelo sistema avaliado. Como fundamento conceitual para a elaboração da modelagem do problema, a abordagem baseia-se na definição genérica de *desempenho* como uma medida de quão bem a *capacidade* do sistema pode lidar com uma determinada mudança na carga de trabalho ou demanda. Mais especificamente, neste trabalho, *desempenho* refere-se a qualquer parâmetro de interesse, como rendimento (*throughput*), tempo de resposta, taxa de utilização, frequência de falhas ou qualquer outra métrica; e *demand*, refere-se a qualquer esforço imposto ao sistema, tal como aumento na taxa de requisições ou variações nos tempos de processamento das tarefas.

Um protocolo experimental simples para expor a dinâmica da *capacidade* pode ser iniciar com uma carga de trabalho fixa e, após um intervalo especificado, elevar abruptamente a intensidade da carga de trabalho até um nível estacionário maior. Com a *capacidade* fixa, o desempenho é medido periodicamente. Um dos requisitos para a aplicação do protocolo é que o *benchmark* não deve se basear num processo em lotes em que toda *demand* (carga de trabalho) é gerada no início do experimento para ser consumida pelo sistema à taxa máxima. Em vez disso, o protocolo precisa aderir a um processo *on-line*, no qual, a carga de trabalho é apresentada ao sistema em quantidades e tempos específicos.

O Modelo de referência MEDC, no entanto, estende estas exigências com a capacidade da modulação da demanda, ou seja, a capacidade de controlar a proporção da demanda ao longo do experimento em função de outros parâmetros adicionais ao tempo, por exemplo, alguma

medida de desempenho. Isto permite, por exemplo, a concepção de um mecanismo de *feedback* para restringir a chegada de requisições em função do tempo de resposta, a fim de emular uma política de controle de admissão, ou então para modelar o “*think time*” do usuário em resposta à qualidade de serviço percebida. Da mesma forma, o MEDC propõe complementarmente a modulação de *capacidade* ou recursos do sistema em função de parâmetros de interesse, tais como a atual demanda submetida ou o desempenho entregue. Esta possibilita, por exemplo, a emulação de um mecanismo de alocação de recurso adaptativo que aumenta ou diminui a capacidade de processamento de acordo com uma estratégia adaptativa.

A fim de que a modulação de ambos, *capacidade* e *demanda* possa ser exercida, adicionalmente, o sistema deve ser dotado da capacidade de monitoramento em tempo real. Finalmente, o aspecto *efetor*, representa os mecanismos de atuação através dos quais modulações na *demanda* e *capacidade* têm efeito no sistema. Por exemplo, o *efetor* modela com atrasos padrões de falha e efeitos inerciais que influenciam prontamente como *demanda* e *capacidade* são alteradas em resposta às suas ordens específicas de modulação.

1.3 Contexto da pesquisa

O modelo de aspectos MEDC foi originalmente introduzido como uma arquitetura de exigências para simulação de eventos discretos de sistemas dinâmicos (MAMANI *et al.*, 2015), e orientou o projeto *OnlineBroker*, uma extensão do *framework* de simulação de computação em nuvem CloudSim (CALHEIROS *et al.*, 2011). No trabalho (MAMANI *et al.*, 2015) é relatada a extensão da classe *DataCenterBroker* do CloudSim, inicialmente desenvolvida para simulação *off-line*, ou em *batch*, para suportar simulação *on-line* sob o paradigma da modulação de *demanda* e *capacidade*.

A arquitetura de referência MEDC é, no entanto, um modelo abstrato de aspectos para a avaliação de desempenho não-estacionário, *não estritamente limitado a abordagens de simulação*. No presente trabalho de pesquisa, são explorados e discutidos os principais conceitos da MEDC, e como esses podem ser aplicáveis também para abordagens de *benchmarking*. Isso é ilustrado através da utilização de princípios arquiteturais para estender um *benchmark* existente para computação em nuvem a fim de adaptá-lo às necessidades de avaliação de desempenho não-estacionárias.

O presente trabalho está inserido em uma linha de pesquisa em modelagem e avaliação de desempenho de sistemas computacionais, junto ao Laboratório de Sistemas Distribuídos e Programação Concorrente — LaSDPC, e contribui com outros trabalhos relacionados ao tema. A Figura 1 ilustra esquematicamente a interação entre estas pesquisas. Na teses de doutorado de (NOBILE, 2013), foi apresentada uma proposta de avaliação de desempenho não-estacionária baseada na teoria de controle clássica para o gerenciamento recursos (máquinas virtuais) dentro de um *datacenter*. Luz (2014) apresenta um método empírico para previsão de sobrecargas no

regime não-estacionário por meio de modelagem dinâmica.

Figura 1 – Fluxo dos trabalhos relacionados.

Fonte: Elaborada pelo autor.

Produto das pesquisas anteriores e dos trabalhos relacionados com avaliação de desempenho (MAMANI *et al.*, 2015), foi proposto um modelo conceitual de avaliação de desempenho chamado MEDC (*Monitor, Efecter, Demand, Capacity*) (PEREIRA *et al.*, 2015), que define e mapeia as responsabilidades dentro dos sistemas computacionais visando uma análise não-estacionária. O modelo conceitual foi validado utilizando o *framework* de simulação de computação em nuvem CloudSim (CALHEIROS *et al.*, 2011). A metodologia proposta pelo MEDC foi contextualizada inicialmente para o entorno de simulação, mas sistemas de computação em nuvem tipicamente apresentam complexidades que não chegam a ser consideradas pela simulação, em consequência é de interesse explorar as limitações do modelo para ambiente mais realísticos. Com esse intuito o presente trabalho utiliza os resultados obtidos dos trabalhos realizados na simulação para um ambiente realístico utilizando para isso os sistemas de *benchmarks*. A metodologia proposta foca na construção de *benchmarks*, considerando as responsabilidades que especifica o modelo conceitual MEDC nos *benchmarks* de computação em nuvem.

1.4 Objetivo

Este trabalho de pesquisa tem como objetivo formular uma metodologia para projeto de *benchmarks* para avaliação de desempenho não-estacionária de sistemas computacionais baseado no modelo de aspectos MEDC. Uma contribuição do trabalho é demonstrar que a metodologia

abstrata MEDC proposta originalmente para a simulação, é aplicável também a problemas de *benchmarks*, bem como ela pode ser instanciada nesse outro domínio.

1.5 Estrutura do documento

Este documento apresenta a seguinte organização:

- O [Capítulo 2](#) aborda os conceitos básicos referentes aos modelos utilizados para aplicar a metodologia num contexto de *benchmark*;
- No [Capítulo 3](#) são apresentadas informações relativas à fundamentação teórica da metodologia relacionada com avaliação de desempenho não-estacionária e computação em nuvem;
- O [Capítulo 4](#) apresenta o estudo de caso relacionado à computação em nuvem com a explanação do uso da metodologia na construção de *benchmarks*;
- No [Capítulo 5](#) são apresentados os resultados obtidos por meio dos experimentos realizados com a aplicação da metodologia;
- No [Capítulo 6](#) são apresentadas as conclusões e principais contribuições deste trabalho, bem como, algumas sugestões para trabalhos futuros.
- Finalmente são apresentadas as Referências bibliográficas.

REVISÃO

A simulação de eventos discretos não-estacionária não é um assunto inteiramente novo, mas também é verdade que este não é um problema amplamente explorado em sistemas computacionais. [Yang e Liu \(2010\)](#) dissertaram sobre este assunto e sobre abordagens matemáticas para análise transiente de redes de filas. [Schriber, Brunner e Smith \(2014\)](#), juntamente com [Choi e Kang \(2014\)](#), fornecem uma abrangente revisão relacionada à simulação de eventos discretos, enquanto [Kaczynski, Leemis e Drew \(2012\)](#) fornecem uma discussão aprofundada sobre os desafios deste campo. Enquanto os trabalhos já citados concentram-se em metodologias analítico-dedutivas, [Coevery \(1995\)](#) explora algumas investigações sobre técnicas empírico-dedutivas, que também é a abordagem do presente trabalho. Relacionado com os aspectos metodológicos, em grande parte, os resultados relatados na literatura muitas vezes dependem de abordagens *ad-hoc* para modelagem dinâmica de sistemas computacionais. Embora trabalhos como de [\(HUANG *et al.*, 2013\)](#) contribuam para o campo do ponto de vista teórico, técnicas disponíveis não raramente são específicas para o âmbito da aplicação e não generalizáveis.

Nessa perspectiva, o trabalho desenvolvido por [Pereira *et al.* \(2015\)](#), contribui com o estado da arte por meio da formulação de uma metodologia sistematizada que aborda o problema do ponto de vista orientado à aplicação em vez do teórico-conceitual. O trabalho citado apresenta uma abordagem de separação de aspectos para simulação transitória de eventos discretos e formaliza um modelo abstrato de referência para a concepção de ferramentas de simulação.

2.1 O modelo conceitual MEDC

Com respeito à análise estacionária de sistemas computacionais, as exigências da avaliação de desempenho de *frameworks* convencionais são abordadas geralmente por processamento em lote (*batch*). Essa é a abordagem padrão exposta por muitas das ferramentas comumente utilizadas. No *workflow* básico, o usuário pré-configura o conjunto de recursos do sistema e as unidades básicas de carga de trabalho, e então permite que o motor de simulação consuma a

demanda com a capacidade disponível segundo um processamento em lote. Não há nenhuma provisão que permita alterar, em tempo real, a quantidade de requisições ou o poder de processamento disponível, previamente definido, durante a execução da simulação, a fim de especificar como a demanda e capacidade mudam ao longo do tempo. A possibilidade de mudar esses parâmetros é, no entanto importante para análise de sistemas dinâmicos, o que implica, que um requisito para a simulação não-estacionária é a passagem do paradigma de simulação em lote para a simulação *on-line*.

Em vez de um conjunto fixo de processadores consumindo um conjunto pré-configurado de requisições, a simulação *on-line* deve permitir a variação na geração de demanda e o redimensionamento da capacidade durante a execução do experimento, bem como monitorar como o desempenho é afetado ao longo do tempo.

Formalmente, essa característica pode ser definida como a capacidade de *modular* tanto a demanda (requisições) quanto a capacidade (recursos). Por exemplo, em vez de atribuir um número fixo de processadores e requisições durante a configuração do experimento, a exigência traduz-se na configuração de regras para controlar estes valores conforme a simulação é executada, quer como uma função do tempo, quer de outras variáveis independentes, por exemplo o tempo de resposta medido, a taxa de utilização, etc.

Como uma forma para abordar esse requisito, a estratégia adotada nesta pesquisa baseia-se sobre a arquitetura de referência conceitual MEDC (JÚNIOR *et al.*, 2015), representada na Figura 2.

Fonte: Adaptada de Júnior *et al.* (2015).

Figura 2 – Arquitetura conceitual da estratégia DES não-estacionária.

O diagrama diferencia a abordagem de avaliação de desempenho *on-line* em quatro aspectos essenciais: **Demanda**, **Capacidade**, **Monitor** e **Efetor**. A primeira é uma referência à capacidade de modulação da *demanda* configurável pelo projetista da simulação a fim de especificar como a carga de trabalho muda ao longo do tempo. A segunda é a *capacidade* de modular os recursos do sistema ao longo do tempo. O *monitor* desempenha o papel de aquisição

de dados temporais, sendo seu dever recolher dados do sistema e torná-los disponíveis para os outros módulos. Finalmente, o *efetor* representa os mecanismos de atuação pelos quais a *demand*a e a *capacidade* mudam os estados internos do sistema. O *efetor* também serve como uma camada de abstração extra disponível para o desenvolvedor modelar operações e componentes que afetam a dinâmica do sistema. No fluxo básico para implementar o modelo de referência, a responsabilidade associada com o aspecto *demand*a é controlar a geração de carga no decorrer do experimento— e do mesmo modo, no que diz respeito do aspecto *capacidade*, para com os recursos do sistema.

Essa ação de controle se traduz em uma política de tomada de decisão que desencadeiam eventos solicitando o aumento da criação ou a demissão de novas entidades de carga de trabalho e recursos. A ação final é executada pelo *efetor*, que recebe ordens dos controladores de *demand*a e *capacidade* e manipula-os antes de passar os eventos para o motor do *framework*. É por meio da entidade que o desenvolvedor pode modelar, por exemplo, atrasos associados à alocação e desalocação de recursos, de políticas de agendamento *on-line*, utilização de *buffers*, os tempos de resposta de dispositivos de *hardware*, padrões de falha de recursos etc. O dever do monitor é alimentar os controladores de *demand*a e *capacidade* com dados atualizados sobre tudo que precise ser atualizado, por exemplo, utilização média do sistema, tempo de permanência dos *jobs*, *deadline*, etc.

2.2 MEDC aplicada na simulação

Em virtude da natureza discreta intrínseca dos processos computacionais, a *simulação de eventos discretos* (DES) desempenha um papel importante na avaliação de desempenho dos sistemas computacionais. Princípios essenciais das técnicas de simulação de eventos discretos podem ser associados a modelos de rede de filas. Através deste arcabouço conceitual, os sistemas são descritos em termos de filas, onde os *jobs* que chegam esperam até serem processados e são posteriormente liberados. Enquanto alguns sistemas de fila única podem ser abordados na perspectiva matemática da teoria de filas, por meio de métodos analíticos Markovianos, redes de filas mais complexas. A simulação é uma alternativa comumente utilizada (SCHRIBER; BRUNNER; SMITH, 2014; CHOI; KANG, 2014).

Enquanto *frameworks* de simulação de eventos discretos disponíveis fornecem os módulos para a concepção de programas de simulação com características diferentes, a API padrão oferecida pelos principais *frameworks* (CALHEIROS *et al.*, 2011) de avaliação de desempenho de sistemas computacionais são orientadas para a simulação estacionária, no sentido de que as abstrações básicas são expressivas para descrever processos estacionários e funcionalidades estão prontamente disponíveis para coletar as medidas globais e outras estatísticas. Projetar um programa de simulação para análise de desempenho transiente tende a requer consideravelmente mais esforço. Isto levou à concepção de um modelo de estrutura DES *on-line* com base nos

requisitos de MEDC (MAMANI *et al.*, 2015).

Para cada uma das quatro responsabilidades do modelo, um conjunto de ações é associado a qual deve ser executado iterativamente pelo motor de simulação. Em termos de um modelo orientado a objetos convencional, estes podem ser especificados pelos métodos `Monitor.get`, `Demand.update`, `Capacity.update`, `Effector.set`, que deve ser invocado na ordem: método `Monitor.get` coleta o estado atual do sistema; método `Demand.update` e `Capacity.update` determinam quantas unidades de recurso respectivas de carga de trabalho devem ser adicionadas ou subtraídas do sistema; método `Effector.set` envia os eventos correspondentes para o mecanismo de simulação. Um quinto método `Monitor.getCurrent` destina-se a ser chamado, se necessário, pelos controladores da capacidade e demanda, para obter as medições.

2.2.1 A extensão *OnlineCloudsim*

Um desenvolvimento concreto da discussão pode ser guiado por meio de um estudo de caso do mundo real, que está baseando no trabalho de pesquisa relacionado com controle dinâmico de recursos em sistema de grande escala (NOBILE, 2013). O estudo investiga o uso de aproximações lineares invariantes no tempo (LTI) para modelar a dinâmica de desempenho de um *data center* focando no modelo de serviço da computação em nuvem IaaS (Infraestrutura como serviço) ciente de QoS, e na criação de estratégia baseada em teoria de controle para alocação de recursos elásticos (MAMANI *et al.*, 2015). Em sua instrumentação experimental, os autores aplicaram princípios da MEDC para estender a ferramenta de simulação CloudSim.

Originalmente desenvolvido na Universidade de Melbourne, CloudSim (CALHEIROS *et al.*, 2011) é um *framework* de simulação de eventos discretos com entidades especializadas para o domínio da computação em nuvem. Construído a partir de uma arquitetura de DES (*discrete event simulation*) convencional subjacente, desenvolvida em linguagem Java, o funcionamento do *framework* consiste em iterar sobre uma lista de eventos futuros, onde cada evento está associado a uma entidade de simulação. Durante a simulação, o motor remove o evento da lista de eventos e invoca a entidade correspondente para processá-la. Além disso, para estender e implementar suas próprias entidades de simulação, o usuário pode recorrer à classe `DataCenterBroker`¹ fornecida pelo *framework*. Para esse fim, é necessário que o usuário instancie um objeto `Datacenter` e defina um conjunto de atributos para a infraestrutura como número e características dos servidores, tecnologia de virtualização, política de multiplexação, etc. O `DataCenterBroker` permite que o usuário especifique o número de máquinas virtualizadas (VMs) e as unidades de carga de trabalho a serem submetidas no sistemas, juntamente com outros atributos. No início da simulação, o agente gera eventos para instanciar VMs e enviar a carga de trabalho para o *data center*. Depois disso, a simulação começa a consumir a carga de trabalho previamente definida. Tendo terminado, o *framework* CloudSim gera um relatório com um conjunto de medições

¹ Exemplo da API para projetar um programa de simulação em lote.

definidas pelo desenvolvedor. [Código-fonte 1](#) ilustra em pseudocódigo de uma simples estrutura da “API de simulação”.

Código-fonte 1: Estrutura simplificada do simulador usando o *framework* CloudSim.

```

1 //Simulator's main function
2 void ExecuteSimulator(){
3   Datacenter dc = new Datacenter(...);
4   Broker broker = new DatacenterBroker(...);
5
6   for(i = 0; i < max_vms; i++){
7     vmList.add(new Vm(...));
8   }
9   for(i = 0; i < max_cloudlets; i++){
10    taskList.add(new Cloudlet(...));
11  }
12
13  broker.submitVms(vmList);
14  broker.submitTasks(taskList);
15
16  CloudSim.startSimulation();
17  CloudSim.stopSimulation();
18
19  //print reports ...
20 }
```

Os desenvolvedores do projeto CloudSim disponibilizaram um portal ² público com o código fonte do *framework*, bem como um conjunto de extensões por meio do projeto *CloudSimEx*, que adiciona ao *framework* original funcionalidades extras para fins específicos, tais como sistemas baseados em Web, o paradigma *MapReduce*, problemas com latência no tráfego, políticas de escalonamento, etc. Outra extensão é o *DynamicCloudSim* (BUX; LESER, 2013), que oferece um conjunto de recursos para simular falhas nas máquinas virtuais de acordo com modelos probabilísticos.

A estratégia de simulação de eventos discretos não-estacionários foi aplicada e permitiu projetar o *OnlineCloudSim*, um *framework* que estende o *CloudSim* original com abstrações para simulação *on-line* e análise (transitória) temporal. Portanto, esta extensão está no escopo do projeto *CloudSimEx*. A [Figura 3](#) apresenta o diagrama de classe associado. Comumente as extensões realizadas no CloudSim seguem o processo de estender a classe *DatacenterBroker*, como é o caso do *DynamicCloudSim* e outras que seguem o mesmo processo, mas esta abordagem não resultou ser a mais apropriada para o gerenciamento dinâmico de carga de trabalho e recursos, por esse principal motivo, a extensão é construída sobre a classe padrão *SimEntity* do *CloudSim*, que fornece um ambiente adequado para a implementação da simulação *on-line*. A nova classe

² <https://github.com/Cloudslab>

OnlineBroker acrescenta quatro membros: Demand, Capacity, Monitor e Effector, que o projetista da simulação deve implementar.

Figura 3 – Arquitetura conceitual da estratégia DES não-estacionária.

Fonte: Adaptada de [Pereira et al. \(2015\)](#).

A classe OnlineBroker tem a capacidade de gerar novos eventos durante o curso da simulação. Para este fim, define um novo evento `SAMPLE_TIME`, que desencadeia a evocação gradual dos métodos para implementar as funções de *monitor*, *efector* e controladores de *demand* e *capacidade*. As propriedades que mantêm a lista de VMs e *cloudlets* (jobs) fornecidas pela abordagem em lote relacionadas com o `DataCenterBroker` não são reproduzidas. Em vez disso, a nova classe OnlineBroker utiliza as instâncias concretas de quatro classes abstratas e seus métodos associados `Monitor.get`, `Monitor.getCurrent`, `Demand.update`, `Capacity.update` e `Effector.set`. Os controladores de *demand* e *capacidade* usam os valores adquiridos pelo *monitor* para gerar eventos de simulação para adicionar ou remover *cloudlets* e máquinas virtuais, respectivamente. O *framework* CloudSim não disponibiliza na API uma classe para criar `SimEvents`; por esse motivo uma classe `Event` é criada para essa finalidade. O *efetor* recebe a lista de eventos emitidos pelos controladores e repassa ao *engine* (motor) de simulação. Todo o processo está de acordo com as políticas definidas pelo desenvolvedor.

As políticas que controlam a quantidade de carga de trabalho que chega ao sistema e o número de máquinas virtuais ativas são definidas pelas classes `Demand` e `Capacity`. Os mecanismos por meio dos quais as mudanças são efetivadas e entram em vigor são modelados na classe `Effector`. Dessa perspectiva, carga de trabalho e recursos como máquinas virtuais são entradas, enquanto fatores que influenciam a dinâmica são características do sistema. Portanto, o balanceamento de carga e algoritmos de alocação de máquinas virtuais, por exemplo, são responsabilidades da classe `Effector`, bem como modelos de falha e atrasos —permitindo, dessa maneira, modelar a alocação de máquinas virtuais e também seu impacto sobre a dinâmica global do sistema.

```

1  delay = 0;
2
3  List<Event> Demand.update(){
4      if(Monitor.cloudletsSubmitted()){
5          return null;
6      }
7      for(Task task : taskList) {
8          events.add(Event(delay, SUBMIT, task));
9      }
10     return events;
11 }
12
13 List<Event> Capacity.update(){
14     if (Monitor.getCurrentTime() != 0 ){
15         return null;
16     }
17     for (Vm vm : vmList) {
18         events.add(Event(delay, VM_CREATE, vm))
19     }
20     return events;
21 }
22
23 Long Monitor.getCurrentTime(){
24     return CloudSim.clock();
25 }
26
27 Boolean Monitor.cloudletsSubmitted(){
28     return vmsCreated && cloudletsDone;
29 }
30
31 void Effector.set(cap, dem){
32     if (cap != null) {
33         myBroker.sendEvents(cap);
34         Monitor.vmsCreate = true;
35     }
36
37     if (dem != null){
38         myBroker.sendEvents(dem);
39         Monitor.cloudletsDone = true;
40     }
41 }

```

Para um melhor entendimento, o [Código-fonte 2](#) exemplifica como o funcionamento do original DataCenterBroker pode ser reproduzido pelo OnlineBroker implementando as quatro classes abstratas. O pseudocódigo mostra como o funcionamento original orientado a

processamento em lote muda para a abordagem *on-line* proposta pelos autores. Por exemplo, para o estado inicial da simulação, a *capacidade* e *demand* funcionam da seguinte maneira: no tempo $t = 0$ todas as *cloudlets* e máquinas virtuais são criadas; Depois disso, os métodos correspondentes não fazem nada. O método do Effector só passa os eventos de entrada diretamente para o mecanismo de simulação, praticamente sem efeito. O método do Monitor lê o tempo atual.

2.2.2 Análise transitória com a MEDC-DES

O sistema descrito tem sido utilizado na pesquisa por meio de experimentos usando uma abordagem de *teoria de controle* para autogestão de recursos em computação em nuvem. Na primeira etapa, os autores tem projetado um experimento de simulação *on-line* com objetivo de de criar um modelo empírico analítico capaz de representar como a média utilização do sistema U (parâmetro de desempenho) é afetada por mudanças no número de máquinas virtuais N (parâmetro de capacidade), sob uma carga de trabalho constante W (parâmetro de demanda). O último, parametrizado pela taxa de chegada das requisições, fixou-se como o ponto de repouso do sistema operacional. A Figura 4a oferece uma descrição de alto nível do modelo de simulação.

No programa de simulação, a classe `OnlineBroker` foi instanciada e as classes do *framework* foram implementadas. Mamani *et al.* (2015), consideraram uma política de alocação de recursos para as máquinas virtuais (VM), obedecendo um modelo de estados. No modelo, uma máquina virtualizada pode estar em: *booting* (consumindo recursos, mas não operacional), *running* (servindo novas requisições), *shutting down* (terminando restantes requisições, mas não receber novas) e *off* (recursos liberados). Um balanceador de carga controla o envio de requisições para os recursos disponíveis.

Características dinâmicas relacionadas com o *hardware* e *software* influenciando nos estados *booting* e *shutting down* são principalmente ligados a dinâmica *data center*. No contexto do programa de simulação, a máquina de estados esta implementada na classe que herda da classe abstrata `Effector`.

A classe abstrata `Monitor` foi implementado para coletar dados sobre o número de máquinas virtuais em cada estado, bem como a taxa de utilização média do sistema. Quanto a classe abstrata `Capacity`, seu método `update` foi implementado para produzir um degrau (abrupto) de aumento ou diminuição no número de máquinas virtuais em um determinado instante do experimento, enquanto a classe abstrata `Demand` foi implementado para manter o sistema no ponto operacional de repouso (a geração da carga de trabalho estacionária é o resultado da utilização de uma distribuição do Poisson para os tempos de chegadas e o nível de processamento). Um gráfico com dados adquiridos pelo monitor ao decorrer do experimento é apresentado na Figura 4b.

Verificou-se a hipótese de linearidade dentro da faixa de operação do sinal de saída, por

(a) Modelo de alto nível do sistema IaaS.

(b) Resposta ao degrau na simulação.

Figura 4 – Análise transitória de um sistema IaaS.

Fonte: Nobile (2013).

meio do sinal de entrada disponível, o qual é de natureza estocástica, mas com várias replicas e pelo princípio de superposição encontrou-se nas proximidades do ponto de repouso predefinido.

A hipótese da linearidade foi verificada excursionando o sinal de entrada na vizinhança do ponto de operação, os autores usaram então a saída como um sinal medido U superposto de ruído estocástico; o sinal foi filtrado pela média espacial de várias réplicas. Baseado no princípio da parcimônia, o modelo de ordem mais baixa capaz de captar a dinâmica do sistema foi procurado. Através do ajuste dos dados $N \times U$ com um modelo linear ARX (*Autorregressive model with exogenous input*) de primeira ordem, os autores parametrizaram uma função de transferência discreta $G = U/N$ que numericamente se aproxima da forma exponencial da inércia do sistema. Tendo avaliado a congruência das previsões do modelo analítico e da simulação de eventos discretos, os autores projetaram um controlador clássico proporcional integral (PI) para manter a

taxa de utilização média do sistema em um determinado intervalo, especificado pelo gerenciador do serviço em nuvem, de acordo com um contrato de QoS.

A estratégia, esquematicamente representada no diagrama de blocos de controle é apresentada na Figura 5a, modelado novamente com o uso de extensão do *framework* OnlineCloudSim. Um algoritmo baseado numa média móvel exponencial (EMA) foi implementada no bloco H como um método de classe Monitor. As classes Effector e Demand foram mantidos inalterados. Quanto à classe abstrata Capacity, foi estendida e implementado um algoritmo baseado num controlador **PI**, que utiliza o erro de sinal E para calcular quantas máquinas virtuais devem ser ativadas para processar a carga de trabalho, assim mantendo a utilização do sistema U perto da referência R ; qualquer variação provoca uma reação contrária do controlador alterando o número de máquinas virtuais U .

(a) Modelo do *loop* fechado.

(b) Resultado da simulação com a variação de parametros.

Figura 5 – Estratégia de controle de retroalimentado dinâmico.

Fonte: Nobile (2013).

O controlador PI é implementado pelo algoritmo $N(k) = N(k-1) + E(k-1)(K_i + K_p)$, onde k é o tempo discreto, K_p e K_i são constantes a determinar em função dos requisitos de desempenho dinâmico. Se a planta G (representando os mecanismos do *data center*) é aproximada por um *Linear Time Invariant* (LTI) de primeira ordem, o controlador *Proporcional Integral* (PI) em série e o EMA no sensor H fecham o ciclo retroalimentado, e podem, por conseguinte, resultar em um comportamento oscilatório amortecido. Ao exigir um sistema estável, com dois polos dominantes e requisitos de máximo *overshoot* e tempo de assentamento (*settling time*) depois de uma mudança abrupta, os parâmetros do controlador PI podem ser calculados pelo método do lugar das raízes (*root-locus*). Os autores configuraram o controlador para cumprir as metas de desempenho a seguir: se o gerenciador de recursos desencadeia uma ordem para alterar a taxa de utilização de referência de 70% para 80%, o alocador máquinas virtuais deve garantir que isso acontece com um *overshoot* não maior do que 10% e com um tempo de assentamento não superior de 300ms. Esses dois requisitos de desempenho dinâmico são usados para encontrar os valores dos parâmetros K_p e K_i . Com parâmetros calculados, a Figura 5b mostra o comportamento transiente da utilização do sistema aplicando uma perturbação no ponto de operação (taxa de chegada das requisições), oferecendo uma visão qualitativa da robustez do controle na faixa de operação. A modulação da carga de trabalho e dos recursos fornecidos pelo OnlineCloudsim permitiu obter os resultados apresentados.

2.3 Benchmarking não-estacionário

Embora a análise transiente de sistemas de evento discreto é ainda pouco explorada, comparativamente à principais abordagens tradicionais de avaliação de desempenho, trabalhos científicos abordam e dão luz sobre o assunto e sua importância para a engenharia de sistemas em diferentes campos (HUANG *et al.*, 2013; YANG; LIU, 2010; Minqi Li; YANG; XU, 2014; KACZYNSKI; LEEMIS; DREW, 2012; COEVERING, 1995). Relacionado com a presente revisão da literatura, a maioria dos trabalhos assumem uma abordagem teórica ou de simulação. Além disso, a investigação experimental visa resultados numéricos para uma determinada entrada em vez de obter um modelo analítico generalizado para o comportamento transitório. O presente trabalho de pesquisa, portanto contribui para ampliar o estado da arte nesse domínio.

2.3.1 Abordagens tradicionais

Na literatura existem abordagens que utilizam de modelos dinâmicos para o provimento de soluções de maneira *ad-hoc*. Não há abordagens sistemáticas para a criação de *benchmarks* que foquem na avaliação de desempenho no regime não-estacionário. Problemas comumente abordados na literatura estão relacionados com o gerenciamento de recursos da computação em nuvem e o tratamento da carga de trabalho. Nos trabalhos foi identificado comportamento dinâmico na carga de trabalho e no funcionamento de sistemas e subsistemas. A seguir são apresentados alguns trabalhos relacionados com abordagem não-estacionária.

Em seu trabalho, (Ronghua Zhang *et al.*, 2002) descrevem o projeto e a implementação de uma arquitetura de *middleware* para controlar parâmetros de QoS usando a teoria de controle como ferramenta. O *SoftBus* é um canal que permite o intercâmbio de informações entre os sensores, atuadores e controladores distribuídos em máquinas diferentes espalhadas numa rede.

Em 2001, a IBM propôs o conceito de computação autônoma. Horn (2001) realiza uma comparação de sistemas de computação complexos com o corpo humano, especificamente com o sistema nervoso autônomo, encarregado de várias funções: respiração, digestão, etc. Foi sugerido que os sistemas computacionais complexos deveriam possuir propriedades autônomicas, isto é, deve ser capazes de se cuidar de forma independente da manutenção regular e otimização de tarefas, reduzindo assim a carga de trabalho dos administradores do sistema. Também foram definidos quatro propriedades: auto-configuração, auto-otimização, auto-recuperação e auto-proteção (KEPHART; CHESS, 2003).

O trabalho de (GANDHI *et al.*, 2001) apresenta a modelagem e o *feedback* controle do IBM Lotus NotesTM. Sistemas como estes tem como objetivos maximizar o rendimento e minimizar o tempo de resposta. A comunicação no sistema está baseada em várias *Remote Procedure Call* (RPC). Depois da modelagem, foi utilizada como entrada de controle o número de usuários ativos. No trabalho de (PAREKH *et al.*, 2002) foi realizado um estudo mais aprofundado acima do mesmo software considerando-o como um sistema dinâmico onde a entrada foi número máximo de usuários *MaxUsers*, e como saída o RIS (chamadas a possesos remotos atuais no servidor). A Figura 6 mostra uma visão geral da implementação realizada acima do servidor de e-mails.

Figura 6 – Controle *feedback* do IBM Lotus NoteTM.

Fonte: Adaptada de Parekh *et al.* (2002).

O trabalho de Singh *et al.* (2010), analisa as cargas de trabalho que flutuam em varias escalas no tempo, argumenta que a característica não-estacionária da carga de trabalho que aplicações da Internet recebem podem ter um impacto significativo sobre o processamento global imposto aos servidores dos *data centers*. Propõem uma técnica de provisionamento dinâmico que lida com cargas de trabalho não-estacionárias e com alterações na capacidade dos servidores nos *data centers*. A técnica emprega o algoritmo de clusterização *k-means* para determinar

automaticamente o tipo de carga de trabalho e um modelo de filas para definir a capacidade do servidor para cada tipo de carga. No protótipo, foi utilizado o *benchmark* TPC-W.

Produto da crescente utilização da teoria de controle para a construção de gerenciadores de recursos em servidores Web, de bancos de dados, entre outros sistemas. [Hellerstein, Morrison e Eilebrecht \(2010\)](#) salientam valor potencial da utilização da teoria de controle. Como resultado da construção do *pool* de *threads* do .NET, desenvolveram uma metodologia que: Fornece um projeto de controlador extensível que integra diversas técnicas de controle. Fornece uma infra-estrutura de teste que permite executar eficientemente um grande número de casos de teste. Constrói casos de teste onde o desempenho ideal do controlador é conhecido a priori para avaliar o desempenho do controlador facilmente. Ajusta os parâmetros do controlador para alcançar bons resultados para várias métricas de desempenho. Os autores concluíram que a metodologia poderia ser estendida, especialmente para projetar controladores para sistemas distribuídos.

O dimensionamento de recursos na computação em nuvem é um tema difícil de tratar por exemplo, [Verma, Bagrodia e Jaiswal \(2014\)](#) estudaram a carga de trabalho de quatro grandes centros de dados e observaram que o custo de instalações que usam o gerenciamento de máquinas virtuais dinâmico comparado com o estático não é tão grande com relação a estimações de estudos anteriores. Além disso, o gerenciamento semi-estático mostra-se melhor, isto coloca um ponto de interrogação relacionado com a aplicabilidade de soluções dinâmica no gerenciamento de máquinas virtuais no mundo real. Como resultado adicional, observaram que os picos na carga de trabalho causaram maior impacto na memória e não sobre as CPUs como tradicionalmente e assumido.

Para sistemas com múltiplas camadas, como por exemplo sistemas PaaS e SaaS, ([GANDHI et al., 2012](#)) apresenta uma solução para o problema de alocação dinâmica de recursos originada por flutuações rápidas da carga de trabalho comuns nos data centers. Utilizam uma abordagem oportunista com recursos “subutilizados” em sistemas multicamada, utiliza a cama de dados alocando servidores de aplicação como reserva para suportar temporariamente picos de carga de trabalho até o recurso final estiver disponível na camada de aplicação. A motivação desta abordagem é a rápida mudança da carga de trabalho e a demora existente na provisão de recursos na camada de aplicação.

Os trabalhos citados relatam processos de modelagem, desenvolvimento e teste de aplicações que trabalham em ambientes de dinâmica intrínseca, fazem abordagem para dimensionamento de sistemas de grande porte na nuvem, mas não dão atenção as próprias ferramentas que são utilizadas, no caso, os *benchmarks*.

2.4 Computação em nuvem

Um cenário comum nos dias atuais é aquele em que as organizações alugam infraestrutura computacional fornecida por outras organizações especializadas no fornecimento de recursos computacionais como serviços. A ideia de terceirizar vários departamentos de uma empresa não é nova; no final do século 20, várias empresas começaram esse processo, por exemplo, na indústria bancária (ALTINKEMER; CHATURVEDI; GULATI, 1994). A Computação em Nuvem pode ser classificada como um novo paradigma para o provisionamento dinâmico de serviços de computação, normalmente alocados em centros de dados ou *data centers*. Nessas instalações há grandes conjuntos de servidores interligados através de uma rede de alta velocidade (BUYYA; RANJAN; CALHEIROS, 2010).

Segundo Sosinsky (2011), a palavra “nuvem” faz referência a dois conceitos essenciais: abstração e virtualização. Em primeiro lugar, a computação em nuvem abstrai os detalhes de implementação dos sistemas para usuários e desenvolvedores. Os aplicativos são executados por computadores não especificados, os dados são armazenados em locais que são desconhecidos, a administração de sistemas é terceirizada e o acesso de usuários é onipresente. Em segundo lugar, a computação em nuvem virtualiza sistemas com o objetivo de compartilhar recursos, tais como armazenamento, CPU, memória, etc. Os recursos virtualizados são disponibilizados segundo a demanda a partir de infraestruturas centralizadas. É apresentado um cenário potencialmente escalável, disponível universalmente. Com a computação em nuvem, pode-se iniciar com recursos básicos e se tornar grande muito rápido. Por esta razão, o fenômeno da computação em nuvem é considerado um advento revolucionário na implantação de sistemas computacionais (ARMBRUST *et al.*, 2010).

Uma definição mais formal foi dada pelo NIST (*National Institute of Standards and Technology*) em (BADGER *et al.*, 2011): “Computação em Nuvem é um modelo para habilitar o acesso ubíquo via rede, conveniente e sob demanda a um conjunto compartilhado de recursos computacionais que podem ser rapidamente provisionados e retornados com mínimo esforço administrativo ou interação com o provedor de serviços”.

2.4.1 Características da computação em nuvem

Segundo Badger *et al.* (2011), o modelo básico compõe-se de 5 características essenciais que são descritas a seguir:

- Auto-atendimento sob demanda. Um consumidor pode unilateralmente, segundo sua necessidade, utilizar recursos computacionais como o tempo do servidor, armazenamento, rede, etc; tudo isso sem intervenção humana de parte do prestador de serviços.
- Amplo acesso à rede. Recursos estão disponíveis através da rede e acessíveis por mecanismos padrão que promovem o uso de plataformas heterogêneas, simples ou complexas tais

como telefones celulares, *tablets*, *notebooks*, *desktops*, entre outros.

- **Agrupamento de recursos.** Os provedores de recursos são reunidos para atender vários consumidores através de um modelo multi-usuários, com diferentes recursos físicos e virtuais alocados e realocados dinamicamente de acordo com a demanda. O cliente geralmente não tem conhecimento exato da localização nem o controle dos recursos, embora em algumas situações possa ser de seu interesse, mas poderia especificar o local (por exemplo, o estado, o país, onde encontra-se o *data center*).
- **Rápida elasticidade.** Recursos podem ser rapidamente e elasticamente provisionadas, em alguns casos, automaticamente, para rapidamente aumentar e diminuir os recursos. Para o consumidor, os recursos disponíveis frequentemente parecem ser ilimitados e podem ser comprados em qualquer quantidade a qualquer momento.
- **Medição do serviço.** Os sistemas de nuvens automaticamente controlam e aperfeiçoam o uso dos recursos através de medição em algum nível de abstração apropriado para do tipo de serviço (por exemplo, processamento, armazenamento, largura de banda e contas de usuários ativos). O uso de recursos pode ser monitorado, controlado e reportado proporcionando transparência para o provedor e consumidor do serviço utilizado.

2.4.2 Tecnologias

A computação em nuvem é constituída de várias tecnologias, [Zhang, Cheng e Boutaba \(2010\)](#) menciona algumas das mais importantes:

- **Computação em grade:** É um paradigma de computação distribuída que coordena os recursos ligados em rede para alcançar um objetivo computacional comum. O desenvolvimento da computação em grade foi originalmente dirigido para aplicações científicas. Existem várias semelhanças com a computação em nuvem, começando pela existência de várias camadas: aplicação, física (máquinas reais ou virtuais), recursos, entre outras ([FOSTER et al., 2008](#)). A utilização de recursos distribuídos para atingir os seus objetivos a faz semelhante com a computação em nuvem. Porém, a computação em nuvem está um passo a frente, especificamente pela utilização intensiva da virtualização, permitindo o compartilhamento e provisionamento dinâmico de recursos ([VAQUERO et al., 2008](#)).
- **Computação de utilidade:** Representa o modelo de fornecer recursos sob demanda e cobra a seus clientes baseado no uso que os clientes fazem do recurso, esse é o caso contrário de estabelecer uma taxa fixa. De alguma maneira a computação em nuvem pode ser percebida como uma realização da computação de utilidade (*Utility computing*). Segundo [Rappa \(2004\)](#), a computação de utilidade se refere a um modelo de negócios baseado na utilização, que é formada por uma série de características que são típicas nos serviços públicos. Os usuários consideram o serviço uma necessidade onde a alta confiabilidade do

serviço é fundamental. Nesse modelo, os serviços são escaláveis, beneficiando economias de larga escala uma vez que a capacidade de utilizar plenamente os recursos de uma única entidade é limitada.

- **Virtualização:** 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 VM (Máquina virtual). Outro termo bem conhecido nesse ambiente é o VMM (Monitor de máquinas virtuais) e a máquina que hospeda o VMM e as VMs é conhecido como *Host*. A virtualização é a base da computação em nuvem, ela estabeleceu uma clara vantagem comparada com a computação em grade (VAQUERO *et al.*, 2008). O compartilhamento de recursos físicos apresenta diferentes níveis de desempenho. Por um lado, o desempenho das CPUs é praticamente o mesmo em máquinas virtualizadas como na máquina *Host*. Por outro lado, como descrito no trabalho de Armbrust *et al.* (2010), a entrada e saída ainda é problema latente na computação em nuvem.
- **Computação autônoma:** Originalmente criado pela IBM em 2001 (HORN, 2001), a computação autônoma visa a construção de sistemas computacionais com capacidades de autogestão, ou seja, esse tipo de sistema reage automaticamente a mudanças internas e externas sem a intervenção humana. O objetivo da computação autônoma é superar a complexidade do manejo dos sistemas modernos. Embora a computação em nuvem apresente certas características autônomicas, tais como provisionamento de recursos automáticos, seu objetivo é reduzir o custo no gerenciamento de recursos.

2.4.2.1 Data center

Um *data center* é o conjunto de servidores instalados dentro de uma grande sala, que pode se encontrar do outro lado do mundo, sendo acessado através da Internet (VELTE; VELTE; ELSERPETER, 2009). Uma explicação mais detalhada é apresentada por Soundararajan e Anderson (2010): “um *data center* consiste em *hosts* físicos, armazenamento e infraestrutura de rede, alimentação/resfriamento de *hardware* e um *framework* de gerenciamento/monitoramento para a operação em curso”.

Segundo Soundararajan e Anderson (2010), os *hosts* estão organizados em bastidores. A energia é encaminhada para cada bastidor, e cada bastidor também inclui certa quantidade de armazenamento e *hardware* de rede. Para permitir que vários *hosts* compartilhem dados e para simplificar o *backup*, a maioria dos *data centers* utilizam tecnologias para o armazenamento compartilhado, tais como o SAN (*Storage Area Networks*) ou NAS (*Network-attached storage*). SAN exporta armazenamento em nível de dispositivo para cada *host*, de modo que esse dispositivo se pareça com um disco rígido local, entretanto, ele está residindo em um bastidor diferente. Os protocolos comuns SAN permitem utilizar *FibreChannel* e o iSCSI (*Internet Small Computer System Interface*). NAS exporta o armazenamento em nível de arquivos e também usa uma

infraestrutura de rede padrão. Estes dispositivos são projetados para alta largura de banda e baixa latência de acesso compartilhado para um grande número de *hosts*.

Os *data centers* também incluem um *framework* de gestão, a fim de avaliar o estado dos componentes. Os administradores têm ferramentas para configurar a rede, o armazenamento, a capacidade de processamento dos *hosts*. Os administradores também devem monitorar o *hardware* para verificar se há falhas com a intenção de realizar as correções.

2.4.3 Arquitetura da computação em nuvem

A computação em nuvem se caracteriza por ser o produto de várias tecnologias, como foi visto na Seção 2.4.2. Algumas das semelhanças mais marcantes vêm principalmente da computação em grade, juntamente com a influência recebida da computação de utilidade, assim como dos sistemas distribuídos em geral (FOSTER *et al.*, 2008).

Arquitetura da computação em nuvem se caracteriza pela presença de várias camadas, onde cada uma delas pode ser oferecida e vendida na forma de serviços (*pay-as-you-go*). O consumidor pode escolher alguma das camadas e começar a utilizá-la por um determinado tempo e com um determinado nível de serviço ou QoS.

Na Figura 7 são apresentadas as principais camadas. A distinção entre elas principalmente é pelo tipo de recurso que é oferecido para o consumidor. Tipicamente a computação em nuvem possui quatro camadas, algumas delas associadas com conceitos tais como: IaaS (Infraestrutura como serviço), PaaS (Plataforma como serviço), e SaaS (*Software* como serviço) (ARMBRUST *et al.*, 2010). Existem uma terminologia relacionada com esses termos, ela é conhecida como “XaaS (X como serviço)” (ARMBRUST *et al.*, 2009; SCHAFFER, 2009), onde o X representa: infraestrutura, *hardware*, plataforma, etc.

2.4.4 Modelos de serviço

Existem diversos modelos de serviços descritos na literatura. Três deles têm sido universalmente aceitos: *IaaS*, *PaaS*, e *SaaS*. Esses três modelos de serviços, em conjunto chegaram a ser conhecidos como o modelo SPI (*Software, Plataforma, e Infraestrutura*) da computação em nuvem (SOSINSKY, 2011). Um modelo de referência é apresentado na Figura 8. Os níveis acima vão herdando as características dos anteriores.

Software como serviço

SaaS (*Software as a Service*), é um termo associado à computação em nuvem, mas vem sendo utilizado desde a década de 1990 e, portanto, é anterior a computação em nuvem. SaaS também é conhecido comumente como “*Web Services*”. Uma definição simples do modelo SaaS é utilizada pelo NIST em (BADGER *et al.*, 2011): “Software implantado como serviço hospedado, acessado pela Internet”.

Figura 7 – Arquitetura da computação em nuvem.

Fonte: Zhang, Cheng e Boutaba (2010).

Este modelo de serviço oferecido pela computação em nuvem é um dos mais completos, porque um fornecedor oferece *hardware* e *software* como solução completa. Na Figura 8 são apresentados todos os recursos envolvidos dentro do modelo SaaS. Com exceção da interação do usuário com o *software*, todos os outros aspectos do serviço são abstraídos.

SaaS conta com vantagens em comparação com o *software* tradicional: os custos de manutenção são reduzidos, incluem atualizações automáticas, a implementação de mudanças são mais rápidas, etc. Suporta vários usuários e fornece um modelo de dados compartilhado por meio de um modelo de instância única, comumente chamado de *Multitenancy*.

Segundo Velte, Velte e Elsenpeter (2009), SaaS pode ser dividido em duas categorias principais:

- Serviços empresariais: são soluções de negócios oferecidas para empresas. Elas são vendidas através de um serviço de subscrição. Aplicações abrangidas nesta categoria incluem processos de negócio, como aplicações de gestão, cadeia, fornecimento, relações com clientes, e ferramentas semelhantes orientadas a negócios.
- Serviços orientados ao cliente: são oferecidos ao público em geral com base em assinatura, mas também eles são oferecidos gratuitamente e sustentados por publicidade. Exemplos desta categoria incluem *Webmail*, jogos *online*, entre outros.

Plataforma como serviço

A PaaS (*Platform as a Service*) em nuvem fornece um conjunto de ferramentas para convenientemente desenvolver, implantar e administrar o *software* que está estruturado para suportar um grande número de assinantes, processar uma grande quantidade de dados, e, potencialmente ser acessado de qualquer ponto da Internet (BADGER *et al.*, 2011).

Figura 8 – Modelo de referência da Computação em Nuvem.

Fonte: Sosinsky (2011).

Nuvens com PaaS normalmente fornecem um conjunto de blocos de construção de *software* e um conjunto de ferramentas de desenvolvimento, tais como linguagens de programação e suporte em tempo de execução, ambientes que facilitem a construção de aplicativos, *software* para armazenamento de dados, entre outros. Na Figura 8 é apresentado o modelo interagindo com o IaaS e o SaaS. A implantação de um novo aplicativo na nuvem usando PaaS não é muito mais difícil do que o *upload* de um arquivo na Web. Nuvens PaaS fornecem e se encarregam da manutenção de vários recursos computacionais —processamento, armazenamento e rede— que os clientes precisam (SOSINSKY, 2011).

Com todas essas facilidades, também existem problemas. Manter essa grande quantidade de pacotes de *software*, gera uma complexidade adicional com possíveis pontos de falha. Zhang et al. (2011b), propõem uma abordagem para ajustar o *middleware* com funcionalidades da computação autônoma de ajuste automático, com o intuito de suportar uma flexível compensação entre funções de negócios e processamentos adaptativos.

PaaS conta com um ambiente de desenvolvimento totalmente integrado, isto é, todas as ferramentas e serviços fazem parte do serviço PaaS. Em um modelo de PaaS, os clientes podem interagir com o *software* para inserir e recuperar dados, implantar mudanças, obter resultados, todo ao redor das limitações que o provedor da plataforma o permita. O cliente não assume responsabilidades pela manutenção do *hardware* ou *software*, ele é responsável apenas pela interação com a plataforma. O provedor é responsável por todos os aspectos

operacionais do serviço, entre eles a manutenção e gestão do ciclo de vida dos produtos (SOSINSKY, 2011).

Uma das principais dificuldades com PaaS é a dependência da plataforma. Um aplicativo escrito na linguagem de programação *Python* usando o *Google App Engine* provavelmente funcionará apenas nesse ambiente.

Infraestrutura como serviço

IaaS (*Infrastructure as a Service*) é um modelo de serviço onde o *hardware* é virtualizado. Nesse modelo, o fornecedor de serviço é proprietário dos equipamentos, tais como: servidores de aplicativos, servidores de armazenamento, infraestrutura de rede, etc. O desenvolvedor pode criar máquinas com *hardware* virtual onde pode instalar e desenvolver aplicações e serviços (SOSINSKY, 2011). A Figura 8 mostra os diferentes componentes que integram no modelo IaaS, desde a parte mais básica como o *hardware* passando pela tecnologia de virtualização até a segurança e o *software* de gerenciamento.

2.4.5 Modelos de implantação

Grandes organizações com uma infraestrutura de virtualização existente podem optar por implementar uma solução de nuvem privada, porque oferece o controle completo, tornando-o mais seguro, em virtude da solução ser completamente implementada por trás de um *firewall* corporativo da organização. Organizações de pequeno e médio porte, sem a infraestrutura adequada podem optar por uma abordagem de nuvem pública através de um acordo com um provedor de serviços. Algumas outras organizações como necessidades diferentes das anteriormente citadas, podem até optar por seguir uma abordagem de nuvem híbrida.

Nuvem privada

A infraestrutura computacional da nuvem é dedicada para uso exclusivo de uma única organização, esta compreende vários consumidores, por exemplo, as unidades de negócios. Fornece a sua organização o controle completo sobre quais recursos virtualizados podem ser disponibilizados para seus usuários e como estes recursos podem ser provisionados e entregues. Com respeito a propriedade e o gerenciamento, pode ser que a mesma organização esteja encarregada (Figura 9) ou pode ser também um terceiro ou alguma combinação deles (Figura 10). A nuvem privada também pode existir fisicamente dentro ou fora das instalações da organização. Nuvens privadas têm algumas características ressaltantes, por exemplo, são mais caras e com maior segurança em comparação com as nuvens públicas.

Nuvem pública

A infraestrutura é provisionada para uso aberto ao público em geral. A propriedade e o gerenciamento pode ser responsabilidade de outra empresa, ou de uma organização acadêmica ou governamental, ou alguma combinação delas. Algumas das características

Figura 9 – Modelo de uma nuvem privada.

Fonte: [Badger et al. \(2011\)](#).

Figura 10 – Modelo de uma nuvem privada fora da organização.

Fonte: [Badger et al. \(2011\)](#).

mais ressaltantes de uma nuvem pública estão indicadas na [Figura 11](#). A nuvem pública permite o acesso a vários tipos de clientes através de várias interfaces.

Um dos principais motivos da utilização das nuvens públicas é a redução dos custos, mas com o perigo latente relacionado à confidencialidade dos dados armazenados. As opções de configuração são normalmente menores em comparação com uma nuvem privada.

Nuvem híbrida

A infraestrutura de uma nuvem híbrida é uma composição de duas ou mais infraestruturas de nuvem distintas (nuvem privado ou pública) que pertencem a entidades diferentes, mas estão unidas por tecnologias de comunicação padronizadas, permitindo a portabilidade de dados e de aplicativos. Com esta comunicação poderia ser realizado um balanceamento de carga das aplicações entre as nuvens que conformam a nuvem híbrida.

Figura 11 – Modelo de uma nuvem pública.

Fonte: [Badger et al. \(2011\)](#).

2.5 Virtualização de sistemas operacionais

A virtualização tem origens na década de 70 com as primeiras incursões em máquinas construídas pela IBM. Elas possuíam grande capacidade de processamento e suportavam a execução de várias instâncias de sistemas operacionais em paralelo.

Com o advento dos computadores pessoais, a virtualização enfrentou-se uma época de esquecimento, principalmente pela flexibilidade e ao baixo custo de aquisição dos computadores pessoais. A indústria ganhou com a arquitetura PC, mas também perdeu uma arquitetura especializada na execução de vários sistemas operacionais simultâneos, que compartilhavam todos os recursos físicos disponíveis.

Atualmente, a virtualização vem ao encontro do aumento da potência computacional do *hardware* (lei de Moore). A cada dia, as máquinas ganham capacidade computacional, enquanto o *software* não acompanha a demanda. Nesse ambiente, a virtualização foi de fundamental importância no aproveitamento da diferença existente entre *hardware* e *software*, maximizando os benefícios dos recursos físicos utilizados. Além disso, a virtualização se tornou o suporte que possibilitou a migração de vários serviços para a computação na nuvem.

A tecnologia que torna possível a virtualização é associado ao conceito de **hipervisor**. Para entendimento pleno do que vem a ser um hipervisor é necessário um entendimento prévio sobre um **VMM** (*Virtual Machine Monitor* - Monitor de Máquinas Virtuais) (AGESEN *et al.*, 2010). O **VMM** é o *software* responsável pela emulação de componentes de *hardware* virtual (i.e. processadores, placas de rede e de vídeo). O *hardware* virtual fornecido pelo **VMM** faz parte do **VM** (*Virtual Machine* - Máquina Virtual). Os sistemas operacionais que são executados pelas máquinas virtuais são comumente conhecidos como sistemas operacionais virtualizados ou sistemas *Guest*. Dispondo dessa pequena explanação, é possível entender o conceito de hipervisor como uma aplicação composta pelo **VMM** e um sistema operacional hospedeiro ou **Host** (AGESEN *et al.*, 2010). O hospedeiro pode ser um sistema operacional de propósito geral ou um sistema modificado ou desenvolvido especificamente para virtualizar outros sistemas operacionais (BARHAM *et al.*, 2003).

No livro de (TULLOCH, 2010) é apresentada uma classificação de hipervisores baseada em dois critérios: pelo acesso ao *hardware* como Tipo 1 (*bare-metal*) ou Tipo 2 (*hosted*); e pelo projeto do hipervisor como monolítico ou microkernel.

2.5.1 Hipervisor tipo 1

Este tipo de hipervisor caracteriza-se pela pouca sobrecarga na execução das máquinas virtuais e, em consequência, sistemas virtualizados obtém melhor desempenho. Do ponto de vista da interação com o *hardware*, este tipo de hipervisor permite a interação direta dos sistemas virtualizados com os recursos físicos do sistema operacional hospedeiro, fornece uma interface que reduz ao mínimo a sobrecarga da virtualização. Esse tipo de hipervisor também é conhecido

por *bare-metal*.

Parte para dar suporte à virtualização (BARHAM *et al.*, 2003), várias estruturas do sistema hospedeiro, como: o escalonador de processos e o gerenciador de memória, são modificadas. Todas essas mudanças permitem ao sistema operacional virtualizar sistemas operacionais com uma sobrecarga reduzida. Na Figura 12, pode-se observar que não há diferenças entre o sistema operacional hospedeiro e o VMM. Entre alguns exemplos de sistemas de virtualização que estão de acordo com essa descrição tem-se o *Microsoft Hyper-V*³, *Citrix XenServer*⁴ e *VMware ESX Server*⁵.

Figura 12 – Hipervisor *bare-metal*.

Fonte: Tulloch (2010).

2.5.2 Hipervisor tipo 2

Está caracterizado pela separação entre o VMM e o sistema operacional hospedeiro, este tipo de hipervisor também é conhecido como *hosted*. Uma das principais vantagens desse tipo de virtualização é simplicidade na instalação, principalmente pelas poucas mudanças no sistema operacional do hospedeiro. No entanto, esse tipo de virtualização não suporta grande número de máquinas virtuais, devido à sobrecarga gerada pela camada de *software* intermediária, dificultando a execução das máquinas virtuais ou diminuindo o desempenho.

O sistema operacional hospedeiro continua com as características originais, por exemplo, na Figura 13 vemos a separação do VMM e o sistema operacional hospedeiro. O hipervisor *hosted* não são projetados para ambientes de produção, mas sim para um ambiente de pequena escala onde o desempenho não é o foco. Exemplos comerciais de hipervisores com essa arquitetura são o *Microsoft Virtual Server* e *VMware Server*.

2.5.3 Hipervisor Monolítico

A presença obrigatória de controladores ou *drivers* é uma característica do hipervisor monolítico, esses controladores permitem um acesso com baixo nível de *overhead* aos recursos

³ <http://www.microsoft.com/HyperV>

⁴ <http://www.xensource.com>

⁵ <http://www.vmware.com/virtualization/>

Figura 13 – Hipervisor *hosted*.Fonte: [Tulloch \(2010\)](#).

de *hardware* disponíveis. Entre as vantagens proporcionadas pela presença de controladores, está a ausência de um gerenciador de acesso (*proxy* de acesso ao *hardware*), que no caso do hipervisor *microkernel* é obrigatório. Dessa forma, todas as máquinas virtuais têm a capacidade de interagir diretamente com o *hardware* sem a necessidade de instalar *software* adicional, a [Figura 14](#) mostra como todas as máquinas possuem acesso direto aos recursos físicos através de um controlador externo. Esse tipo de acesso quase direto ao *hardware* traz, no entanto, uma preocupação com a segurança na execução de processos, um fator crítico para garantir um bom funcionamento de todas as máquinas virtuais garantindo a privacidade das informações.

Figura 14 – Hipervisor monolítico.

Fonte: [Tulloch \(2010\)](#).

Dentre as desvantagens desta arquitetura pode-se citar uma limitação na quantidade de *hardware* disponível que permite essa configuração. Um exemplo é a arquitetura de processadores da família x86, que não permite naturalmente a virtualização ([AGESEN et al., 2010](#)). Somente os processadores mais atuais fornecem características especiais para acelerar o funcionamento das máquinas virtuais que utilizam este tipo de hipervisor, um exemplo dessas tecnologias é o conjunto de instruções VT-X ([UHLIG et al., 2005](#)). Em consequência, esses hipervisores são

dependentes dos fabricantes de *hardware*. Como exemplos de implementações que utilizam essa arquitetura estão o *VMware ESX Server* e *KVM*⁶ (*Kernel-based Virtual Machine*).

O hipervisor KVM (*Kernel-based Virtual Machine*) (KIVITY *et al.*, 2007) é uma alternativa nativa fornecida pelo Kernel Linux, “encontra-se implementado como um módulo no Kernel”, e quando é carregado, transforma o sistema (sistema operacional e VMM) em um hipervisor do Tipo 1, *bare-metal*, vide subseção 2.5.1.

O KVM foi projetado após o advento da virtualização assistida por *hardware* em processadores com arquitetura x86 explicado na Figura 2.5.5. Uma vantagem é a abstenção de adaptação do sistema hospedeiro para implementar funcionalidades que já estão fornecidas pelo *hardware* (UHLIG *et al.*, 2005), também ele aproveita todas as funcionalidades que tem o sistema operacional para armazenamento e comunicação através do VMM-QEMU (BELLARD, 2005) e de técnicas de paravirtualização usando o *VirtIO* (RUSSELL, 2008). O hipervisor KVM precisa das instruções Intel *VT-X* ou *AMD-V* habilitadas.

2.5.4 Hipervisor Microkernel

Existe uma similaridade com o hipervisor monolítico, mas com a diferença de que no microkernel os controladores estão instalados em uma das máquinas virtuais, que tem a função de gerenciador de acesso ao *hardware*. Essa máquina virtual tem o propósito de gerenciamento, ela tem uma grande importância na arquitetura, porque está encarregada da gestão dos dispositivos de entrada e saída através de controladores especiais, permitindo o acesso rápido ao *hardware*. Acessos à CPU são realizados diretamente pelas máquinas virtuais e não passam pela anuência da máquina virtual administradora.

Os hipervisores com arquitetura microkernel são de alto desempenho e permitem que várias máquinas virtuais possam ser executadas em paralelo. Isso deve-se ao fato de que o hipervisor é praticamente o sistema operacional hospedeiro. Relacionado com a segurança, existe um isolamento de certo conjunto de instruções que são executadas pelas máquinas virtuais no VMM e não diretamente no hipervisor.

Na Figura 15 é apresentado um esquema geral da arquitetura, onde existe um canal de comunicação entre as máquinas virtuais e a máquina virtual encarregada de gerenciar os dispositivos de entrada e saída. O objetivo desse canal é permite gerenciar os recursos como rede ou armazenamento, que a diferença das CPUs, geralmente não estão presentes em grande número nos computadores. Como exemplos de implementações que utilizam essa arquitetura tem-se o *Microsoft Hyper-V* e *Citrix XenServer*.

⁶ <http://www.linux-kvm.org>

Figura 15 – Hipervisor microkernel.

Fonte: [Tulloch \(2010\)](#).

2.5.5 Arquitetura x86 na virtualização

A virtualização de sistemas operacionais surgiu há alguns anos com os conhecidos *Mainframes*, mas com o decorrer do tempo esse enfoque da computação foi mudando para uma alternativa mais barata e flexível: os PCs. Assim, fabricantes de *hardware* deixaram de lado a virtualização, que por vários anos viveu em esquecimento ([RAJ; SCHWAN, 2007](#)).

Uma vez que o *hardware* começou a evoluir mais rapidamente do que o *software* e a subutilização de recursos ficaram mais evidentes, um novo ambiente propício à virtualização foi criado. No ano de 1998, a VMware colocou no mercado o primeiro sistema capaz de virtualizar sistemas operacionais acima da arquitetura x86. Vários problemas tiveram que se solucionar para que fosse possível o suporte à virtualização, já que essa arquitetura não suportava nativamente a execução de vários sistemas operacionais simultâneos de maneira otimizada. Em meados de 2004, o projeto XEN⁷ da Universidade de Cambridge foi iniciado com o mesmo propósito, dar suporte à virtualização por máquinas da arquitetura x86. Como resultados das diversas pesquisas na área, foram desenvolvidas várias tecnologias e novos termos como:

- Virtualização total (*full virtualization*)
- Paravirtualização
- Virtualização assistida por *hardware*

Conforme ilustrado na [Figura 16](#), a arquitetura x86 oferece quatro níveis de privilégios conhecidos como anéis (*rings*) 0, 1, 2 e 3. Como referência, as aplicações em nível de usuário normalmente são executados no anel três ([VMWARE, 2007](#)). O sistema operacional, como precisa ter acesso direto à memória e ao *hardware* e instruções específicas, atua no anel zero. No caso da virtualização, ela afronta o problema de gerar instabilidade no sistema todo, por exemplo, algumas das instruções das máquinas virtuais podem interferir no correto funcionamento do

⁷ <http://xen.org>

sistema hospedeiro, a alteração de registros como SMSW⁸ que permite salvar o estado da máquina (virtual ou real) em um instante determinado, a sobre escrita de valores afeta a integridade dos sistemas operacionais que acessam a CPU.

Figura 16 – Níveis de execução na arquitetura x86.

Fonte: VMware (2007).

Virtualização total

Esta abordagem é composta pela combinação de tradução binária e técnicas de execução direta na CPU. Sempre que for possível, o código de nível de usuário será executado diretamente no processador para garantir um bom desempenho. A tradução de instruções acontece quando são detectadas chamadas que possam gerar instabilidade no sistema hospedeiro ou em outras máquinas virtuais. A tradução envolve um processo de execução controlada realizada pelo VMM como é ilustrado na Figura 17, como parte do processo, segue a atualização dos respectivos registros na máquina virtual que originou a chamada. O VMM fornece às máquinas virtuais todos os serviços de um sistema físico, incluindo um BIOS virtual, dispositivos virtuais de entrada e saída, etc.

Entre as vantagens oferecidas pela virtualização total estão: melhor isolamento e segurança entre as máquinas virtuais e aumento de portabilidade e simplicidade para migração. Como desvantagens tem-se a perda de desempenho para um grupo de instruções que precisam ser traduzidas e executadas de maneira controlada (AGESEN *et al.*, 2010). Soluções de virtualização que utilizam essa abordagem são: VMware e Microsoft Virtual Server.

Paravirtualização

A paravirtualização está baseada em um conjunto de técnicas que permitem uma comunicação entre o hipervisor e o sistema operacional virtualizado (*Guest*), com o intuito de reduzir a sobrecarga em cada instrução para se obter um bom desempenho e eficiência nos processos envolvidos.

⁸ <http://css.csail.mit.edu/6.858/2012/readings/i386/SMSW.htm>

Figura 17 – Tradução binária.

Fonte: VMware (2007).

Como é apresentado na Figura 18, existe uma camada ou *interface* de virtualização constituída por *hypercalls* que permitem uma comunicação direta com o hipervisor. Essa camada de *hypercalls* é acessada pelo sistema operacional virtualizado através de um *driver* que redireciona as requisições para o verdadeiro controlador. No sistema hospedeiro, a camada de *hypercalls* fornece várias interfaces para o gerenciamento de memória, interrupções, *clock* do sistema (*time keeping*), rede, etc.

Figura 18 – Hypercalls na paravirtualização.

Fonte: VMware (2007).

A paravirtualização difere da virtualização total, porque o sistema virtualizado precisa de controladores que permitam acessar as *hypercalls* para a obtenção de melhoras no desempenho. Existem sistemas operacionais com certas restrições que não permitem alterações profundas realizadas por terceiros, por exemplo: Windows⁹, Mac OS, etc. Por esses motivos a paravirtualização enfrenta problemas de portabilidade que diminui o

⁹ Windows fornece soluções próprias de paravirtualização.

número de sistemas operacionais que podem ser suportados por um determinado hipervisor. A paravirtualização também é realizada através da instalação de *software* utilitários nos sistemas virtualizados, a maneira de *backdoor* o hipervisor muda o estado das estruturas internas dos sistemas virtualizados. Exemplos dessas funcionalidades são: a sincronização de tempo, registro e desligamento convidado, etc.

Virtualização assistida por hardware

Com o fortalecimento das tecnologias de virtualização, fornecedores de *hardware* começaram rapidamente a desenvolver novas soluções para simplificar e otimizar os processos realizados pela virtualização. Como resultado, empresas como Intel e AMD (*Advanced Micro Devices*) criaram processadores com conjuntos de instruções especiais chamadas comercialmente de **VT-X** e **AMD-V** respectivamente, essas mudanças proporcionavam melhorias que permitiam ao VMM executar em um novo modo, abaixo do anel zero como é ilustrado na Figura 19, onde instruções possivelmente perigosas automaticamente são capturadas para serem tratadas pelo hipervisor eliminando a necessidade de uma monitoração constante em *software*.

Figura 19 – Esquema da virtualização assistida por *hardware*.

Fonte: VMware (2007).

2.5.6 Entrada e saída na virtualização

Um dos principais problemas persistentes na virtualização é a entrada e saída. As CPUs foram evoluindo rapidamente, principalmente pelo aumento no número de núcleos, que permite suportar uma maior quantidade de processamento em paralelo. Essa característica é ótima para a virtualização, mas por outro lado, o ritmo do avanço das CPUs não foi proporcional para outros tipos de dispositivos, como por exemplo: unidades de armazenamento secundário, placas de rede, entre outros. Nas placas de rede, foram realizadas pesquisas tentando superar essa limitação, em

(RAJ; SCHWAN, 2007) utilizaram um dispositivo de rede com vários núcleos e conjuntamente fizeram adaptações no hipervisor com o intuito de diminuir a sobrecarga imposta por várias máquinas virtuais disputando o mesmo dispositivo, mesmos assim, o desempenho ainda não é comparável o obtido pelas CPUs.

Como foi visto na [subseção 2.5.5](#), conceitos como virtualização total, paravirtualização, e virtualização assistida por *hardware* são fundamentais para entender o funcionamento dos hipervisores. Os hipervisores modernos são o resultado da combinação de todas essas tecnologias já mencionadas, por exemplo, no trabalho de Nakajima (2007) foi apresentado um protótipo que utiliza várias tecnologias juntas interagindo para a melhora do desempenho na virtualização de sistemas operacionais.

Segundo Waldspurger e Rosenblum (2012), há desafios fundamentais para processos de entrada e saída, procuram a obtenção de benefícios com o mínimo de sobrecarga. Uma série de abordagens inteligentes para *hardware* e *software* foram desenvolvidas para atingir níveis de alto desempenho, incluindo a paravirtualização e mudanças nos dispositivos de *hardware*. O gerenciamento de recursos, agendamento, e priorização de tarefas, tornam-se importantes quando existe uma multiplexação de recursos de *hardware* para tarefas com prioridades diferentes. Definição de interfaces limpas e semânticas adequadas para dispositivos virtuais são também desafios.

Existe um grande esforço na condução de pesquisas a fim de melhorar efetivamente a gestão de dispositivos de armazenamento secundário (GE; DENG; FANG, 2010), o que inclui soluções como: NAS (*Network Attached Storage*), *FiberChannel* e *Multipath I/O*. Esses esforços incluem uma gama enorme de trabalhos em sistemas distribuídos e em *fabrics* que conectam os componentes físicos com os virtuais. O sucesso comercial de sistemas baseados em máquinas virtuais dependerá do abastecimento da demanda por otimizações de virtualização com novas capacidades de I/O (WALDSPURGER; ROSENBLUM, 2012).

2.6 Bench4Q

A finalidade de um *benchmark* Web é medir o desempenho do sistema funcionando acima de uma plataforma experimental, por exemplo, uma nuvem pública. Normalmente estão compostos de uma aplicação Web conectada a um banco de dados e um gerador de carga de trabalho. Alguns dos *benchmarks* que são comumente usados nas pesquisas em computação em nuvem são RUBiS (CECCHET, 2009), que trata de um protótipo de um site de leilões, similar ao sistema *eBay*, que oferece funcionalidades básicas como: venda, navegação e de licitação, também suporta três tipos de sessões de usuário: visitante, comprador e vendedor. Compõe-se de várias tecnologias que incluem: Apache *load balancer*, servidor de aplicações JBoss e o MySQL.

Bench4Q implementa uma nova metodologia para *benchmarking* orientada a QoS de sistemas *e-commerce* (ZHANG *et al.*, 2011a). Tem características para produzir de maneira flexível e controlada cargas de trabalho complexas baseadas em sessão e para simular o comportamento do cliente de uma maneira realística. Além disso, oferece uma maneira simples de instalação, configuração e análise de métricas. O software estende várias características da especificação *Transaction Processing Performance Council-Web* (TPC-W)¹⁰ para medir o QoS dentro de infraestruturas, especialmente de simulação de carga e análise de métricas. Entre as características do *benchmark* estão:

- Tratamento de múltiplas sessões Web;
- Geração de páginas dinâmicas com acesso e atualização a banco de dados;
- Bancos de dados constituído de várias tabelas com uma grande variedade de tamanhos, atributos e relacionamentos;
- Gerenciamento e contenção no acesso aos dados pela simultaneidade nas requisições.

Figura 20 – Arquitetura distribuída do Bench4Q.

Fonte: Adaptada de Zhang *et al.* (2011a).

¹⁰ <<http://www.tpc.org/tpcw/default.asp>>

Uma métrica herdada do TPC-W é a *Web Interaction Per Second* (WIPS). O problema dessa métrica é que um cliente geralmente precisa de uma série de interações Web que conformam uma sessão, por exemplo, para culminar um processo de compra. Assim, para um sistema de *e-commerce*, métricas baseadas em sessões são mais significativas do que as métricas baseadas em interações Web, porque uma sessão incompleta vai trazer um impacto representativo para o cliente de sistemas *e-commerce*. Em consequência, um valor alto de WIPS não implica necessariamente elevados lucros. O Bench4Q fornece métricas próprias para avaliação de desempenho usando as sessões de usuários: *Valid Sessions Per Second* (VSPS) representa a média de sessões válidas concluídos por segundo durante um experimento de *benchmarking*; *Valid Session Ratio* (VSR), representa a razão de sessões válidas com respeito ao total das sessões durante um *benchmarking*. Ambas métricas baseadas em sessão são sensíveis a QoS.

A ferramenta é composta de três partes: Console, Agente e SUT (*System Under Test*), conforme apresenta a Figura 20, e também disponibiliza interfaces para o monitoramento de recursos para o servidor de aplicação e para o banco de dados, este monitoramento incluem CPU, memória, rede, etc.

Console

O *Console* possui uma interface GUI que de maneira gráfica, permite a configuração e execução dos experimentos de *benchmark*. A Figura 21 apresenta algumas das características relacionadas com o gerenciamento de Agentes, cada Agente possui um entorno de configuração individual.

Desde a perspectiva arquitetural, a Figura 20 mostra como o *Console* interage diretamente com os Agentes, essa comunicação é realizada através da troca de mensagens pelo protocolo RMI, tanto resultados, como os comandos de controle são enviados pelo canal fornecido pela plataforma de comunicação.

O console está encarregado de coordenar a execução dos experimentos, define o número de EBs gerados em cada Agente, assim como o comportamento e duração de cada grupo de EBs. Além disso, permite definir o tempo de duração do experimento e os parâmetros de monitoração para os servidores de aplicação e de banco de dados.

Finalizado o experimento, o *Console* requisita dos Agentes os relatórios parciais resultado da submissão da carga de trabalho sobre o sistema SUT. Essas informações servem para gerar relatórios e gráficos globais utilizando as métricas definidas no *benchmark*.

Agentes

Os agentes são uma parte fundamental do *benchmark*, porque encarregam-se da geração da carga de trabalho propriamente dita de maneira distribuída com as configurações que foram previamente definidas no *Console*. Supondo um conjunto de agentes definidos como $A = a_0, a_1, \dots, a_{n-1}$, cada a_i corresponde a um EB que vai gerar requisições com os parâmetros definidos pelo *Console*. Os resultados das requisições de cada EB são

Figura 21 – Interface gráfica do módulo Console usado pelo Bench4Q.

Fonte: Adaptada de [Zhang et al. \(2011a\)](#).

registradas nas estruturas internas de cada Agente. Cada EB é independente e só termina quando uma variável do *fim* do experimento muda de valor. Cada requisição pode ter estados de sucesso (com o tempo que demorou em ser atendida) ou erro (com o tipo de erro HTTP). Finalizado o experimento os agentes geram um relatório local simplificado com todos os resultados das requisições geradas. Finalmente, quando recebida a ordem do *Console*, os agentes retornam essas informações para o *Console*. Um agente sempre está disponível para realizar mais requisições após uma ordem vinda do *Console*.

EB - *Emulated browsers*

Clientes Web são emulados através de EBs. No Bench4Q os EBs são implementados através *threads*. Virtualmente, quando uma *thread* é iniciada, uma sessão de usuário é iniciada no *benchmark*. Pode-se dizer que cada um das *threads* está associada com uma sessão única e o tempo da sessão está restringido ao tempo de vida da *thread*. Os EBs realizam requisições Web para o SUT numa sequencia definida por um modelo de probabilidades. O SUT fornecido pelo Bench4Q disponibiliza um site de compra de livros

Figura 22 – Exemplo de CBMG.

Fonte: Adaptada de [Zhang et al. \(2011a\)](#).

com 14 páginas diferentes, as probabilidades de que uma *thread* solicite uma determinada página durante seu tempo de vida estão definidas pelo modelo CBMG (*Customer Behavior Model Graph*) ([MENASCÉ et al., 1999](#)). A [Figura 22](#) apresenta um grafo orientado que descreve o comportamento das *threads* (clientes) com padrões de navegação bem definidos.

Atividades relacionadas como acessar a página inicial (*Home*) do SUT, navegar e procurar por produtos, realizar operações que envolvam o carrinho de compras e finalizar uma compra, todas elas estão controladas pelo CBMG. São distinguidas três tipos de operações: *procurar*, *navegar* e *comprar*. Dentro das arestas do CBMG há pesos que representam a probabilidade de transição de um site para outro ([MENASCÉ et al., 1999](#)).

No contexto do EBs na sessão fechada, a geração de carga de trabalho está relacionada com dois possíveis resultados para cada requisição a) requisição com sucesso, o tempo que demorou a requisição desde sua submissão até a chegada do resultado é registrado; b) requisição com erro, quando o resultado da requisição é um código de erro, a requisição é registrada com o erro respectivo e o tempo que estava sendo medido é descartado. Todas as requisições de uma sessão, fora da inicial, estão restringidas por um tempo de espera ou *think time*, esse tempo é utilizado para a geração de todas as requisições menos a primeira, que é executada imediatamente após ser criada. O *think time* é um valor gerado estocasticamente a partir de um intervalo definido no *Console*. O processo de execução de uma nova requisição depende da chegada da resposta anterior mais o *think time* gerado.

Sessão fechada

É uma implementação definida no TPC-W para a geração de carga no processo de *benchmarking* do Bench4Q. Na [Figura 23](#), os clientes simulados ou EBs que são gerenciados

Figura 23 – Geração de sessões definidas no TPC-W.

Fonte: Adaptada de [Zhang et al. \(2011a\)](#).

pelos Agentes, são inicializado com parâmetros definidos previamente no Console. Quando as atividades de um determinado EB termina após o recorrido da rota no grafo CBMG ([MENASCÉ et al., 1999](#)) a sessão é terminada e uma nova sessão é inicializada através de uma nova *thread* que irá iniciar seu próprio recorrido. O número total de clientes (EBs) é mantido durante todo o experimento até atingir o tempo máximo do experimento.

Sessão aberta

A abordagem de sessão aberta para a geração de carga de trabalho é uma das contribuições do Bench4Q, representa uma maneira simples de simular geração de rajadas na carga de trabalho. Na sessão fechada existe uma restrição no número de EBs utilizados para a geração de carga de trabalho. Na sessão aberta esse limite não existe, a lógica é inicializar certo número de EBs em determinados pontos do experimento. Podemos definir esses pontos pelo conjunto T , tal que $T = t_0, t_1, \dots, t_{i-1}$, cada t_i representa a hora em que os grupos de EBs serão inicializados. Se os valores de t_i estão muito juntos teremos uma rajada pronunciada, se estiverem espalhados não existirá uma rajada representativa. Dentro da sessão aberta, cada EB realiza o roteiro tradicional definido pelo grafo CBMG, finalmente a sessão termina.

A [Figura 24](#) apresenta um esquema geral de como é realizada a geração de carga utilizando a abordagem de sessões abertas. A implementação realizada por [Zhang et al. \(2011a\)](#) na versão 1.2 do *benchmark*, obedecendo um intervalo *fixo* de tempo para a inicialização de grupos de EBs por cada Agente.

SUT

O *Software Under Test* (SUT), compreende todos os componentes que fazem parte da

Figura 24 – Extensão do Bench4Q para o TPC-W na geração de sessões.

Fonte: Adaptada de [Zhang et al. \(2011a\)](#).

aplicação que está sendo simulada. Isso inclui conexões de rede, servidores Web, servidores de aplicações, servidores de banco de dados, etc.

Algumas das funções que o SUT realiza são: solicitações HTTP GET através do servidor HTTP, ou retornando informações estáticas a partir de arquivos armazenados no sistema de arquivos ou através da geração de páginas dinâmicas utilizando informações obtidas a partir do banco de dados; solicitações HTTP POST através do servidor HTTP, invocando a transação adequada no banco de dados. Em geral o SUT controla todos os acessos ao banco de dados, todas as funções de comunicação com os clientes geradores de carga e toda a funcionalidade necessária para implementar as interações Web da aplicação.

O SUT fornece um conjunto de imagens definidas pelo padrão de referência TPC-W, são imagens de diferentes tamanhos no formato JPEG. Os tamanhos definidos pela especificação são: 5 KB, 10 KB, 50 KB, 100 KB e 250KB. As imagens são destinadas a representar o fluxo de dados de mídia diferente na Web, tais como imagens, áudio, vídeo, fluxo de GIF animado, etc. Cada uma dessas imagens é gerada utilizando o gerador de imagem TPC-W (disponível no site da TPC ¹¹). A distribuição para os diferentes tamanhos de imagem são:

- 5 KB em 45%
- 10 KB em 35%
- 50 KB em 15%
- 100 KB em 4%
- 250 KB em 1%

O banco de dados utilizado pelo SUT deve estar definido no mínimo de oito tabelas de base individuais. As relações entre essas tabelas são definidos no diagrama de entidade relacionamento da [Figura 25](#).

¹¹ <http://www.tpc.org/>

Figura 25 – Entidades do banco de dados, relacionamentos e características.

Fonte: Adaptada de [Menasce \(2002\)](#).

METODOLOGIA

A metodologia para a condução do trabalho baseia-se em um estudo de caso generalizável, cujo propósito é desenvolver o método proposto apoiado em um problema-objeto que ressalta, de modo concreto as noções abstratas elaboradas. O caso selecionado atende aos critérios de evidenciar a) o problema de avaliação de desempenho; b) como esse problema se traduz em requisitos para projeto de experimentos de *benchmarking*; c) como esses requisitos podem ser mapeados em termos do modelo de responsabilidades MEDC para o projeto de um *framework* de *benchmarking* não-estacionário.

O estudo de caso aborda um domínio relevante em que a contribuição metodológica do trabalho é pertinente; sua aplicabilidade, contudo é generalizável a outros problemas com características semelhantes. Através da instanciação concreta dessa proposta, o *software* e demais artefatos produzidos a partir desta pesquisa podem ser considerados contribuições práticas que exemplificam a metodologia proposta.

3.1 Fundamentação teórica e prática

O presente trabalho baseia-se na noção abstrata de *desempenho* como uma avaliação quantitativa de quão bem a *capacidade* de um sistema é capaz de absorver a *demand*a recebida — todas as quantidades definidas genericamente. Com respeito aos vários propósitos da engenharia, a capacidade do sistema $C(\cdot)$ pode ser conceitualmente formulada como uma função que relaciona os sinais de *demand*a $d(t)$ e *desempenho* $p(t)$ tal que pode ser denotado por $p(t) = C(d(t))$. Embora muitas vezes não expressamente distinta, a análise *estacionária* torna-se implícita. Sobre uma típica carga de trabalho estocástica, uma abordagem experimental adequada consiste em medir os sinais de entrada e saída por algum tempo, descartando efeitos de inicialização, e depois aplicar técnicas de regressão para parametrizar um modelo analítico adequado. A razão subjacente para garantir um ponto operacional estável é que, durante a inicialização, a dinâmica do sistema pode ser expressiva, por exemplo, devido ao enchimento de *buffers*, instanciação

de recursos etc. Capturar as propriedades dinâmicas dos sistemas, por sua vez, é precisamente o objetivo da análise *transiente* (não-estacionária). Segundo esta abordagem, a capacidade é para ser modelada como um sistema *dinâmico*, matematicamente correspondente a uma equação diferencial (ou de diferenças, no domínio do tempo discreto). Formalmente, podemos então denotar capacidade $C(\cdot, t)$ e, assim, $p(t) = C(d(t), t)$, para indicar que desempenho em qualquer instante não depende unicamente da demanda imediata, mas também do tempo, dado que o comportamento inercial faz com que as alterações produzidas pela entrada apareçam na saída com um certo atraso e variando com o tempo, mesmo que a demanda seja mantida constante.

Figura 26 – Parâmetros de desempenho dinâmico.

Fonte: Júnior *et al.* (2015).

A Figura 26 ilustra um exemplo no qual um sistema encontra-se originalmente no ponto de operação inicial em que exibe certo desempenho $p(t)$ estacionário. No instante t_d a carga de trabalho $w(t)$, inicialmente estacionária, é abruptamente alterada. Se, por exemplo a métrica de carga é o número de requisições por segundo que chega ao sistema, e a métrica de desempenho é o tempo de serviço das requisições que deixam o sistema, o gráfico mostra um cenário em que, logo após uma elevação brusca de carga, o tempo de serviço experimenta picos transitórios antes de estacionar no novo patamar. A função que descreve como o estado estacionário varia em função da chegada é comumente chamado de ganho *Direct current gain* (DC)¹ portanto um parâmetro de desempenho estático. O máximo valor de pico atingido relativo ao valor estacionário final denominado *overshoot*, por sua vez, é um parâmetro de desempenho dinâmico. O tempo de pico (*peak time*) e tempo de assentamento (*settling time*), indicados na Figura 26, são outros exemplos de parâmetros de desempenho dinâmicos.

¹ *Direct current gain*, tem origem na análise de sinais de sistemas elétricos e refere-se à relação entre o ganho (amplificado) da saída do sistema quando sujeito a uma corrente contínua (*direct current*) ou seja, estacionária.

O protocolo experimental para obter um modelo empírico de desempenho estacionário é relativamente simples:

- a) Selecionar os parâmetros de demanda e desempenho (SISO);
- b) Trazer o sistema para um ponto operacional de repouso Q_o pela utilização de uma carga de trabalho estacionária;
- c) Medir o desempenho estacionário P_o após fim dos transitórios;
- d) Repetir os passos (b) e (c) dentro de um ponto de repouso diferente Q_1 para obter o d_1 e p_1 ;
- e) O ganho DC é a razão $K_{DC} = \frac{p_1 - p_0}{d_1 - d_0}$

Em que p_1 , p_0 , w_1 , w_0 são valores médios.

Se repetidas as execuções deste protocolo, geram os mesmos valores para o ganho DC, então o parâmetro K_{DC} é invariante no tempo. Se, ainda, a execução do protocolo em diferentes pontos de repouso leva também ao mesmo ganho, esta é uma indicação de que o sistema é linear (ou pode ser aproximado por um modelo linear), e K_{DC} é uma constante tal que $p(t) = K_{DC} \cdot d(t)$.

As necessidades industriais e de pesquisa têm motivado a criação de vários *frameworks* para avaliação de desempenho adequados na realização desse tipo de experimento. O protocolo para obter um modelo dinâmico do sistema, por outro lado, é diferente. Em primeiro lugar, a medição não é feita no regime estacionário. Logo, é necessário que o conjunto de ferramentas experimentais permitam a aplicação de uma perturbação controlada na demanda capaz de provocar, no desempenho, a aparição do transiente, de modo a expor a dinâmica do sistema. Em segundo lugar, os dados necessários para ajustar um modelo de capacidade não são limitados à média temporal, mas a uma sequência de medições ao longo do tempo. Finalmente, o modelo de capacidade não é obtido por uma relação estatística $\frac{\Delta p}{\Delta d}$, mas a partir da solução da equação diferencial (ou diferença), dada que as séries temporais de $p(t)$ e $w(t)$ são analisadas e não somente sua média de tempo.

Infelizmente, as soluções para sistemas de equações diferenciais não são genéricas e para muitos problemas elas são matematicamente desconhecidas. Por isso, para fins de engenharia, geralmente é conveniente considerar a hipótese linear e invariante no tempo (LTI). Mesmo se a relação entre duas variáveis, $y(x)$ e x não é linear, muitas vezes é possível assumir que elas são lineares na vizinhança de um ponto de repouso X_0 , e usar uma aproximação LTI dentro deste intervalo como é ilustrado na [Figura 27](#). Isso é chamado de análise de pequenos sinais.

3.1.1 Modelagem dinâmica

Enquanto *benchmarking* de desempenho em estado estacionário concentra-se em estatísticas agregadas, tais como o rendimento, fração de solicitações não atendidas, tempos de resposta,

Figura 27 – Intervalo linear na aproximação LTI.

Fonte: Elaborada pelo autor.

Figura 28 – Representação no domínio do tempo.

Fonte: Elaborada pelo autor.

calculados como totais ou médias durante todo o experimento, as preocupações da avaliação de desempenho transitória preocupa-se com a forma como estes valores variam ao longo do tempo. Formalmente, o valor de uma quantidade ao longo do tempo é referido como um *signal*, e arcabouço da engenharia para estudar suas propriedades e transformações está firmemente fundamentada na base matemática da teoria de análise de sinais. Uma revisão em profundidade dos fundamentos da análise e modelagem de sinal de sistema dinâmico não caberiam nas restrições do espaço de estudo do presente trabalho. No entanto, por uma questão de esclarecer as razões essenciais da metodologia de *benchmarking* não-estacionário introduzidas, é válido rever alguns conceitos básicos.

A Figura 28 descreve uma representação gráfica da definição genérica de desempenho já mencionado como uma função da carga de trabalho, dado pela capacidade inerente do sistema.

Figura 29 – Representação no domínio da frequência.

Fonte: Elaborada pelo autor.

Nos diagramas de blocos, o sistema está dividido em dois componentes que afetam o desempenho global. Na parte superior, estes dois componentes são representados por modelos estáticos obtidos por uma abordagem de análise estacionária. O diagrama de blocos expressam a transformação do sinal de entrada, uma vez que atravessam os dois componentes em série. Em um modelo estático, se $f(\cdot)$ e $g(\cdot)$ são conhecidos, o efeito combinado dos dois componentes pode ser explicitamente escrito na sua forma analítica, tal que o sinal de desempenho se traduz em uma função da carga de trabalho dada pela composição $p(t) = f \circ g(t)$, tal que, podemos escrever $p(w(t))$ analiticamente. Contudo, se as propriedades dinâmicas² forem consideradas, precisamos denotar $f(\cdot, t)$ e $g(\cdot, t)$, como na Figura 28b. A contribuição do estado interno para a saída pode ser associado a uma equação diferencial (ou equação de diferenças no domínio do tempo discreto). Infelizmente, nem sempre é possível determinar uma formulação analítica para $f(\cdot, t)$ e $g(\cdot, t)$, de modo tal que a composição $f \circ g(\cdot, t)$ não pode ser derivada analiticamente através de um método geral. Existem soluções, no entanto, para a classe de funções que satisfazem as propriedades de linearidade e invariância no tempo. Por esta razão, aproximações lineares invariantes no tempo (LTI) são amplamente utilizadas em abordagens de engenharia para a modelagem de sistemas dinâmicos. O arcabouço matemático clássico para lidar com sistemas LTI, utiliza da transposição dos modelos de domínio do tempo para o domínio da frequência de Laplace. Neste domínio, a variável t correspondente ao tempo contínuo, torna-se implícita passando a se denotar como a variável s explícita, representando a frequência complexa contínua — ou pela variável z , representando a frequência discreta, correspondente ao tempo discreto variável k . Ambos modelos, estáticos e dinâmicos, são então englobados pelo mesmo modelo no domínio da frequência como na Figura 29.

Uma vantagem prática de tais transformações, é que as equações diferenciais no domínio do tempo são convertidas em equações algébricas. Assim, por exemplo, se ambos os sistemas são LTI de primeira ordem, teríamos no domínio do tempo o sistema de equações diferenciais

² Um sistema dinâmico é aquele cuja saída depende não só da presente entrada, mas também das entradas anteriores. Intuitivamente, se o sistema tiver inércia, pode produzir uma saída diferente de zero, mesmo se a presente entrada é nula como resultado da energia armazenada nos estados internos.

ordinárias:

$$\begin{cases} \frac{dy(t)}{dt} + a \cdot y(t) = w(t) \\ \frac{dp(t)}{dt} + b \cdot p(t) = y(t) \end{cases} \quad (3.1)$$

com a e b constantes e parâmetros dos respectivos sistemas. No domínio da frequência, por outro lado, temos o equivalente algébrico

$$\begin{aligned} P(z) &= F \circ G(z) = F \cdot Z(z) = H(z) \\ \text{com } H(z) &= F \cdot G(z) \\ P(z) &= F \cdot G \cdot W(z) \end{aligned} \quad (3.2)$$

que pode ser resolvido diretamente para $P(z)$; a expressão do domínio do tempo de $p(t)$ pode ser obtida pelo inverso da transformada Z . Os modelos de domínio de frequência $F(z)$ e $G(z)$ são referidos como *funções de transferência*. O composto de duas funções de transferência é apenas a sua multiplicação algébrica, de modo que a função de transferência resultante do exemplo contínuo é $H(z) = F \cdot G(z)$.

Outra conveniência da modelagem com a transformada Z é que os métodos no domínio da frequência de análise de sinal clássica podem ser diretamente utilizadas para tirar conclusões importantes sobre o sistema, o que pode revelar-se consideravelmente mais difícil de obter na abordagem no domínio do tempo. Por exemplo, a técnica de identificação indutiva empírica sistematizada na metodologia aqui introduzida baseia-se nos recursos teóricos de análise de sinais.

3.2 Modelo de capacidade dinâmica

A fim de obter empiricamente o modelo de capacidade dinâmica um protocolo consiste em uma sequência de passos:

- Selecionar os parâmetros de desempenho e demanda (SISO);
- Ajustar o ponto operacional de repouso;
- Verificar a linearidade e análise de pequenos sinais;
- Aplicar ao sistema uma perturbação do tipo degrau (Figura 1a) e medir o desempenho;
- Aplicar compensações e normalizações para que o sinal de desempenho represente a resposta de uma entrada de degrau unitário na condição inicial zero;
- Aplicar aos sinais discretos um filtro passa-baixo para remover o ruído estocástico;

Figura 30 – Arquitetura DES não-estacionária

Fonte: Júnior *et al.* (2015).

- Selecione a ordem mais baixa do modelo dinâmico (parcimônia) capaz de capturar a dinâmica do sistema (primeira ordem se o transiente é monotônico, segunda a ordem se apresenta oscilação estável);
- Usar sinais filtrados para ajustar um modelo auto-regressivo correspondente com o modelo de entrada exógena (ARX);
- A partir do modelo ARX derive uma função de transferência (TF). A expressão TF é uma aproximação analítica empírica da dinâmica do sistema que pode ser validada por comparação com a saída do sistema (simulação).

Em grande parte, os atuais *frameworks* de avaliação de desempenho de sistemas computacionais são originariamente destinados à análise estacionária. O presente trabalho de pesquisa investiga abordagens para estender metodologias e ferramentas para uso na avaliação de desempenho não-estacionário. Uma das contribuições propostas nesta área é uma abordagem sistemática para extrair modelos dinâmicos empírico-analíticos de desempenho de sistema computacionais a partir de dados experimentais obtidos do *benchmark*. Resumindo, a abordagem baseia-se no modelo de referência para a avaliação de desempenho não-estacionário, um protocolo experimental para adquirir dados de identificação, e um método de análise no tempo para obter um modelo dinâmico empírico-analítico. O modelo de referência MEDC separa as preocupações do desenvolvedor do experimento em um conjunto de responsabilidades que compreendem as modulações da demanda e capacidade ao longo do experimento, as variáveis de monitoramento no tempo e a modelagem da dinâmica do *efetor* do sistema. A arquitetura proposta MEDC (*monitor*, *effector*, *demand*, *capacity*) é descrita no diagrama de classes da Figura 30. Para mais detalhes pode ser consultado o trabalho (JÚNIOR *et al.*, 2015).

3.3 Análise de resultados e avaliação

O desenvolvimento da metodologia proposta é feito apoiado em um estudo de caso que ilustra concretamente os conceitos abstratos abordados. Esse estudo tem como contribuição o projeto de um *framework* de *benchmark* para avaliação de desempenho não-estacionária. A aplicabilidade desse resultado pode ser investigada por meio da utilização do *framework* em um experimento real de modelagem de capacidade dinâmica de um sistema computacional, comparando-se os resultados do modelo produzido com observações empíricas. A capacidade de expor a dinâmica do sistema em estudo é o objetivo do *benchmark* transiente.

Um estudo concreto da discussão anterior pode se apoiar em um estudo de caso do mundo real decorrente de necessidades práticas de um trabalho de pesquisa realizado no grupo de pesquisa. O referido projeto de pesquisa centra-se em aproximações analíticas para modelar a dinâmica do desempenho de um *data center*, que hospeda uma aplicação IaaS³ da computação em nuvem habilitada para oferecer QoS. Como parte desse trabalho, foi necessário derivar um modelo analítico que representasse como o sistema reage às mudanças sutis nas condições de operação. A fim de verificar isto experimentalmente, o processo indutivo implica aplicar uma carga de trabalho representativa sobre o sistema para medir os efeitos observáveis sobre o desempenho entregue.

Um *benchmark* extensivamente referenciado na literatura é o Bench4Q (ZHANG *et al.*, 2011a), que apresenta uma extensão da especificação TPC-W (MENASCE, 2002). O *benchmark* é flexível e rico em recursos, mas o *workflow* experimental, no entanto, é voltado para análise estacionária. As principais características do Bench4Q incluem apoio à análise de métricas baseadas em sessão e simulação de carga sensível a QoS para análise da capacidade.

Com a finalidade de expor as propriedades de desempenho dinâmico do sistema, foi necessário estender algumas características do Bench4Q modo a dotá-lo das capacidades MEDC. O resultado deste trabalho, aqui apresentado, é o conjunto de ferramentas **OnlineBench4Q**. Essencialmente, as facilidades fornecidas permitem especificar padrões de carga de trabalho não-estacionárias, políticas de dimensionamento de recursos e monitoramento em tempo real de parâmetros de desempenho.

3.4 Materiais e métodos

O estudo de caso é baseado no *software* Bench4Q, um *benchmark open source* distribuído sob a licença *GPL v2.1*. O sistema é desenvolvido na linguagem Java. A implantação do sistema foi realizada nas instalações do LaSDPC e utilizou-se o *hardware* e *software* descrito no ??.

Todos os artefatos produzidos pela pesquisa são distribuídos como *open source* sob a

³ Infraestrutura como serviço: um modelo de serviço da computação em nuvem onde máquinas virtuais são oferecidas como recursos.

licença *GPL* v3.

ONLINEBENCH4Q

A fim de materializar essa ilustração, o estudo baseia-se em um exemplo prático em que o *framework* OnlineBench4Q foi utilizado para revelar e modelar a dinâmica de uma aplicação de *e-commerce* implantada sobre uma infraestrutura de computação em nuvem.

O estudo de caso será contextualizado inicialmente com um exemplo da utilização da metodologia em um contexto de simulação que utiliza a avaliação de desempenho não-estacionária, e seguidamente é explicado o estudo de caso utilizando o OnlineBench4Q.

4.1 Problema-exemplo

Um dos trabalhos em andamento do grupo de pesquisa do autor trata da arquitetura de um *broker* para gerenciamento de recursos em ambiente de computação em nuvem (NOBILE, 2013). A abordagem leva em consideração que o cliente deseja, por um lado, prover um serviço com o melhor desempenho possível para seus usuários, com o objetivo de manter sua reputação e manter sua fidelidade, porém, por outro lado, a custo mínimo, consumindo apenas a quantidade suficiente de recursos (Máquinas Virtuais — MVs) no *datacenter* para manter seu serviço. O lucro ótimo corresponde ao equilíbrio eficiente dos níveis de serviços acordados (*Service Level Agreement* — SLA) nos dois lados, usuários e *datacenter*. A solução investigada neste trabalho baseia-se na mesma estratégia explorada em outros trabalhos relacionados (LIM *et al.*, 2009; AN *et al.*, 2010; YAZIR *et al.*, 2010; ZHANG *et al.*, 2011c), que focam em manter a taxa de utilização das máquinas virtuais provisionadas a um nível desejado. A motivação dessa técnica é o fato de que uma taxa de utilização baixa pode indicar que algumas máquinas podem ser desligadas, ao passo que se ela estiver alta, um aumento abrupto na demanda (e. g. pico de requisições), pode fazer com que haja a sobrecarga no sistema e SLAs sejam violados no lado dos usuários. Como encontrar uma taxa de utilização ótima é um problema além do escopo deste trabalho e pode ser tratado por métodos de otimização.

Seja o caso de um cenário em que para maximizar a receita de um cliente, uma política otimizada exige que os objetivos de controle devam manter a taxa de utilização das máquinas virtuais a 70%, denominado ponto de operação, de tal maneira que na média as máquinas virtuais ficarão ociosas 30% do tempo. Muito embora esses valores sejam arbitrários, eles possuem um sentido prático, pois aproveita consideravelmente os recursos dedicados ao sistema e permite a absorção de variações na demanda.

Intuitivamente simples para esse problema, uma ideia simplista seria implementar um mecanismo de reação proporcional: continuamente medir a taxa de utilização do sistema e aumentar ou diminuir a alocação de máquinas virtuais proporcionalmente. Se essa reação for suficiente para manter a taxa de utilização no ponto de operação imediatamente após uma perturbação, o restante desta discussão é não é necessária. Porém, este não é o caso no exemplo dado, pois o processo de alocação de novas máquinas virtuais pelo hipervisor não é instantâneo, uma vez que há a presença do tempo de *boot*, característica inevitável e que adiciona um atraso não negligenciável na execução das ações tomadas pelo controlador. Em um cenário em que a utilização em um determinado instante de tempo é sutilmente acima do ponto de operação e o provisionamento de uma nova máquina virtual é suficiente para diminuir a utilização para abaixo do ponto de operação, então o resultado esperado é que, mesmo com uma carga de trabalho estocástica estacionária, um controlador adaptativo proporcional pode resultar em um comportamento de ligar e desligar máquinas virtuais, o que é claramente ineficiente.

Uma estratégia razoável para mitigar o indesejado processo de oscilação de ligar e desligar, é desacelerar a política de reação proporcional. Isso pode ser implementado por um acumulador ponderado que, a cada ciclo de amostragem, mede a taxa de utilização instantânea e a compara com o ponto de operação, adicionando a diferença ao acumulador. Caso a diferença seja positiva (recursos insuficientes), o acumulador é incrementado com um valor; se for negativa (excesso de recursos), ele é decrementado. Com o fator ponderante menor do que 1, percebe-se um efeito de filtragem a variações súbitas na carga de trabalho, uma vez que a diferença na taxa de utilização gastará alguns ciclos de amostragem para que se configure uma situação de alteração do valor inteiro (*integer value*) e, portanto, disparar o mecanismo de alocação de recursos.

Essas duas abordagens de controladores *ad hoc* podem analisadas por meio de simulação. Entretanto, há uma sólida base de teórica para análise de transiente e controle dinâmico, já estabelecido em outras áreas da ciência e engenharia, que podem ser utilizados no presente domínio de aplicação. Com a abordagem de modelagem de sistemas dinâmicos, seria possível obter conclusões relevantes com menos esforço e adicionalmente, revelar mais sobre o comportamento esperado do sistema sob variação na carga de trabalho. Esse tipo de modelo provê ferramentas não somente para análise, mas também para o projeto de estratégias que atendam a requisitos de desempenho transientes, tais como degradação do desempenho após uma perturbação na carga de trabalho e o máximo tempo de reajuste até que o novo nível de desempenho seja conseguido.

Nessa perspectiva, é possível especificar o máximo *overshoot* e o máximo tempo de assentamento (*settling time*) que o sistema deve exibir após um aumento súbito na carga de trabalho, bem como especificar os objetivos de controle dado um conjunto de requisitos.

Figura 31 – Uma malha fechada de controle integral.

Fonte: Mamani *et al.* (2015).

No jargão de teoria de controle clássica, o mecanismo de acumulação ponderada desempenha a função de um controlador integral (pois integra os valores das amostragens no tempo). O diagrama de blocos da Figura 31 representa o caminho dos sinais de entrada e saída pelos componentes dos sistemas conectados em um laço de controle retroalimentado. Os elementos são:

- G é o sistema a ser controlado, ou planta;
- C é o controlador, responsável por transformar o erro do sinal em força restauradora no sistema;
- R é o sinal de referência, ponto de operação desejado;
- U é a saída, a taxa de utilização no exemplo utilizado;
- E é o sinal de erro, a diferença entre o ponto de operação e a saída medida, idealmente deve ser zero;
- N é o número de máquinas requisitado pelo sistema;
- H é um filtro de média móvel exponencial $H(z) = \frac{(1-c)}{(1-c \cdot z^{-1})}$;

Neste trabalho assume-se que sistemas dinâmicos podem ser aproximados por sistemas lineares discretos invariantes no tempo, de tal forma que podem ser abordados por meio de técnicas de controle clássico. O sistema é modelado no domínio Z (o equivalente discreto da transformada de Laplace no domínio contínuo).

O objetivo é encontrar parâmetros K_i para controladores de tal forma que, caso o usuário deseje alterar a taxa de utilização de 70% para então 50%, mudando-se o ponto de referência

abruptamente, então a saída deve ir de $u(t_0) = 70\%$ para $u(t_1) = 50\%$ em um intervalo $\delta t = t_1 - t_0 \leq T_s$ (tempo de assentamento) e durante esse processo, $u(t)$ não deve ultrapassar $u(t_1)$ por mais de um certo valor de *overshoot*.

O parâmetro K_i é determinado pelo método do lugar das raízes (*root-locus*) (OGATA, 2010) para que a estabilidade e desempenho requisitados sejam garantidos. Para essa finalidade, o primeiro passo é obter o modelo para a planta G (o *datacenter* e a aplicação em nuvem). Deduções analíticas para encontrar os componentes inerciais de um sistema é uma tarefa complexa. Por isso, utiliza-se o método caixa preta, aplicando-se uma entrada conhecida, medindo-se a saída correspondente e identificando-se um modelo empírico capaz de representar o sistema.

Como exemplificação, pelo princípio da parcimônia, a aplicação G é aproximada para um modelo discreto de primeira ordem, linear e invariante no tempo. Para tal finalidade, uma parametrização caixa-preta similar àquela utilizada em (PAREKH *et al.*, 2002) foi aplicada e um modelo auto-regressivo com entradas exógenas (*auto-regressive with exogenous input* — ARX) e ajustado pelo método dos mínimos quadrados. As Equação 4.1 mostra o modelo ARX estimado e a 5.1 a função de transferência resultante.

$$u(k+1) = 0.57961u(k) - 0.016263n(k) \quad (4.1)$$

$$G(z) = \frac{U(z)}{N(z)} = \frac{-0.016263}{z - 0.57961} \quad (4.2)$$

Com u e U sendo a taxa de utilização nos domínios do tempo e frequência respectivamente e n e N o número de máquinas virtuais que compõem o sistema nos domínios do tempo e frequência, respectivamente. Os parâmetros a e b devem ser obtidos por dados gerados através de experimentos não-estacionários, executados por uma aplicação de *benchmark*, com o seguinte protocolo:

- a) Iniciar o experimento com uma entrada estacionária;
- b) Esperar que os efeitos transientes terminem;
- c) Aplicar um degrau na entrada em um determinado instante de tempo;
- d) Mediar os dados de entrada e saída;
- e) Estimar os parâmetros para o modelo ARX e derivar uma função de transferência equivalente G .

No trabalho de (NOBILE, 2013), na procura da elasticidade, o problema-exemplo considera que máquinas virtuais são ligadas e desligadas de acordo com uma política que pode ser associada a uma máquina de estados. Caso uma máquina virtual esteja operacional, ela recebe

requisições de um balanceador de carga de trabalho, o qual distribui segundo algum algoritmo de balanceamento as requisições entre as máquinas virtuais ativas. Se todas as máquinas virtuais estão operacionais e o gerenciador de recursos necessite reduzir o contingente de máquinas virtuais ativas, algumas máquinas deixam de receber requisições e são movidas para o estado *desligando*, o que significa que elas processarão as requisições pendentes para depois serem desativadas; uma máquina virtual em estado *desligando* pode ser retornada para o contingente de máquinas ativas, caso nesse período seja necessário aumentar novamente o número de recursos da aplicação. Caso o gerenciador necessite de mais recursos e não haja máquinas em estado *desligando*, uma nova máquina é alocada e entra no estado de *inicializando*, estado em que ficará até que esteja pronta para receber requisições e se tornar operacional. O desligamento é implementado em dois passos distintos: a máquina virtual é pausada; após algum tempo de inatividade a máquina é desativada definitivamente. A Figura 32 ilustra o ciclo de vida de uma máquina virtual por meio de seus possíveis estados:

Figura 32 – Diagrama de estados que representa o ciclo de vida de uma máquina virtual.

Fonte: Nobile (2013).

- *Desligada*: a máquina virtual está desligada e não consome recursos físicos no *data center*, apenas armazenamento. Neste estado a máquina virtual pode ser religada caso o gerenciador de recursos julgue necessário;
- *Inicializando*: a máquina virtual está ligada, mas os serviços da MV estão sendo inicializados. Neste a máquina já consome recursos, porém não está disponível para atender requisições;
- *Desligando*: a máquina virtual é considerada desnecessária e é marcada para desligamento assim que possível. A máquina não é desligada imediatamente, porque ainda possuem requisições em fila. Máquinas neste estado consomem recursos, mas não recebem requisições;

4.3 Efetor

O modelo de referencia MEDC define um conjunto de requisitos que são atendidos pelo módulo **Efetor**. Entre as responsabilidades desse modelo incluem-se fornecer os mecanismos de atuação pelos quais as políticas de *Demanda* e *Capacidade* mudam os estados internos do sistema. O *Efetor* também serve como uma camada de abstração extra disponível para o desenvolvedor modelar operações e componentes que afetam a dinâmica do sistema. No exemplo em estudo as responsabilidades atuais do *Efetor* encontram-se a modulação da taxa de requisições que chegam ao sistema e a quantidade de máquinas virtuais. Para gerenciar a dinâmica do sistema relacionado com a capacidade é necessário definir uma abstração acima dos estados reais das máquinas virtuais.

4.3.1 Estados das máquinas virtuais no Efetor

O controle dos recursos é realizado por *hardware* ou *software*; para o caso específico da virtualização de máquinas o gerenciamento é realizado pelo hipervisor, que abstrai o ciclo de vida das máquinas virtuais através de estados como: iniciando, pausado, salvo, etc.

A implementação do *Efetor* pode ser realizada modificando diretamente o hipervisor, ou criando uma camada de abstração adicional que permita gerenciar as máquinas virtuais. A última alternativa é a mais recomendada para tal fim, porque os hipervisores são projetados para fornecer uma *Application program interface* (API) —por exemplo, Libvirt ([LIBVIRT, 2014](#))— para que terceiros construam sistemas de gerenciamento e monitoração de recursos.

A [Figura 32](#) apresenta um diagrama com os estados e o ciclo de vida que o *Efetor* implementa acima dos estados fornecidos pelo hipervisor de máquinas virtuais. A seguir é apresentada uma descrição sucinta dos estados e do fluxo:

Desligado: A máquina virtual existe, está desligada e não consome recursos físicos, exceto o espaço de armazenamento que ocupa a imagem do disco. Uma máquina virtual desligada está disponível para ser ligada e entrar no estado *iniciando*. O tempo de inicialização é variável, mas existe e pode ocasionar problemas no caso de precisar com urgência da máquina virtual.

Iniciando: Quando a máquina virtual encontra-se no processo de inicialização do sistema operacional junto com os serviços instalados. Neste estado a máquina virtual já consome recursos como memória e processamento, mas ainda não está disponível para atender às requisições da carga de trabalho. Para sair do estado a máquina notifica ao gerenciador de recursos que todos os serviços estão prontos para serem utilizados. Para os efeitos da presente pesquisa o processamento gerado pela inicialização não é contabilizado.

Executando: A máquina virtual está ligada e pronta para processar as requisições. Apenas a máquina virtual entra no estado executando, o *Load Balancer* considera que a instância

faz parte do seu conjunto de recursos ou *workers* e encaminhar requisições para serem atendidas. Neste estado o *Monitor* começa a coletar informações da máquina virtual e as repassa para outros módulos da arquitetura.

Parando: Este é um estado intermediário que permite terminar de resolver requisições que estiverem sendo atendidas antes da ordem de desligar ou pausar a máquina virtual. A máquina em estado *Parando*, continuam consumindo recursos de processamento e memória de igual maneira que as máquinas ligadas, mas deixam de receber requisições. Com as requisições pendentes resolvidas, o estado *Parando* permite continuar com o fluxo de desligar ou pausar, neste ultimo, há economia de recursos e ter disponibilidade se a demanda aumentar rapidamente.

Pausado: A máquina virtual está pausada e não consome recursos de processamento, exceto espaço de armazenamento e memória. Uma máquina virtual pausada pode ser reativa rapidamente para atender requisições se o hipervisor considerar adequado. Uma diferença importante com o estado *Desligado*, é o tempo necessário para ter a máquina virtual disponível e passar para o estado *Executando*, esse processo é realizado em menos de um segundo. O estado *Pausado* permite desligar a máquina virtual diretamente e passar para o estado *Desligado*.

4.4 Monitor

O monitor está encarregado de coletar temporalmente, reunir, correlacionar e filtrar dados dos módulos que estão sendo analisados. São utilizados sensores para a coleta de dados por meio de uma interface única. Os detalhes observados através do monitor podem incluir informação de eventos, métricas, configurações, etc. De uma maneira prática as informações podem ser: taxa de requisições ativas, porcentagem de utilização da CPU, entre outros. As informações coletadas pelos sensores são repassadas para o **Monitor** que interpreta, pré processadas e coloca em um nível mais alto de abstração para então serem enviadas para a etapa seguinte no ciclo, a fase de análise e planejamento de **Capacidade** e **Demanda**. Como resultado do fluxo de monitoração e planejamento, tem-se como produto um plano de trabalho que consiste de um conjunto de ações que o **Efetor** deve executar.

A monitoração no *VM Manager* coleta informações do hipervisor focando na utilização da CPU por parte das máquinas virtuais. Foi implementada uma abordagem para obter a média de utilização de todas as máquinas virtuais que é descrita a seguir: Cada máquina virtual é monitorada segundo um intervalo de tempo que podemos definir como $T_{vm} = 0,25$ segundos e o conjunto de máquinas são monitoradas utilizando o intervalo $T_{pool} = 1$ segundo. O intervalo de tempo é diferenciado, porque cada máquina virtual possui uma utilização da CPU individual e diferente uma da outra, por esse motivo foi definido um *buffer* de 3 ou 4 amostras para garantir um valor final em T_{pool} a cada segundo. Como regra foi assumido que sempre $T_{pool} > T_{vm}$ para

efeitos de obter um consolidado da utilização de todas as máquinas virtuais. Vale salientar que somente são monitoradas as máquinas virtuais com estados *Parando* e *Executando*.

A monitoração no *Node balancer* é realizada utilizando um intervalo de tempo T , os dados coletados pelos sensores são a taxa de requisições *ativas*. A diferença de sistemas de simulação, onde o número de requisições submetidas ao sistema é uma quantidade que pode ser medida de maneira simples, em implementações de realizadas para sistemas de múltiplas camadas (GANDHI *et al.*, 2012), a quantidade de requisições é um valor relativo. Se o tempo médio das requisições não é pequeno e acontecer uma perturbação abrupta na quantidade de requisições, o valor da taxa de requisições ativas medida no *Node balancer* vai ser maior do que o número de requisições que chegam ao sistema após a perturbação, porque o sistema terá ainda as requisições da perturbação e as requisições que chegam a cada instante.

4.5 Capacidade e Demanda

Pelas próprias características abstratas do modelo de referencia MEDC, foi necessário focar em algumas funcionalidades do *benchmark* e mapeá-las usando o modelo de referencia. A **Capacidade** esta relacionada com o número de máquinas virtuais que pode ser modulado de maneira local ou remotamente no hipervisor. A **Demanda** está relacionada com a taxa de conexões ativas registradas no *Node balancer* e são geradas pelos Agentes do Bench4Q. Para todos os casos, a métrica utilizada para avaliar o desempenho do sistema é a porcentagem de utilização da CPU das máquinas virtuais. Tanto a *Capacidade* como a *Demanda* podem alterar o sistema em tempo real.

Independente do tipo de abordagem utilizada, a execução de *benchmarks* são necessários para que os desenvolvedores e usuários finais possam conhecer o funcionamento do ambiente, seja *software* e ou *hardware* no contexto da avaliação de desempenho não-estacionaria. Condições básicas para o projeto do *benchmark* são apresentados na Figura 34. Na computação em nuvem as alterações de demanda ou capacidade podem ser de origem interna ou externa.

4.5.1 Modelo de capacidade

O modelo de capacidade visa a avaliação de desempenho considerando mudanças nos recursos disponíveis do sistema, existem vários tipos de recursos, por exemplo: número de máquinas virtuais, tamanho do *buffer*, número máximo de clientes, memória alocada, etc. A *Capacidade* é definida como o número de máquinas virtuais que pode ser modulado de maneira local ou remota. A origem da ordem para mudar a capacidade está relacionada com dos tipos de soluções existentes, sistemas descentralizados enxergam os recurso como um todo e utilizam camadas de hierarquia para a toma de sedição, dentro desse contexto existe uma autonomia local e uma responsabilidade de serviço global (WANG; KANDASAMY, 2012).

Figura 34 – Casos de uso que representa a interação do desenvolvedor ou usuário do *benchmark* com os aspectos definidos pela MEDC.

Fonte: Elaborada pelo autor.

Figura 35 – Modelo dos módulos no relacionados com o efetor.

Fonte: Elaborada pelo autor.

A [Figura 35](#) apresenta um diagrama com o mapeamento do modelo de requisitos MEDC. A política de *Capacidade* foi definida como a modulação do número de máquinas virtuais. A interação do módulo *Efetor* com o hipervisor é realizada através do *Atuador*, este elemento permite em última instância aplicar as mudanças solicitadas pelo **Efetor** se valendo da API disponibilizada pelo hipervisor. O *Monitor* coleta informações do hipervisor através dos sensores, os dados coletados estão relacionados com a utilização da CPU pelas máquinas virtuais. Os processos são iterativos e em tempo real.

No contexto do gerenciador de recursos, a modulação da capacidade pode ser o resultado de uma alteração externa ou interna. A **alteração externa** aparece quando a política de *capacidade* encontra-se implementada fora do ambiente utilizado pelo *Efetor*, por exemplo, existe a possibilidade da política estar implementada em outra camada de *software* (*Node balancer*, gerador de carga), em outra máquina pertencente à mesma rede ou inclusive em outro *data*

center(GROZEV; BUYYA, 2015) para federações de máquinas. As duas últimas caracterizam-se por integrar o *Efetor* local com um conjunto de controladores externos (WANG; KUSIC; KANDASAMY, 2010). Esse tipo de implementação colaborativa enxerga os recursos do controlador local como parte de um todo, e não tem conhecimento pleno do estado interno do gerenciador de recursos (**Efetor**) —Quando a política de *capacidade* está implementada externamente ao gerenciador de recursos, primeiro, o fluxo de carga de trabalho é suspenso por ação dos controladores externos ao gerenciador de recursos local, seguidamente o *Efetor* realiza a ação localmente.

A **alteração interna** é o resultado de uma análise do gerenciador de capacidade local, levando em conta o sistema de múltiplas camadas, para diminuir os recursos primeiro é necessário o cancelamento ou suspensão do fluxo de carga de trabalho das máquinas selecionadas para serem desativadas, o que envolve uma notificação ao *Node balancer* com os IDs das máquinas virtuais selecionadas. Os próximos passos são os mesmos que uma alteração externa.

4.5.1.1 Desligar máquinas

Com as informações coletadas pelo *Monitor* o *Efetor* tem conhecimento do estado das máquinas virtuais que permite-lhe modular a quantidade de recursos de maneira coordenada. Para uma ordem de modulação externa, assume-se que o fluxo de carga de trabalho foi suspenso e a máquina possivelmente encontra-se processando as requisições remanescentes. A seguir são descritos alguns dos passos para o desligamento de máquinas virtuais:

1. É Recebida a ordem com um número de máquinas para serem desligadas;
2. São selecionadas as máquinas com estados: *Iniciando* e *Executando* na ordem respectiva;
3. É executada a ordem de desligar as máquinas segundo o número definido no passo 1, e a ordem definida no passo 2;
4. São atualizadas as informações relacionadas com a tabela de estados do **Efetor**.

As máquinas que estavam no estado *Iniciando* são desligadas imediatamente. Por outro lado, as máquinas com estado *Executando*, são passadas para o estado intermediário *Parando* com o objetivo de terminar de processar as requisições remanescentes; finalmente são desligadas e terminam no estado *Desligado*.

4.5.1.2 Ligando máquinas

Analogamente ao processo de desligar máquinas, o processo de ativar ou ligar novas máquinas virtuais considera os estados atuais das máquinas junto com uma sequencia de prioridades para disponibilizar os novos recursos solicitados. Para essa tarefa são utilizadas as informações fornecidas pelo *Monitor*. A seguir é apresentada uma sequencia de passos que o gerenciador de recursos executa:

1. É recebida a ordem com um número de máquinas a ser disponibilizados;
2. São selecionadas as máquinas segundo a ordem de prioridades: *Parando*, *Pausado*, e finalmente *Desligado* respectivamente;
3. É executada a ordem para ativar ou ligar certo número de máquinas definido no passo 1 e segundo a ordem definida pela prioridade de estados do passo 2;
4. São atualizados os estados das máquinas.

As máquinas que estiverem no estado *Pausado* imediatamente são ativadas e passam para *Executando*, da mesma maneira a mudança de estado acontece com as máquinas no estado *Parando* para *Executando*. Após as mudanças dos estados, o *Node balancer* deve ser notificado com o objetivo de ativar o fluxo de requisições para as máquinas. Dois casos são tratados no processo de notificação: Quando a máquina encontra-se no estado inicial *Desligado*, após a inicialização do sistema operacional e dos serviços, a própria máquina virtual notifica ao *Node balancer* a disponibilidade para receber requisições, isto é necessário, porque é um requisito que todas as aplicações que processam as requisições devam estar funcionando com antecipação. Quando as máquinas encontram-se nos estados iniciais *Pausado* e *Parando*, a notificação por parte dos serviços não é necessária, porque as máquinas já estão disponíveis para receber requisições.

4.5.2 Modelo de demanda

O modelo de demanda visa a avaliação de desempenho de sistemas computacionais considerando mudanças na carga de trabalho que chega ao sistema. No contexto de sistemas *e-commerce* com várias camadas, a carga de trabalho está representada por requisições Web que são submetidas à camada de aplicação. A geração de carga de trabalho é o processo pelo qual vários clientes realizam solicitações com diferentes taxas e tempos de processamento para o SUT (*Software Under Test*).

Para efeitos da implementação em sistemas de múltiplas camadas e levando em consideração a [seção 4.5](#), o *Node balancer* foi identificado como o ponto que permite o acesso à taxa de requisições ativas no sistema, dentro do conjunto de camadas, porque é o lugar onde é registrado em tempo real a carga de trabalho que chega ao sistema.

A [Figura 36](#) apresenta a interação entre os módulos **Efeto** e **Monitor** com o *Node balancer* para a modulação da *Demanda*. Da perspectiva do *Efeto*, as modulações são realizadas utilizando atuadores que enviam mensagem para os Agentes geradores de carga do Bench4Q para aumentar ou diminuir o nível de carga que chega à camada de aplicação.

Para a geração de carga propriamente dita, os Agentes utilizam pequenos módulos chamados EBs. Esses módulos geram solicitações HTTP diretamente ao Servidor onde está

Figura 36 – Esquema geral do modelo de demanda implementado no *Node balancer*.

Fonte: Elaborada pelo autor.

instalado o SUT que fornece um conjunto de 14 páginas com diferentes funções. Esses EBs agem como *browsers* enviando e recebendo solicitações usando o protocolo HTML. Cada EB envia uma série de pedidos dentro de uma sessão do usuário. O padrão de navegação dentro de uma sessão é governado por um grafo dirigido chamado CBMG (MENASCÉ *et al.*, 1999)

Para realizar a análise de desempenho não-estacionária, é preciso utilizar cargas de trabalho que possam excitar o sistema até evidenciar o transiente na métrica que está sendo observada, para que as características dinâmicas do sistema sejam claramente observadas. Em (HELLERSTEIN *et al.*, 2004b), são propostas algumas funções, ou sinais, de perturbação, por exemplo: impulso, degrau, rampa, seno, exponencial, seno modulada, etc.

4.5.2.1 Gerando a perturbação

A definição de parâmetros é um passo prévio do *Efetor* implementado no *Node balancer*. Com este passo, o sistema encarregado de gerar a carga de trabalho é programado inicialmente para a execução do experimento, mas posteriormente pode ser alterado em tempo de execução. Os parâmetros necessários da metodologia proposta para a execução de experimentos que permitam mostrar no regime transiente as características dinâmicas do sistema avaliado são descritos sucintamente a seguir:

- **Número de EBs:** É um fator determinante na execução dos experimentos, porque a implementação do degrau está baseado no controle do número de EBs ativos por Agente;
- **Tempo do degrau:** O tempo é definido em segundos e permite determinar a duração da perturbação no experimento;
- **Tempo sem degrau:** O tempo que o sistema após a ativação do degrau espera para o próximo a reativação do degrau;

- **Tempo de experimento:** Representa o tempo total do experimentos e é dado em segundos. Finalizado esse tempo, todos os EBs são desativados.

A Figura 37 apresenta uma representação básica da estratégia utilizada para a geração do degrau. No processo são utilizados os parâmetros definidos no *Node balancer*. O objetivo da abordagem proposta não é reproduzir fielmente uma carga de trabalho realística. Segundo Mamani *et al.* (2015), uma carga suficiente e necessária para extrair as características dinâmicas do sistema pode ser um degrau ou uma sequencia de perturbações.

Figura 37 – Modelos de estados das máquinas virtuais definidos pelo Efetor.

Fonte: Elaborada pelo autor.

O *Node balancer* define os parâmetros de configuração e, quando a taxa de requisições ativas alcança um nível estacionário, é enviada a ordem para iniciar o degrau e controlar o processo de avaliação até finalizar o tempo de experimento.

4.6 Protótipo do gerenciador da demanda e capacidade

Alterações na submissão de requisições para o sistema em tempo real durante o experimento são uma limitante de vários tipos de *benchmarks* estacionários. Essa limitação implica que toda a carga de trabalho que vai ser utilizada não possa ser alterada, o que impossibilita a execução de experimentos que possam expor as características dinâmicas dos sistemas, como é o foco deste trabalho.

O Bench4Q é um *benchmark* robusto que oferece varias ferramentas e métricas novas e herdadas do TPC-W, mas por estar projetado para avaliação de desempenho de sistemas computacionais no regime estacionário, não oferece o ferramental mínimo necessário para a execução de experimentos. Não suporta nativamente a alocação e a deslocação dos recursos e também não faz o controle da taxa de requisições que chegam ao sistema: a carga de trabalho é definida de maneira indireta com a quantidade de Agentes instanciados, EBs por agente, tamanho do *think times*, entre outros.

Para a geração da carga de trabalho algumas abordagens foram implementadas para mudar o comportamento padrão de *benchmarks* similares ao TPC-W (MI *et al.*, 2009; MI *et al.*, 2010). O trabalho realizado por (SOUZA, 2016) segue uma orientação similar no Bench4Q com a extensão de algumas classes da versão original. Essa versão modificada é utilizada no presente trabalho para a geração da carga de trabalho. Utilizando o modelo de referencia MEDC foram implementados mais dois módulos. O primeiro módulo chamado **Load balancer**, que permitir o gerenciamento de máquinas virtuais e carga de trabalho, e da mesma maneira foi projetado o módulo chamado **VM Manager**, que gerencia a capacidade do sistema através do número de máquinas virtuais em tempo real.

4.6.1 Node balancer

A Figura 38 mostra um diagrama de classes simplificado do protótipo apresentando algumas das classes mais representativas. A classe **AbstractManagerServer** é uma classe abstrata que fornece a base necessária para a definição de objetos que permitam a comunicação com controladores remotos e com o módulo *VM Manager*, a classe **HaProxyManager** estende as funções básicas, adicionalmente tem como atributos o objetos relacionados com o modelo de requisitos MEDC, realiza a interligação de dependências entre o *Efector*, *Capacidade*, *Demanda* e o *Monitor*.

A classe **Effector** está encarregada de gerar uma camada de abstração para as máquinas virtuais e o nível de carga de trabalho, abstrai as máquinas virtuais através da classe **VMachine** e gerencia o conjunto levando em consideração os estados definidos na subseção 4.3.1, a classe **HaProxyActuator** permite abstrair a camada de comunicação pela rede para o envio de comandos tanto, para os Agentes do Bench4Q como para o *VM Manager*. Adicionalmente, aplica as configurações aos fluxos de carga de trabalho associados com os servidores, desabilitando ou habilitando o fluxo.

A classe **Monitor** coleta e processa informações do próprio *HaProxy* utilizando a classe **HaProxyLogParser** que filtra a *taxa de requisições* de informações tais como: estado dos servidores, número de conexões atuais com erro e o máximo alcançado para cada servidor, taxa de requisições atuais por cada máquina virtual e o total que chega ao sistema, entre outras informações.

A interface **Policy** obriga a implementação do método *update()* requisito da MEDC, dessa maneira as classes relacionadas com a *Demanda* ou *Capacidade* são definidas. Como parte da implementação a classe **StepDemand** define a intensidade e o tempo de duração de um degrau na carga de trabalho repassando para a classe *Effector* a responsabilidade da aplicação de tal mudança. A classe **ContractModulation** permite enviar ordens para o módulo *MV Manager* indicando o número de máquinas virtuais para serem desligadas ou ligadas.

Figura 38 – Diagrama de classes do *Node balancer*.

Fonte: Elaborada pelo autor.

4.6.2 VM Manager

O projeto do módulo gerenciador de recursos segue o mesmo padrão que a MEDC estabelece. A classe **HypervisorManager** concentra a funcionalidade e interliga as diferentes dependências que a MEDC estabelece. Entre outras tarefas, esta classe é a encarregada de coletar as informações iniciais do sistema através do método *scanHypervisor()* como o intuito de preencher com informações a camada de abstração que o protótipo define. Essas informações são repassada para as classes **Monitor** e *Effector* por única vez, posteriormente a classe *Monitor* sincroniza as informações com o hipervisor.

A classe abstrata **Policy** define o método *update()* que é implementada pela classe **ControllerPI** que aplica modulações no sistema segundo o critério adaptativos do controle clássico. A classe abstrata *Policy* fornece a base para outro tipos de moduladores, por exemplo, lógica fuzzy, poderiam ser implementados e adicionados na arquitetura.

A classe **Monitor** coleta informações relacionadas com a utilização da CPU de cada máquina virtuais que se encontra nos estados *Executando* ou *Parando* da [subseção 4.3.1](#). Essas informações são concentradas para ser gerada uma media de utilização que pode ser utilizada tanto pelo *ControllerPI* como o *Effector*. Vale salientar que a interface **ISensor** habilita a monitoração de outro tipo processos, tais como: I/O de rede ou disco, memória, etc.

Figura 39 – Diagrama de classes do *middleware* gerenciador do Hypervisor.

Fonte: Elaborada pelo autor.

A classe **Effector** abstrai a implementação das máquinas virtuais utilizando a classe **VMachine** onde os estados são armazenados junto com informações relacionadas às métricas utilizadas na avaliação de desempenho. A classe *Effector* está encarregada do gerenciamento das máquinas virtuais adicionando um comportamento temporal (*delay*) nas ordens para ligar e desligar máquinas assim como definir qual é o próximo estado da máquina virtual.

RESULTADOS E DISCUSSÕES

Com o objetivo de demonstrar a utilidade da metodologia para a construção de *benchmarks* proposta foram planejados e executados uma série de experimentos. São apresentados alguns resultados relacionados com o atraso na disponibilização de recursos (máquinas virtuais) pelo gerenciador de recursos (hipervisor) existente na computação em nuvem. O *benchmark* é utilizado para avaliar o comportamento dinâmico de dos gestores de banco de dados conhecidos na literatura. É também replicada em uma escala menor, o experimento realizado no projeto de doutorado de Nobile (2013) utilizando o *benchmark*. O capítulo também e descreve as configurações do ambiente como: características das máquinas utilizadas e dos *softwares* utilizados nos experimentos.

5.1 *Hardware e software*

Para realizar a avaliação de desempenho de sistemas computacionais no regime não-estacionário é imprescindível detalhar todos os elementos de *hardware* e *software* utilizados e verificar como tais elementos podem influenciar de forma positiva ou negativa no desempenho do *benchmark*. A Tabela 1 exibe alguns detalhes das máquinas utilizadas nos experimentos.

O ambiente de testes foi implementado utilizando máquinas com dispositivos de rede Gigabit. A seguir uma breve descrição dos elementos utilizados nos experimentos:

Load balancer Uma máquina com duas interfaces de rede que permite a comunicação entre a rede dos clientes geradores de carga de trabalho e as máquinas virtuais. O *software* utilizado para realizar o balanceamento de carga é o HaProxy¹ da versão 1.5. Para o controle do HaProxy foi projetado um módulo do tipo *wrapper* na linguagem Java que foi chamado no trabalho de *Load balancer*. Ele implementa de maneira automática a configuração e o gerenciamento da carga de trabalho através do HaProxy.

¹ <<http://www.haproxy.org/>>

Tabela 1 – Características de *software* e *hardware* do *Node balancer*.

Nome	Características				#
	CPU (GHz)	Mem (GB)	HD (MB/sec)	OS	
Load balancer	i5-3330-(3.0) x 4	7,8	87,74	ClearOS 6.6 - 64 bits	1
DB Server (pool)	Q6600-(2.4) x 4	7,8	71,21	Ubuntu 14.04.2 - 64 bits	8
DB Server	FX-8320-(3.5) x 8	23	285,58	Ubuntu 14.04.3 - 64 bits	1
Clientes	Q6600-(2.4) x 4	7,8	76,96	Ubuntu 14.04.2 - 64 bits	9
Hipervisor	X5660-(2.8) x 12	11	251,98	Ubuntu 15.04 - 64 bits	1
Máquina virtual	QEMU Virtual x 1	1	178,60	Ubuntu 12.04.5 - 32 bits	-

Fonte: Dados da pesquisa.

DB Server (pool) São 8 máquinas utilizadas como servidores de banco de dados. As máquinas são utilizadas para alguns experimentos com o objetivo evitar o gargalo produzido pela grande quantidade de requisições que chegam à camada de aplicação e são repassadas para a camada de dados. O *software* utilizado como banco de dados para todas as máquinas é o PostgreSQL² da versão 9.3.

DB Server A máquina foi utilizada para a realização do *benchmarking* de dois reconhecidos gestores de banco de dados: o PostgreSQL da versão 9.3 e o DB2 da versão 10.5. Nos testes evidenciaram-se problemas relacionados com gargalos na quantidade de requisições por limitações nos recursos disponíveis.

Clientes Os clientes são máquinas utilizadas pelos geradores de carga de trabalho (Agentes). As máquinas encontram-se em uma rede isolada e o único ponto de comunicação com a camada de aplicação é através do *Load Balancer*. O console é uma das máquinas, mas ele não gera requisições.

Hipervisor A máquina possui duas interfaces de rede que foram compartilhadas com as máquinas virtuais para receber requisições da rede dos clientes e encaminhá-las para a camada de dados. O *software* utilizado para realizar a virtualização de máquinas foi QEMU (KVM) 2.2.0. Na máquina foi instalado o módulo *software* chamado *VM Manager* do tipo *wrapper*

² <<http://www.postgresql.org/>>

para gerenciar os recursos do hipervisor (máquinas virtuais), a linguagem utilizada para o desenvolvimento foi o Python com o intérprete na versão 2.7.9.

Máquina virtual Possui duas interfaces de rede virtuais conectadas a duas interfaces físicas fornecidas pelo *Hipervisor*. A máquina virtual tem instalado o servidor de aplicações WebSphere (WAS CE) ³ da versão 3.0 utilizando a máquina virtual OpenJDK 1.6.

O banco de dados é um fator de grande influencia nos experimentos e características como o tamanho são temas de preocupação. A [Tabela 2](#) apresenta informações relacionadas ao número de elementos inseridos nas diferentes tabelas definidas pelo Bench4Q para o DB2 e o PostgreSQL:

Tabela 2 – Número de elementos no banco de dados por tabela.

Tabela	Número de elementos
CUSTOMER	1440000
ORDERS	1296000
ITEM	1000
AUTHOR	250
ADDRESS	2880000

Fonte: Dados da pesquisa.

Para a conexão das máquinas virtuais com o banco de dados foi utilizado o *driver* JDBC com os seguintes parâmetros: *Pool Max Size* = 100, *Blocking Timeout* = 5000 milissegundos, *Idle Timeout* = 15 minutos.

5.2 Tempo de inicialização de máquinas virtuais

Sistemas residentes na nuvem são afetados por fatores antes não conhecidos. Um deles é o atraso na disponibilidade de recursos, os atrasos estão influenciados pelo nível de compartilhamento dos recursos físicos, por exemplo, um hipervisor de máquinas virtuais poderia estar servindo a vários clientes de maneira simultânea.

O tempo que demora uma máquina virtual num ambiente em nuvem onde *hardware* e *software* diversos interagem escapa do controle do usuário que contrata o serviço. Por diferentes motivos esse tempo pode mudar e o impacto gerado em algoritmos o sistemas de gerenciamento de recursos do lado do usuário pode ser determinante no sucesso do sistema em manter os níveis de serviço contratados (HUANG; HE; MIAO, 2014).

Dado um cenário de violação do SLA, o sistema gerenciador de desempenho providenciará mais recursos para diminuir a taxa de utilização. De maneira simplificada, o processo

³ http://www.ibm.com/developerworks/br/local/websphere/was_ce/

compreende: a ordem do gerenciador de desempenho procurando mais recursos a serem disponibilizados para o hipervisor e os processos internos do hipervisor para satisfazer o pedido.

A seguir são apresentados alguns resultados são produto de uma pesquisa realizada em conjunto com Nobile (2013) para quantificar o impacto da inicialização de máquinas virtuais com diferentes níveis de sobrecarga inicial no servidor físico. Este cenário é característico em ambientes de computação em nuvem e são uma pequena amostra da crescente complexidade de *software* e *hardware* existentes.

Para o hipervisor fornecer os recursos solicitados, tem de lidar com diferentes condições de carga no servidor físico. Num sistema ideal, as máquinas virtuais estariam disponíveis imediatamente após o pedido, mas não é o caso observado em data centers reais. Uma solução utilizada é manter um conjunto de máquinas de reserva em estado latente, essa solução apresenta custos adicionais, porque as instâncias são cobradas pelo provedor por terem recursos físicos alocados, por exemplo, a memória usada pela máquina virtual.

Para entender como os tempos de inicialização de máquinas virtuais são afetados pelas condições anteriores do servidor físico ao pedido de novos recursos, foram realizados experimentos levando em conta dois fatores: a carga do servidor e a quantidade de máquinas virtuais que são requisitadas.

Como servidor de máquinas virtuais (hipervisor) foi utilizado um Dell PowerEdge T4103, com processador Intel Xeon X56604, dotado de 6 núcleos físicos de 2.8 GHz e 12 MB de cache por núcleo. Além disso, com memória principal de 12 GB e 1.5 TB de memória secundária. O sistema operacional foi utilizado o um GNU/Linux, com kernel 3.2.0-31 e sistema de arquivos ext4. Foi utilizado um hipervisor do tipo *bare-metal* descrito na subseção 2.5.1.

Foram definidas duas possíveis situações de necessidade de recursos, a primeira com 6 máquinas virtuais e a segunda com 9 máquinas virtuais. Essas máquinas virtuais são ligadas todas ao mesmo tempo, o intervalo de tempo entre a ordem de inicialização até que a máquina virtual estivesse pronta e disponível foi registrado pelo gerenciador de desempenho. A sobrecarga do servidor antes da solicitude de recursos foi dividida em 8 níveis que representam máquinas virtuais preexistentes consumindo recursos. Esses níveis são: 2, 4, 6, 8, 10, 12, 14 e 16 instâncias previamente ligadas e executando processos. Os experimentos foram executados segundo um arranjo fatorial completo e cada experimento foi repetido 6 vezes.

Para simular uma máquina virtual executado tarefas, foi utilizado um *benchmark* do tipo *CPU-bound* chamado Smallpt⁴, de modo que a inicialização de novas máquinas virtuais não acontecesse em condições ociosas. A Figura 40 apresenta o tempo médio de inicialização para 6 e 9 máquinas virtuais sob diferentes condições de carga inicial no servidor.

A influencia da inicialização de 6 máquinas se manteve com uma média praticamente constante independente da quantidade de máquinas virtuais que já eram executadas pelo hipervi-

⁴ <<http://openbenchmarking.org/test/pts/smallpt>>

Figura 40 – Tempo de inicialização das máquinas virtuais em condições diferentes de carga no servidor.

Fonte: Nobile (2013).

sor. Por sua vez, para a quando iniciadas 9 máquinas houve um aumento no tempo médio de inicialização.

Os resultados dos experimentos realizados demonstram que existem intervalos de tempo variáveis que são parte de um comportamento dinâmico inerente dos processos que a computação em nuvem lida. Os SLAs estabelecem níveis de desempenho prestado pelos provedores aos seus consumidores, que não contemplam situações de sobrecarga no regime transitório, mas que podem afetar o funcionamento de gerenciadores adaptativos em sistemas medianos e de grande porte, sendo este um fator que comumente não é contemplado, mas que esconde uma potencial fonte de falhas nos níveis de serviço contratados.

5.3 Comparação de PostgreSQL e DB2

O experimento foi projetado para identificar comportamento transiente dentro de alguns *software* que compõem as camadas dos sistemas *e-commerce* tradicionais. As configurações iniciais do sistema é:

- Quatro clientes (agentes geradores de carga);
 - Cada agente gerencia 20 EBs iniciais passando para 40 EBs finais;
 - As requisições utilizadas pelos EBs são do tipo *browsing* definido no Bench4Q, esse modo caracteriza-se por gerar consultas de só leitura (*read only*) nos banco de dados.
- Quatro máquinas virtuais foram utilizadas na camada de aplicação;

- O SUT fornecido pelo *benchmark* foi projetado para suportar o DB2, mas foram realizadas as alterações pertinentes nas consultas SQL para também suportar o PostgreSQL.
- A métrica utilizada na avaliação é o tempo de resposta em milissegundos medidos nos clientes;
- Os experimentos foram executados utilizando a máquina **DB Server** descrito na [Tabela 1](#) para hospedar os bancos de dados;
- PostgreSQL e DB2 são os gerenciadores de banco de dados utilizados;
- Foram realizadas 4 réplicas para cada caso.

5.3.1 PostgreSQL

PostgreSQL é um conhecido banco de dados de código aberto frequentemente utilizado na computação em nuvem, e para fins de alcançar o melhor desempenho possível do banco de dados, foi utilizada a ferramenta *pgtune*⁵ que permite redefinir os valores dos parâmetros definidos inicialmente para um contexto de alto desempenho. A [Figura 41](#) apresenta o tempo de resposta de 500 segundos de experimento. A estratégia foi executar o *benchmark* inicialmente com a metade da quantidade de EBs disponíveis, após o segundo 100, o módulo *Demanda* do *Load balancer* indica ao *Efetor* duplicar a carga de trabalho. O processo consiste em ativar o restante dos EBs que se encontravam em estado latente, o resultado da mudança é uma perturbação que leva o sistema de 150 milissegundos para um valor estacionário aproximado de 286 milissegundos.

Figura 41 – Tempo de resposta das requisições nos clientes utilizando o banco de dados PostgreSQL.

Fonte: Dados da pesquisa.

⁵ <<http://pgfoundry.org/projects/pgtune>>

O tempo de resposta mantém-se durante a execução do degrau estacionando em 286 milissegundos aproximadamente. Com a execução do experimento foi verificado que a versão do PostgreSQL utilizada não apresenta um comportamento adaptativo para mudanças abruptas.

5.3.2 DB2

O DB2 é um *software* de caráter comercial, mas com uma ampla influência na área acadêmica. Vários trabalhos foram desenvolvidos visando um funcionamento adaptativo, principalmente abordagens relacionadas com a gestão automática de memória (HELLERSTEIN *et al.*, 2004a; MATEEN *et al.*, 2009; STORM *et al.*, 2006).

Figura 42 – Tempo de resposta das requisições nos clientes utilizando o banco de dados DB2.

Fonte: Dados da pesquisa.

A Figura 42 mostra o comportamento do tempo de resposta. Aplicando o mesmo protocolo de experimentos com o mesmo nível de carga e durante o mesmo tempo de experimento, os resultados são diferentes. No segundo 100 o módulo *Demanda* do *Load balancer* requisita do sistema gerador de carga aplicar um degrau na carga de trabalho para dobrá-la. Utilizando os termos de avaliação de desempenho não-estacionário, podemos observar um primeiro nível estacionário em torno de 90 milissegundos passando para um novo patamar em torno de 146 milissegundos.

Na perspectiva do monitor de recursos do sistema operacional, a memória virtual ocupada pelo banco de dados aumento de 3GB para 11GB quando acontece a mudança na carga de trabalho. Não foi realizado nenhum *tuning* no banco de dados. A documentação refere que o *software* realiza todo o *tuning* necessário sob demanda e em tempo real.

Este experimento evidencia a utilidade de um *benchmark* no regime não-estacionário, para o que está sendo desenvolvido um módulo adaptativo na camada de aplicação ou no balanceador de carga. Resulta necessária a aplicação de um *benchmark* que possa revelar e

parametrizar os comportamentos adaptativos já existentes em outras camadas de *software*. Este cenário não é particular, mas sim um caso comum na computação em nuvem, onde existem uma ampla variedade de provedores de *software* e *hardware* com diversos comportamentos dinâmicos.

5.3.3 Mudança no banco de dados

Como foi observado existem diferenças no funcionamento dos diferentes bancos de dados. Foi evidenciado um ponto de estrangulamento no funcionamento do sistema *e-commerce*, por tal motivo foi realizada uma mudança na configuração do ambiente utilizado nos experimentos. A Figura 43 apresenta uma arquitetura onde existe um único banco de dados (*DB Server*) que resolve todas as requisições.

Figura 43 – Arquitetura do OnlineBench4Q com um único banco de dados.

Fonte: Elaborada pelo autor.

Esta arquitetura sugerida não permite realizar um *benchmarking* satisfatório focando em outras camadas do sistema *e-commerce* limitando a taxa de requisições e em consequência a utilização da CPU das máquinas virtuais mantém-se em níveis baixos, adicionalmente com cargas de trabalho que lidam o limite do banco de dados, o número de erros no resultado das requisições aumenta consideravelmente.

A Figura 44 apresenta uma alternativa para o problema da distribuição da carga de trabalho entre vários servidores (*DB Server (pool)*) de bancos de dados. Esta mudança permite escalar facilmente em outras camadas da arquitetura.

O banco de dados utilizado para esta nova instalação foi o PostgreSQL com as mesmas características indicadas na Tabela 2. Os próximos experimentos utilizam esta nova arquitetura.

Figura 44 – Arquitetura do OnlineBench4Q com o *pool* de bancos de dados.

Fonte: Elaborada pelo autor.

5.4 Adaptação às mudanças internas

Existem múltiplos cenários onde o *benchmark* para avaliação de desempenho não-estacionária pode ser utilizado. Um contexto de avaliação é o ambiente de um gerenciador de máquinas virtuais sensível a mudanças externas. O experimento vai simular um caso comum de mudança externa que o gerenciador de recursos deve lidar. Um controlador PI (*Proportional Integral*)⁶ foi utilizado para o gerenciamento da quantidade de máquinas virtuais ativas dentro no sistema. Mudanças externas como *backups*, picos na carga de trabalho, compartilhamento de recursos, perda de servidores, etc. são comuns no funcionamento de sistemas de computação em nuvem (HELLERSTEIN *et al.*, 2005). As configurações do ambiente são descritas a seguir:

- A capacidade do sistemas foi definida inicialmente com oito máquinas virtuais;
 - O cenário é definido para com as oito máquinas processando requisições;
 - Vai ser gerada uma perturbação na capacidade do sistema.
- A demanda é representada por uma taxa de requisições fixa;
 - Oito agentes geradores de carga de trabalho;
 - Cada agente foi configurado para 45 EBs que representam uma taxa aproximada de 1450 *req/s*.
- A métrica utilizada na avaliação de desempenho é a porcentagem de utilização da CPU (média da utilização de todas as máquinas virtuais);
- O gerenciador de banco de dados utilizado foi o PostgreSQL no arranjo de máquinas DB Server (pool) descrito na Tabela 1.

⁶ Esta técnica tende a zerar o erro com respeito a um valor referencial.

Em tempo de execução vai ser alterado o contrato SLA (*Service Level Agreement* - Acordo de Nível de Serviço) previamente definido em 45% de utilização da CPU para 58%. O módulo **Capacidade** através do controlador PI terá que gerenciar os recursos internos para cumprir o novo contexto.

Figura 45 – Comportamento do controlador PI ante uma mudança no contrato de 45% de utilização para 58%.

Fonte: Dados da pesquisa.

A [Figura 45](#) apresenta o comportamento do controlador no decorrer da mudança do contrato. A linha na cor azul representa a utilização da CPU de todas as máquinas virtuais, a linha vermelha representa uma media móvel exponencial (comumente usada em implementações para evitar o ruído) e finalmente a linha na cor laranja representa o número de máquinas virtuais no estado *Executando*.

Aproximadamente no segundo 38, o módulo *Capacidade* do *Load balancer* envia uma solicitação para o *VM Manager* indicando o novo valor de referência. O novo valor é adicionado e o módulo de *Capacidade* local aplica as primeiras ordens —desligar máquinas virtuais. Antes do segundo 38 o sistema está em um nível estacionário usando 8 máquinas virtuais para resolver a carga que estava sendo submetida, após a mudança da referência progressivamente são desligadas as máquinas virtuais até chegar ao número de 5. Até esse ponto a utilização da CPU de todas as máquinas aumenta lentamente até chegar ao novo patamar. Quando o novo patamar é alcançado acontecem alguns problemas de instabilidade, porque os valores das constantes $k_i = -0.01$ e $k_p = 0$ foram definidos de maneira empírica, somente com o intuito de testar o *benchmark* ante uma aplicação realística.

5.5 Análise do degrau

O experimento aqui descrito reproduz um ambiente típico de implantação de um ambiente de computação em nuvem atendendo a um *e-commerce* SaaS, o que mostrou como é possível medir o desempenho que varia ao longo do tempo, utilizando a metodologia de *benchmarking* de desempenho não-estacionário proposto. As configurações do ambiente de experimento são descritas a seguir:

- A capacidade do sistema foi definida de maneira fixa por meio de oito máquinas virtuais;
- A demanda é representada pela taxa de requisições medida em *req/s*;
 - São utilizados oito agentes geradores de carga;
 - Cada agente foi configurado para executar inicialmente 39 EBs (1375 *req/s* aprox.) para 62 EBs (2153 *req/s* aprox.).
- A métrica utilizada é a utilização da CPU das máquinas virtuais;
- Foram executadas 6 réplicas;
- O gerenciador de banco de dados utilizado foi o PostgreSQL no arranjo de máquinas DB Server (pool) descrito na [Tabela 1](#).

A [Figura 46](#) apresenta as taxas de utilização do sistema real para os experimentos realizados com base nos valores da entrada (carga de trabalho), medida em requisições por segundo, juntamente com a saída simulada do modelo extraído no sistema real. A carga de trabalho utilizada é do tipo degrau, no instante inicial a execução do experimento, a carga está próximo a 1376 requisições por segundo. Já no instante zero, quando inicia o degrau e a carga aumenta de maneira brusca chegando ao patamar de 2153 em aproximadamente 1 segundo, representando um crescimento superior a 50% ao comparado com a carga inicial, se mantém até o termino do experimento. A resposta a essa carga é representado pela taxa de utilização em preto no gráfico da [Figura 46](#), que é possível observar que a taxa de utilização está abaixo de 40% antes do instante zero da execução do experimento. Para alguns instantes depois que a perturbação em degrau foi introduzida no sistema, após o instante zero, a taxa de utilização apresenta um comportamento transiente chegando a ultrapassar os 80% da taxa de utilização da CPU. Na [Figura 46](#), é ilustra a taxa de utilização da simulação oriunda ao modelo extraído do sistema, essa taxa de utilização é apresentada pela cor Azul. As saídas reais e simuladas não estão sobrepostas, mas elas se acompanham o que sinaliza para um bom ajuste do modelo. Em alguns instantes chegam a valores muito próximos como nos segundos 6 e 27 aproximadamente.

Para que uma função de transferência seja estimada, experimentos específicos devem ser executados. A partir dos dados produzidos, deve-se excluir os efeitos transitórios iniciais (*warm-up*) e manter o mínimo de informação sobre o desempenho antes do degrau. O efeito

Figura 46 – Dados experimentais e função de transferência gerada.

transiente observado deve ser mantido e pode-se retirar os dados após o sistema ter atingido o estado estacionário final. Para este estudo de caso, como pode ser observado na [Figura 46](#), os dados foram formatados conforme representação gráfica exposta. Ao utilizar o método dos mínimos quadrados, pode-se estimar os parâmetros a e b para um modelo auto-regressivo com entradas exógenas (*auto-regressive with exogenous input* — ARX) de primeira ordem, o qual pode ser representado por uma Função de Transferência de primeira ordem, desta forma

$$G = \frac{2.075}{z - 0.9019}. \quad (5.1)$$

Vale ressaltar que a carga de trabalho utilizada possuiu um degrau, pois ela é eficaz para expor a dinâmica do sistema. No instante inicial a execução do experimento, era imposto ao sistema uma taxa média de 1376 requisições por segundo. Já no instante denominado zero, quando o degrau ocorre, a carga aumenta de maneira brusca para o patamar de 2153, representando um crescimento superior a 50% quando comparado à carga inicial. A resposta a essa carga é representada pela taxa de utilização em preto no gráfico da [Figura 46](#), em que é possível observar uma taxa de utilização em torno de 40%. Após a perturbação, a taxa de utilização apresentou um comportamento transiente, chegando a aproximadamente 60% da taxa de utilização no final do experimento. No mesmo gráfico ([Figura 46](#)), dados de uma simulação feita a partir do modelo extraído ([Equação 5.1](#)) são apresentados. As saídas real e simulada mostraram-se bem aproximadas e representam bem o desempenho do sistema.

O exemplo ilustra a realização de um *benchmarking* não estacionário e seu uso na modelagem das características dinâmicas de desempenho de um sistema computacional em nuvem.

5.6 Dificuldades e Desafios

A implementação de um protótipo traz desafios comuns para os desenvolvedores, mais especificamente na criação do OnlineBench4Q, além dos problemas típicos relacionados ao desenvolvimento de sistemas de software, apresentaram-se alguns particulares nas etapas de definição dos requisitos passando pelo desenvolvimento, configuração do ambiente e execução de experimentos apresentou desafios. Também, a análise de sistemas computacionais no regime transitório apresenta desafios relacionados

5.6.1 Definição da taxa de requisições

Para definir uma carga representativa que satisfaça os requisitos definidos para o *benchmark* de tal maneira que possam aparecer os comportamentos dinâmicos no regime transiente foram definidos vários níveis de carga na tentativa de alcançar valores extremos produto de falhas no funcionamento do *software* ou limites de capacidade no *hardware*. A Figura 47 mostra um com um número fixo de 4 máquinas virtuais recebendo requisições em diferentes taxas W1, W2, W3. Para os valores de W2 e W3 apresentou-se limitações.

Entre as possíveis causas foram listadas o balanceador de carga (HaProxy), o banco de dados (PostgreSQL). Foram aplicados vários outros experimentos e foram descartadas as duas alternativas anteriores, a verdadeira origem era a limitação de entrada e saída presente no ambiente de virtualização, que para o cenário foi definido em 7000 req/s.

Figura 47 – Limites na aplicação de carga da trabalho.

Fonte: Elaborada pelo autor.

5.6.2 Configuração das camada de aplicação e banco de dados

Uma vez definido os valores de taxa de requisições, outro problema foi apreciado. Dado um tempo de experimento maior a 100 segundos os servidores de aplicação apresentavam comportamentos com utilização de CPU altos (100%). A origem de tal comportamento foi as configurações padrões dos servidores, as configurações foram mudadas para padrões de alto desempenho, uma explicação mais detalhada relacionada com essa mudança encontra-se no [Apêndice A](#).

As limitações relacionadas com a camada de banco de dados são explicadas de maneira detalhada na [subseção 5.3.3](#).

CONCLUSÕES

Este capítulo apresenta as conclusões obtidas sobre os resultados e as contribuições geradas com o desenvolvimento deste projeto de pesquisa. Algumas conclusões são facilmente verificáveis por meio da análise dos gráficos, enquanto outras envolveram desprende-se da experiência obtida ao longo da pesquisa tanto em *frameworks* de simulação como na implementação dos requisitos acima do *benchmark* escolhido.

O capítulo também mostra um panorama sobre os rumos futuros deste projeto, uma vez que, mesmo com a obtenção de resultados satisfatórios, o tema ainda permite que muito seja explorado e desenvolvido. Finalmente são apresentadas as publicações geradas.

6.1 Conclusões

Embora a análise transiente de sistemas de eventos discretos sejam ainda pouco explorada, comparativamente com abordagens de avaliação de desempenho tradicional, trabalhos de pesquisa abrangentes lançam luz sobre o tema, ressaltando a importância para sistemas de engenharia de sistemas em diversos campos (PAPADOPOULOS *et al.*, 2015; HUANG *et al.*, 2013; YANG; LIU, 2010; KACZYNSKI; LEEMIS; DREW, 2012; Minqi Li; YANG; XU, 2014; COEVERING, 1995). Este trabalho atende a essa perspectiva com foco na avaliação de desempenho de sistemas computacionais baseado em *benchmarking*. Não obstante a avaliação de desempenho não-estacionária seja relevante e venha atraindo interesse da pesquisa, há um hiato metodológico permanece aberto na área de sistemas distribuídos de grande escala e aplicações complexas oriundas de arquiteturas multicamada.

O trabalho de pesquisa discute questões relacionadas à extensão de *frameworks* projetados para avaliação de desempenho no regime estacionário. Especialmente aborda o uso de análise transiente para derivar modelos empíricos que possam capturar as características dinâmicas de desempenho, bem como as consequências de se mudar de um paradigma baseado em lotes

para um paradigma *online*. Além disso, o presente trabalho sistematiza os principais requisitos para realizar avaliação de desempenho não-estacionária em um conjunto de características: a) modulação da *Demanda*, ou a habilidade de controlar o fluxo da carga de trabalho que chega ao sistema; b) modulação da *Capacidade*, habilidade de controlar os recursos de processamento; c) monitoração, habilidade de registra dados e transformá-los em informações em determinados intervalos de tempo; d) Efetor, ou a habilidade para modelar os diferentes estados dinâmicos dos sistemas no tempo. Esses requisitos, sugeridos pelo modelos MEDC podem ser formulados como um modelo conceitual de referência para construir *benchmarks* focados na análise não-estacionária. O presente trabalho ilustra a utilização do modelo conceitual de referência para estender *frameworks* usados para realizar *benchmarking* na computação em nuvem. A extensão começa usando um conhecido *benchmark* de sistemas *e-commerce* focado na avaliação de desempenho estacionária, traduzindo-o para uma avaliação não-estacionária.

A aplicação prática da extensão do *benchmark* é exemplificada através de experimentos de análise empírica de ambientes de computação em nuvem.

O modelo de referencia MEDC foi originalmente desenvolvido para simulação de eventos discretos. Esta pesquisa investiga sua aplicabilidade para avaliação de desempenho por meio de aferição. A transposição da arquitetura de aspectos do modelo MEDC para *benchmark* não-estacionário é a principal contribuição do trabalho.

A discussão é desenvolvida sobre um estudo de caso em computação em nuvem, uma aplicação relevante para o problema abordado (dinâmica em sistemas computacionais de grande escala). O resultado deste trabalho é a extensão do *framework* Bench4Q, que constitui também uma contribuição. Todos os artefatos produzidos são distribuídos publicamente sob licença *open source*. A principal conclusão da pesquisa é a demonstração da utilidade do modelo MEDC para *benchmark* não-estacionário em avaliação de desempenho de sistemas computacionais. Os exemplos de uso corroboram a conclusão. Como trabalhos futuros destaca-se a modelagem da carga de trabalho em função de métricas de desempenho, o que permitiria o desenvolvimento de um controle de admissão que protege o sistema contra sobrecargas transientes.

6.2 Publicações

O desenvolvimento desta tese de doutorado gerou as seguintes publicações:

- MAMANI, E. L. C.; PEREIRA, L. A.; SANTANA, M. J.; et al. Transient performance evaluation of cloud computing applications and dynamic resource control in large-scale distributed systems. **2015 International Conference on High Performance Computing & Simulation (HPCS)**. Anais... . p.246–253, 2015. Amsterdam, The Netherlands: IEEE.

– Disponível em: <http://dx.doi.org/10.1109/HPCSim.2015.7237046>

- JÚNIOR, L. A. P.; MAMANI, E. L. C.; SANTANA, M. J.; et al. Extending discrete-event simulation frameworks for non-stationary performance evaluation: requirements and case study. **2015 Winter Simulation Conference**. Anais... , WSC '15. p.3150–3151, 2015. Huntington Beach, California: IEEE Press.
 - Disponível em: <http://dl.acm.org/citation.cfm?id=2889013>
- PEREIRA, L. A.; MAMANI, E. L. C.; SANTANA, M. J.; et al. Non-stationary Simulation of Computer Systems and Dynamic Performance Evaluation: A Concern-Based Approach and Case Study on Cloud Computing. **2015 27th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)**. Anais... . p.130–137, 2015. Florianópolis, Brasil: IEEE.
 - Disponível em: <http://dx.doi.org/10.1109/SBAC-PAD.2015.9>

REFERÊNCIAS

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, dec 2010. ISSN 01635980. Disponível em: <http://dl.acm.org/citation.cfm?id=1899928.1899930>. Citado 3 vezes nas páginas 53, 55 e 58.

ALTINKEMER, K.; CHATURVEDI, A.; GULATI, R. Information systems outsourcing: Issues and evidence. **International Journal of Information Management**, v. 14, n. 4, p. 252–268, aug 1994. ISSN 02684012. Disponível em: <http://linkinghub.elsevier.com/retrieve/pii/0268401294900035>. Citado na página 44.

AN, B.; LESSER, V.; IRWIN, D.; ZINK, M. Automated negotiation with decommitment for dynamic resource allocation in cloud computing. In: **Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1 - Volume 1**. Toronto, Canada: International Foundation for Autonomous Agents and Multiagent Systems, 2010. p. 981–988. ISBN 978-0-9826571-1-9. Disponível em: <http://dl.acm.org/citation.cfm?id=1838338>. Citado na página 79.

ARMBRUST, M.; FOX, A.; GRIFFITH, R.; JOSEPH, A. D.; KATZ, R. H.; KONWINSKI, A.; LEE, G.; PATTERSON, D. A.; RABKIN, A.; STOICA, I.; ZAHARIA, M. **Above the Clouds: A Berkeley View of Cloud Computing**. Berkeley, 2009. 25 p. Disponível em: <http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-28.pdf>. Citado na página 47.

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, apr 2010. ISSN 00010782. Disponível em: <http://portal.acm.org/citation.cfm?doid=1721654.1721672>. Citado 3 vezes nas páginas 44, 46 e 47.

BADGER, L.; GRANCE, T.; PATT-CORNER, R.; VOAS, J. **DRAFT Cloud Computing Synopsis and Recommendations**. Gaithersburg, 2011. 84 p. Disponível em: <http://csrc.nist.gov/publications/drafts/800-146/Draft-NIST-SP800-146.pdf>. Citado 5 vezes nas páginas 44, 47, 48, 51 e 52.

BARHAM, P.; DRAGOVIC, B.; FRASER, K.; HAND, S.; HARRIS, T.; HO, A.; NEUGEBAUER, R.; PRATT, I.; WARFIELD, A. Xen and the art of virtualization. **ACM SIGOPS Operating Systems Review**, v. 37, n. 5, p. 164, dec 2003. ISSN 01635980. Disponível em: <http://dl.acm.org/citation.cfm?id=1165389.945462>. Citado 2 vezes nas páginas 53 e 54.

BELLARD, F. QEMU, a Fast and Portable Dynamic Translator. In: **Proceedings of the Annual Conference on USENIX Annual Technical Conference**. Berkeley, CA, USA: USENIX Association, 2005. (ATEC '05), p. 41. Disponível em: <http://dl.acm.org/citation.cfm?id=1247360.1247401>. Citado na página 56.

BUX, M.; LESER, U. DynamicCloudSim: simulating heterogeneity in computational clouds. In: **Proceedings of the 2nd ACM SIGMOD Workshop on Scalable Workflow**

Execution Engines and Technologies - SWEET '13. New York, New York, USA: ACM Press, 2013. v. 46, n. 0, p. 1–12. ISBN 9781450323499. ISSN 0167739X. Disponível em: <http://www.scopus.com/inward/record.url?eid=2-s2.0-84897369322&partnerID=tZOtx3y1http://dl.acm.org/citation.cfm?doid=2499896.2499897http://www.sciencedirect.com/science/article/pii/S0167739X14001770>. Citado na página 35.

BUYYA, R.; RANJAN, R.; CALHEIROS, R. N. InterCloud: Utility-Oriented Federation of Cloud Computing Environments for Scaling of Application Services. In: CHING-HSIEN, H.; LAURENCE, Y.; JONG, P.; SANG-SOO, Y. (Ed.). **Algorithms and Architectures for Parallel Processing**. Springer Berlin / Heidelberg, 2010. v. 6081, p. 13–31. ISBN 978-3-642-13118-9. Disponível em: http://dx.doi.org/10.1007/978-3-642-13119-6_2http://link.springer.com/10.1007/978-3-642-13119-6_2. Citado na página 44.

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**, John Wiley & Sons, Ltd., v. 41, n. 1, p. 23–50, jan 2011. ISSN 00380644. Disponível em: <http://doi.wiley.com/10.1002/spe.995>. Citado 5 vezes nas páginas 27, 28, 29, 33 e 34.

CASANOVA, H. Simgrid: a toolkit for the simulation of application scheduling. In: **Proceedings First IEEE/ACM International Symposium on Cluster Computing and the Grid**. IEEE Comput. Soc, 2001. p. 430–437. ISBN 0-7695-1010-8. Disponível em: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=923223>. Citado na página 27.

CECCHET, E. **RUBiS: Rice University Bidding System**. 2009. Disponível em: <http://rubis.ow2.org/index.html>. Citado na página 62.

CHOI, B. K.; KANG, D. How to develop your own simulators for discrete-event systems. In: **Proceedings of the Winter Simulation Conference 2014**. Piscataway, New Jersey: IEEE, 2014. p. 147–161. ISBN 978-1-4799-7486-3. Disponível em: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=7019885>. Citado 2 vezes nas páginas 31 e 33.

COEVERING, M. C. T. van de. Computing transient performance measures for the M/M/1 queue. **Operations-Research-Spektrum**, Springer-Verlag, v. 17, n. 1, p. 19–22, mar 1995. ISSN 0171-6468. Disponível em: <http://link.springer.com/10.1007/BF01719726>. Citado 3 vezes nas páginas 31, 41 e 111.

FOSTER, I.; ZHAO, Y.; RAICU, I.; LU, S. Cloud Computing and Grid Computing 360-Degree Compared. **2008 Grid Computing Environments Workshop**, Ieee, p. 1–10, nov 2008. Disponível em: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4738445>. Citado 2 vezes nas páginas 45 e 47.

GANDHI, A.; ZHU, T.; HARCHOL-BALTER, M.; KOZUCH, M. A. SOFTScale : Stealing Opportunistically for Transient Scaling. In: NARASIMHAN, P.; TRIANTAFILLOU, P. (Ed.). **Middleware 2012**. 13. ed. Springer Berlin Heidelberg, 2012. p. 142–163. ISBN 978-3-642-35169-3. Disponível em: http://link.springer.com/chapter/10.1007/978-3-642-35170-9_8. Citado 2 vezes nas páginas 43 e 87.

GANDHI, N.; TILBURY, D.; PAREKH, S.; HELLERSTEIN, J. Feedback control of a Lotus Notes server: modeling and control design. In: **Proceedings of the 2001 American Control Conference. (Cat. No.01CH37148)**. IEEE, 2001. v. 4, p. 3000–3005. ISBN 0-7803-6495-3. Disponível em: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=946372>. Citado na página 42.

GARCÍA, D. F.; GARCÍA, J. TPC-W e-commerce benchmark evaluation. **Computer**, v. 36, n. 2, p. 42–48, feb 2003. ISSN 0018-9162. Disponível em: <<http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1178045>>. Citado na página 27.

GE, J.-w.; DENG, Y.-l.; FANG, Y.-q. Research on Storage Virtualization Structure in Cloud Storage Environment. In: **2010 International Conference on Multimedia Technology**. IEEE, 2010. p. 1–4. ISBN 978-1-4244-7871-2. Disponível em: <<http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5630956>>. Citado na página 61.

GROZEV, N.; BUYYA, R. Performance Modelling and Simulation of Three-Tier Applications in Cloud and Multi-Cloud Environments. **The Computer Journal**, v. 58, n. 1, p. 1–22, jan 2015. ISSN 0010-4620. Disponível em: <<http://comjnl.oxfordjournals.org/cgi/doi/10.1093/comjnl/bxt107>>. Citado na página 89.

HELLERSTEIN, J.; STORM, A.; SURENDRA, M.; LIGHTSTONE, S.; PAREKH, S.; GARCIA-ARELLANO, C. Incorporating cost of control into the design of a load balancing controller. **Proceedings. RTAS 2004. 10th IEEE Real-Time and Embedded Technology and Applications Symposium, 2004.**, Ieee, p. 376–385, 2004. Disponível em: <<http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1317284>>. Citado na página 103.

HELLERSTEIN, J. L.; DIAO, Y.; PAREKH, S.; TILBURY, D. M. **Feedback Control of Computing Systems**. 1. ed. Hoboken, NJ, USA: John Wiley and Sons, Inc., 2004. 429 p. ISBN 047126637X. Disponível em: <<http://doi.wiley.com/10.1002/047166880X>>. Citado na página 91.

HELLERSTEIN, J. L.; MORRISON, V.; EILEBRECHT, E. Applying Control Theory in the RealWorld: ExperienceWith Building a Controller for the .NET Thread Pool. **ACM SIGMETRICS Performance Evaluation Review**, v. 37, n. 3, p. 38, jan 2010. ISSN 01635999. Disponível em: <<http://portal.acm.org/citation.cfm?doid=1710115.1710123>>. Citado na página 43.

HELLERSTEIN, J. L.; PAREKH, S.; GRIFFITH, R.; KAISER, G. E.; 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, dec 2005. ISSN 0733-8716. Disponível em: <<http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1546093>>. Citado na página 105.

HORN, P. **Autonomic computing: IBM's Perspective on the State of Information Technology**. IBM, 2001. 1–40 p. (Tertiary Autonomic computing: IBM's perspective on the state of information technology, 26). Disponível em: <http://www.research.ibm.com/autonomic/manifesto/autonomic_computing.pdf>. Citado 2 vezes nas páginas 42 e 46.

HUANG, D.; HE, B.; MIAO, C. A Survey of Resource Management in Multi-Tier Web Applications. **IEEE Communications Surveys & Tutorials**, v. 16, n. 3, p. 1–17, 2014. ISSN 1553-877X. Disponível em: <<http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6728783>>. Citado na página 99.

HUANG, Y.-L.; FUJIMOTO, R.; ALEXOPOULOS, C.; HUNTER, M. On the transient response of open queueing networks using ad hoc distributed simulations. In: **2013 Winter Simulations Conference (WSC)**. Piscataway, New Jersey: IEEE, 2013. p. 2947–2958. ISBN 978-1-4799-3950-3. Disponível em: <<http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6721663>>. Citado 3 vezes nas páginas 31, 41 e 111.

JÚNIOR, L. A. P.; MAMANI, E. L. C.; SANTANA, M. J.; SANTANA, R. H. C.; NOBILE, P. N.; MONACO, F. J. Extending discrete-event simulation frameworks for non-stationary performance evaluation: requirements and case study. In: **2015 Winter Simulation Conference**. Huntington Beach, California: IEEE Press, 2015. (WSC '15), p. 3150—3151. ISBN 978-1-4673-9741-4. Disponível em: <<http://dl.acm.org/citation.cfm?id=2888619.2889013>>. Citado 4 vezes nas páginas 27, 32, 70 e 75.

KACZYNSKI, W. H.; LEEMIS, L. M.; DREW, J. H. Transient Queueing Analysis. **INFORMS Journal on Computing**, v. 24, n. 1, p. 10–28, feb 2012. ISSN 1091-9856. Disponível em: <<http://dx.doi.org/10.1287/ijoc.1110.0452><http://pubsonline.informs.org/doi/abs/10.1287/ijoc.1110.0452>>. Citado 3 vezes nas páginas 31, 41 e 111.

KEPHART, J.; CHESS, D. The vision of autonomic computing. **Computer**, v. 36, n. 1, p. 41–50, jan 2003. ISSN 0018-9162. Disponível em: <http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1160055<http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1160055>>. Citado na página 42.

KIVITY, A.; KAMAY, Y.; LAOR, D.; LUBLIN, U.; LIGUORI, A. kvm: the Linux virtual machine monitor. In: **OLS '07: The 2007 Ottawa Linux Symposium**. [s.n.], 2007. p. 225–230. Disponível em: <<http://www.kernel.org/doc/ols/2007/ols2007v1-pages-225-230.pdf>>. Citado na página 56.

LIBVIRT. **libvirt - The virtualization API**. 2014. Disponível em: <<http://libvirt.org/index.html>>. Citado na página 85.

LIM, H. C.; BABU, S.; CHASE, J. S.; PAREKH, S. S. Automated control in cloud computing: Challenges and opportunities. In: **Proceedings of the 1st workshop on Automated control for datacenters and clouds - ACDC '09**. New York, New York, USA: ACM Press, 2009. (ACDC '09), p. 13. ISBN 9781605585857. Disponível em: <<http://doi.acm.org/10.1145/1555271.1555275><http://dl.acm.org/citation.cfm?id=1555275><http://portal.acm.org/citation.cfm?doid=1555271.1555275>>. Citado na página 79.

LUZ, H. J. F. **Um método para previsão de sobrecarga transiente em sistemas computacionais por meio de modelos dinâmicos obtidos empiricamente**. 41 p. Tese (Doutorado) — Universidade de São Paulo, 2014. Disponível em: <<http://www.teses.usp.br/teses/disponiveis/55/55134/tde-18082015-084441>>. Citado na página 28.

MAMANI, E. L. C.; PEREIRA, L. A.; SANTANA, M. J.; SANTANA, R. H. C.; NOBILE, P. N.; MONACO, F. J. Transient performance evaluation of cloud computing applications and dynamic resource control in large-scale distributed systems. In: **2015 International Conference on High Performance Computing & Simulation (HPCS)**. Amsterdam, The Netherlands: IEEE, 2015. p. 246–253. ISBN 978-1-4673-7812-3. Disponível em: <<http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=7237046>>. Citado 6 vezes nas páginas 28, 29, 34, 38, 81 e 92.

MATEEN, A.; RAZA, B.; HUSSAIN, T.; AWAIS, M. Autonomicity in Universal Database DB2. In: **2009 Eighth IEEE/ACIS International Conference on Computer and Information Science**. IEEE, 2009. p. 445–450. ISBN 978-0-7695-3641-5. Disponível em: <<http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5222927>>. Citado na página 103.

MENASCE, D. TPC-W: a benchmark for e-commerce. **IEEE Internet Computing**, v. 6, n. 3, p. 83–87, may 2002. ISSN 1089-7801. Disponível em: <<http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1003136>>. Citado 2 vezes nas páginas 68 e 76.

MENASCÉ, D. A.; ALMEIDA, V. A. F.; FONSECA, R.; MENDES, M. A. A Methodology for Workload Characterization of E-commerce Sites. In: **Proceedings of the 1st ACM conference on Electronic commerce - EC '99**. New York, New York, USA: ACM Press, 1999. p. 119–128. ISBN 1581131763. Disponível em: <<http://portal.acm.org/citation.cfm?doid=336992.337024>>. Citado 3 vezes nas páginas 65, 66 e 91.

MI, N.; CASALE, G.; CHERKASOVA, L.; SMIRNI, E. Injecting Realistic Burstiness to a Traditional Client-server Benchmark. In: **Proceedings of the 6th International Conference on Autonomic Computing**. New York, New York, USA: ACM Press, 2009. p. 149–158. ISBN 978-1-60558-564-2. Disponível em: <<http://doi.acm.org/10.1145/1555228.1555267>>. Citado na página 93.

_____. Sizing multi-tier systems with temporal dependence: benchmarks and analytic models. **Journal of Internet Services and Applications**, v. 1, n. 2, p. 117–134, nov 2010. ISSN 1867-4828. Disponível em: <<http://www.springerlink.com/index/10.1007/s13174-010-0012-9>>. Citado na página 93.

Minqi Li; YANG, F.; XU, J. A metamodeling-based approach for production planning. In: **Proceedings of the Winter Simulation Conference 2014**. Piscataway, New Jersey: IEEE, 2014. p. 2204–2215. ISBN 978-1-4799-7486-3. Disponível em: <<http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=7020064>>. Citado 2 vezes nas páginas 41 e 111.

NAKAJIMA, J. Hybrid-virtualization: Enhanced virtualization for linux. **Proceedings of the Linux Symposium**, 2007. Disponível em: <<http://kernel.org/doc/ols/2007/ols2007v2-pages-87-96.pdf>>. Citado na página 61.

NGAN, L. D.; S, T. F.; KEONG, C. C.; KANAGASABAI, R. Towards a Common Benchmark Framework for Cloud Brokers. In: **2012 IEEE 18th International Conference on Parallel and Distributed Systems**. IEEE, 2012. p. 750–754. ISBN 978-1-4673-4565-1. ISSN 15219097. Disponível em: <<http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6413611>>. Citado na página 27.

NOBILE, P. N. **Projeto de um broker de gerenciamento adaptativo de recursos em computação em nuvem baseado em técnicas de controle realimentado**. 164 p. Tese (Tese) — University of São Paulo, 2013. Disponível em: <<http://www.teses.usp.br/teses/disponiveis/55/55134/tde-21062013-110725/en.php>>. Citado 11 vezes nas páginas 28, 34, 39, 40, 79, 82, 83, 84, 97, 100 e 101.

OGATA, K. **Modern Control Engineering**. 5th. ed. [S.l.: s.n.], 2010. 905 p. ISBN 9780136156734. Citado na página 82.

PAPADOPOULOS, A. V.; MAGGIO, M.; TERRANEO, F.; LEVA, A. A dynamic modeling framework for control-based computing system design. **Mathematical and Computer Modelling of Dynamical Systems**, v. 21, n. 3, p. 251–271, 2015. Disponível em: <<http://dx.doi.org/10.1080/13873954.2014.942785>>. Citado na página 111.

PAREKH, S.; GANDHI, N.; HELLERSTEIN, J. L.; TILBURY, D.; TAYRAM, T.; BIGUS, J. Using control theory to achieve service level objectives in performance management. **Journal Real-Time Systems**, Kluwer Academic Publishers, v. 23, n. 2, p. 127–141, 2002. ISSN 0922-6443. Disponível em: <<http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=918084>>. Citado 2 vezes nas páginas 42 e 82.

PEREIRA, L. A.; MAMANI, E. L.; SANTANA, M. J.; SANTANA, R. H.; NOBILE, P. N.; MONACO, F. J. Non-stationary Simulation of Computer Systems and Dynamic Performance Evaluation: A Concern-Based Approach and Case Study on Cloud Computing. In: **2015 27th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)**. Florianópolis, Brasil: IEEE, 2015. p. 130–137. ISBN 978-1-4673-8011-9. Disponível em: <<http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=7379843>>. Citado 3 vezes nas páginas 29, 31 e 36.

RAJ, H.; SCHWAN, K. High performance and scalable I/O virtualization via self-virtualized devices. In: **Proceedings of the 16th international symposium on High performance distributed computing - HPDC '07**. New York, New York, USA: ACM Press, 2007. p. 179. ISBN 9781595936738. Disponível em: <<http://portal.acm.org/citation.cfm?doid=1272366.1272390>>. Citado 2 vezes nas páginas 57 e 61.

RAPPA, M. A. The utility business model and the future of computing services. **IBM Systems Journal**, v. 43, n. 1, p. 32–42, 2004. ISSN 0018-8670. Disponível em: <<http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5386779>>. Citado na página 45.

Ronghua Zhang; Chenyang Lu; ABDELZAHAR, 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. ISBN 0-7695-1585-1. ISSN 1063-6927. Disponível em: <<http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1022267>>. Citado na página 42.

RUSSELL, R. virtio: Towards a De-Facto Standard For Virtual I/O Devices. **ACM SIGOPS Operating Systems Review**, v. 42, n. 5, p. 95–103, jul 2008. ISSN 01635980. Disponível em: <<http://portal.acm.org/citation.cfm?doid=1400097.1400108>>. Citado na página 56.

SCHAFFER, H. E. X as a Service, Cloud Computing, and the Need for Good Judgment. **IT Professional**, v. 11, n. 5, p. 4–5, sep 2009. ISSN 1520-9202. Disponível em: <<http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5271524>>. Citado na página 47.

SCHRIBER, T. J.; BRUNNER, D. T.; SMITH, J. S. Inside discrete-event simulation software: How it works and why it matters. In: **Proceedings of the Winter Simulation Conference 2014**. Piscataway, New Jersey: IEEE, 2014. p. 132–146. ISBN 978-1-4799-7486-3. ISSN 08917736. Disponível em: <<http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=7019884>>. Citado 2 vezes nas páginas 31 e 33.

SINGH, R.; SHARMA, U.; CECCHET, E.; SHENOY, P. Autonomic mix-aware provisioning for non-stationary data center workloads. In: **Proceeding of the 7th international conference on Autonomic computing - ICAC '10**. New York, New York, USA: ACM Press, 2010. p. 21. ISBN 9781450300742. Disponível em: <<http://dl.acm.org/citation.cfm?doid=1809049.1809053>>. Citado na página 42.

SOSINSKY, B. **Cloud Computing Bible**. 1. ed. Indianapolis: Wiley Publishing, Inc., 2011. 497 p. ISBN 0470903562. Disponível em: <<https://books.google.com.br/books?id=hvv2pDEAbOEK>>. Citado 4 vezes nas páginas 44, 47, 49 e 50.

SOUNDARARAJAN, V.; ANDERSON, J. M. The impact of management operations on the virtualized datacenter. In: **Proceedings of the 37th annual international symposium on Computer architecture - ISCA '10**. New York, New York, USA: ACM Press, 2010. p. 326. ISBN 9781450300537. Disponível em: <<http://portal.acm.org/citation.cfm?doid=1815961.1816003>>. Citado na página 46.

SOUZA, F. L. d. S. de. **Extensão da geração de carga do Bench4Q para benchmark de desempenho em regime transiente**. 49 p. Tese (Doutorado) — Universidade de São Paulo, 2016. Citado na página 93.

STORM, A. J.; GARCIA-ARELLANO, C.; LIGHTSTONE, S. S.; DIAO, Y.; SURENDRA, M. Adaptive self-tuning memory in DB2. **Proceedings of the 32nd international conference on Very large data bases**, p. 1081–1092, 2006. Disponível em: <<http://dl.acm.org/citation.cfm?id=1182635.1164220>>. Citado na página 103.

TULLOCH, M. **Understanding Microsoft Virtualization Solutions**. 2. ed. Redmond, Washington 98052-6399: Microsoft Press, 2010. 480 p. Disponível em: <<http://download.microsoft.com/download/5/B/4/5B46A838-67BB-4F7C-92CB-EABCA285DFDD/693821ebook.pdf>>. Citado 4 vezes nas páginas 53, 54, 55 e 57.

UHLIG, R.; NEIGER, G.; RODGERS, D.; SANTONI, A.; MARTINS, F.; ANDERSON, A.; BENNETT, S.; KAGI, A.; LEUNG, F.; SMITH, L. Intel virtualization technology. **Computer**, v. 38, n. 5, p. 48–56, may 2005. ISSN 0018-9162. Disponível em: <<http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1430631>>. Citado 2 vezes nas páginas 55 e 56.

VAQUERO, L. M.; RODERO-MERINO, L.; CACERES, J.; LINDNER, M. A Break in the Clouds: Towards a Cloud Definition. **ACM SIGCOMM Computer Communication Review**, v. 39, n. 1, p. 50, dec 2008. ISSN 01464833. Disponível em: <<http://portal.acm.org/citation.cfm?doid=1496091.1496100>>. Citado 2 vezes nas páginas 45 e 46.

VELTE, A. T.; VELTE, T. J.; ELSENPETER, R. **Cloud Computing: A Practical Approach**. 1. ed. McGraw-Hill, 2009. 334 p. ISBN 978-0-07-162694-1. Disponível em: <http://books.google.com.br/books/about/Cloud_Computing_A_Practical_Approach.html?hl=pt-BR&id=mf0LMXve2gEC>. Citado 2 vezes nas páginas 46 e 48.

VERMA, A.; BAGRODIA, J.; JAISWAL, V. Virtual machine consolidation in the wild. In: **Proceedings of the 15th International Middleware Conference on - Middleware '14**. [s.n.], 2014. p. 313–324. ISBN 9781450327855. Disponível em: <<http://dl.acm.org/citation.cfm?doid=2663165.2663316>>. Citado na página 43.

VMWARE. **Understanding Full Virtualization, Paravirtualization, and Hardware Assist**. [S.l.], 2007. 17 p. Disponível em: <http://www.vmware.com/files/pdf/VMware_paravirtualization.pdf>. Citado 4 vezes nas páginas 57, 58, 59 e 60.

WALDSPURGER, C.; ROSENBLUM, M. I/O virtualization. **Communications of the ACM**, v. 55, n. 1, p. 66, jan 2012. ISSN 00010782. Disponível em: <<http://dl.acm.org/citation.cfm?doid=2063176.2063194>>. Citado na página 61.

WANG, R.; KANDASAMY, N. On the design of decentralized control architectures for workload consolidation in large-scale server clusters. In: **Proceedings of the 9th international conference on Autonomic computing - ICAC '12**. New York, New York, USA: ACM Press, 2012. v. 16, n. 2, p. 115. ISBN 9781450315203. Disponível em: <<http://dl.acm.org/citation.cfm?doid=2371536.2371558>>. Citado na página 87.

WANG, R.; KUSIC, D. M.; KANDASAMY, N. A distributed control framework for performance management of virtualized computing environments. In: **Proceeding of the 7th international conference on Autonomic computing - ICAC '10**. New York, New York, USA: ACM Press, 2010. p. 89. ISBN 9781450300742. Disponível em: <<http://portal.acm.org/citation.cfm?doid=1809049.1809066>>. Citado na página 89.

YANG, F.; LIU, J. Transient analysis of general queueing systems via simulation-based transfer function modeling. In: **Proceedings of the 2010 Winter Simulation Conference**. Piscataway, New Jersey: IEEE, 2010. p. 1110–1122. ISBN 978-1-4244-9866-6. Disponível em: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5679079>>. Citado 3 vezes nas páginas 31, 41 e 111.

YAZIR, Y. O.; MATTHEWS, C.; FARAHBOD, R.; NEVILLE, S.; GUITOUNI, A.; GANTI, S.; COADY, Y. Dynamic Resource Allocation in Computing Clouds Using Distributed Multiple Criteria Decision Analysis. In: **2010 IEEE 3rd International Conference on Cloud Computing**. IEEE, 2010. p. 91–98. ISBN 978-1-4244-8207-8. Disponível em: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5558005>>. Citado na página 79.

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, apr 2010. ISSN 1867-4828. Disponível em: <http://www.springerlink.com/index/10.1007/s13174-010-0007-6>>. Citado 2 vezes nas páginas 45 e 48.

ZHANG, W.; WANG, S.; WANG, W.; ZHONG, H. Bench4Q: A QoS-Oriented E-Commerce Benchmark. In: **2011 IEEE 35th Annual Computer Software and Applications Conference**. IEEE, 2011. v. 20000, p. 38–47. ISBN 978-1-4577-0544-1. Disponível em: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6032323>>. Citado 8 vezes nas páginas 27, 62, 64, 65, 66, 67, 76 e 84.

ZHANG, Y.; HUANG, G.; LIU, X.; MEI, H. Tuning Adaptive Computations for Performance Improvement of Autonomic Middleware in PaaS Cloud. In: **2011 IEEE 4th International Conference on Cloud Computing**. IEEE, 2011. p. 732–733. ISBN 978-1-4577-0836-7. Disponível em: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6008677>>. Citado na página 49.

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. ISBN 978-1-4577-1637-9. Disponível em: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6138531>>. Citado na página 79.

CONFIGURAÇÃO DO AMBIENTE

A.1 Configuração da camada de aplicação

O servidor de aplicações *WebSphere* inicialmente está configurado com parâmetros que não permitem a execução adequada em um ambiente com níveis elevados de carga de trabalho. Por exemplo, o número de *threads* que o servidor pode utilizar para o processamento interno das requisições é um parâmetro que precisa ser adequado para alcançar o desempenho esperado (alto).

Por padrão, o valor do número de *threads* está definido em 500. Para os experimentos realizados nas avaliações com um valor referencial a 800 EBs (alto) notou-se que essa quantidade é insuficiente. Dado um experimento com duração de 400 segundos e com 800 EBs realizando consultas de só leitura (select), o sistema torna-se inestimável.

Por esse motivo, recomenda-se mudar o valor predefinido de 500 para 700 *threads*. Essa alteração pode ser efetuada editando o arquivo de configuração `config-substitutions.properties`. Considerando uma instalação com permissões de super usuário o *path* é apresentados a seguir:

```
/opt/IBM/WebSphere/AppServerCommunityEdition/var/config/config-  
substitutions.properties
```

Deve ser procurada a variável *MaxThreadPoolSize*:

```
#MaxThreadPoolSize = 500  
MaxThreadPoolSize = 700
```

A.1.1 Configurações do WebSphere

O parâmetro de número de *threads* afeta diretamente à quantidade de memória utilizada pelo servidor. Além disso, para ambientes com níveis de carga de trabalho altos, é recomendável aumentar a quantidade de memória que a máquina virtual Java aloca para o servidor Web. O nome do arquivo relacionado com essa alteração é *geronimo.sh*, e encontra-se na pasta:

`/opt/IBM/WebSphere/AppServerCommunityEdition/bin/geronimo.sh`

Dentro do arquivo, deve ser localizada a variável que define os parâmetros enviados à máquina virtual Java. Aproximadamente na linha 194 encontra-se a linha que define as opções utilizadas pela máquina virtual Java para a execução do servidor de aplicações Web.

```
JAVA_OPTS= "-Xmx512m -XX:MaxPermSize=256m"
```

A.2 Instalação e configuração do banco de dados

Os experimentos realizados têm como objetivo a camada de aplicação dentro de sistemas *e-commerce*, mas a camada de banco de dados é de suma importância, principalmente pelo tipo de experimentos (alto desempenho) realizados com o *benchmark*. Por esse motivo, deve ser configurada de modo personalizado. A instalação do banco de dados ¹ DB2 é descrita a seguir:

```
sudo apt-get install alien binutils build-essential debhelper gawk gettext html2text  
intltool-debian ksh lib32bz2-dev lib32z1-dev libaio-dev libaio1 libbeecrypt7 libc6 libc6-dev  
libc6-dev libdb5.3 libelf-dev libelf1 libltdl-dev libltdl7 libmotif4 libodbcinstq4-1 libqt4-core  
libqt4-gui libsqlite3-0 lsb lsb-core lsb-cxx lsb-desktop lsb-graphics lsb-qt4 make odbcinst  
openjdk-7-jdk pax po-debconf rpm rpm-common sysstat tzdata-java unixodbc unixodbc-dev  
unzip
```

Deverá ser criado o arquivo `/etc/sysctl.d/99-ibm.conf` e adicionar as seguintes linhas:

```
fs.aio-max-nr = 1048576  
fs.file-max = 6815744  
kernel.shmall = 2097152  
kernel.shmmax = 536870912  
kernel.shmmni = 4096
```

¹ Baseada em: <http://hswong3i.net/blog/hswong3i/db2-express-c-10-1-ubuntu-12-04-howto>

```
kernel.sem = 250 32000 100 128
net.ipv4.ip_local_port_range = 9000 65500
net.core.rmem_default = 262144
net.core.rmem_max = 4194304
net.core.wmem_default = 262144
net.core.wmem_max = 1048586
```

Executar o seguinte comando para recarregar os parâmetros do kernel sem reiniciar a máquina:

```
service procps start
```

Depois dos passos anteriores, será possível iniciar a instalação com os seguintes comandos:

```
ssh user@db-server
sudo ./db2setup
```

Devem ser levados em consideração os nomes dos usuários de sistema criados pelo instalador, porque na pasta do usuário db2inst1 é onde é criada a instancia do banco de dados. Para definir de maneira personalizada os valores relacionados com a memória, devem ser executados o comandos a seguir:

```
lasdpc@desk-11:~$ getconf PAGE_SIZE
4096
```

Com as informações anteriores o arquivo `/etc/sysctl.conf` deve ser editado:

```
# with 12GB
# 12*1024*1024*1024/4096
kernel.shmall = 3145728
kernel.shmmax = 12884901888
```

A.2.1 Criação do banco de dados no DB2

Segundo a documentação do DB2, para iniciar o processo de criação da instancia do banco de dados, deve ser utilizando o usuário db2inst1:

```
ssh db2inst1@database-server
```

O usuário *db2inst1* proporciona um ambiente que permite executar os comandos de criação de instancias de banco de dados, a seguir são apresentados os comandos necessários para esse fim:

```
db2 create db JASPTE
db2 connect to JASPTE
db2stop force
db2start
```