Extensão da geração de carga do Bench4Q para benchmark de desempenho em regime transiente.

Flavio Luiz dos Santos de Souza

SERVIÇO DE PÓS-GRADUAÇÃO DO ICMC-USP

Data de Depósito:

Assinatura: _____

Flavio Luiz dos Santos de Souza

Extensão da geração de carga do Bench4Q para benchmark de desempenho em regime transiente.

Dissertação apresentada ao Instituto de Ciências Matemáticas e de Computação – ICMC-USP, como parte dos requisitos para obtenção do título de Mestre 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
Junho 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)

S634e

Souza, Flavio Luiz dos Santos de
Extensão da geração de carga do Bench4Q para benchmark de desempenho em regime transiente. / Flavio Luiz dos Santos de Souza; orientador Francisco José Monaco. - São Carlos - SP, 2016.
75 p.

Dissertação (Mestrado - Programa de Pós-Graduação em Ciências de Computação e Matemática Computacional) - Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, 2016.

1. Avaliação de Desempenho. 2. *Benchmark*. 3. Computação em Nuvem. 4. Modelagem. I. Monaco, Francisco José, orient. II. Título.

Flavio Luiz dos Santos de Souza

Extension of the load generation for Bench4Q benchmark performance transient regime.

Master dissertation submitted to the Instituto de Ciências Matemáticas e de Computação – ICMC-USP, in partial fulfillment of the requirements for the degree of the Master 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
June 2016

Este trabalho é dedicado a Deus, o que seria de mim sem a fé que eu tenho nele.

À minha família, por sua capacidade de acreditar em mim e investir em mim. Mãe, seu cuidado e dedicação foi que deram, em alguns momentos, a esperança para seguir. Pai, sua presença significou segurança e certeza de que não estou sozinho nessa caminhada.

À minha queria amada esposa, que soube entender a minha ausência nos muitos momentos desde que ingressei no mestrado até a sua conclusão, e por me aguentar meus momentos de ansiedade e estresse.

Ao professor Monaco, por seus ensinamentos, pelo convívio, apoio, compreensão e pela amizade; e juntamente com a Tarsila me ajudaram a superar um dos momentos mais difíceis da minha vida, nunca irei esquecer destes momentos que me acolheram, muito obrigado.

A todos aqueles que de alguma forma estiveram e estão próximos de mim, fazendo esta vida valer cada vez mais a pena.

AGRADECIMENTOS

Os agradecimentos principais são direcionados à Deus, pelas oportunidades que me confiastes e permitir a realização dos meus sonhos. Aos meus pais, Francisca e Pedro e à minha esposa Gleice, pelo amor, palavras de apoio, carinho e pela compreensão e paciência que sempre tiveram.

Ao meu orientador, Francisco José Monaco. Agradeço pela esmerada orientação, pela confiança, paciência, contribuições e ensinamento. Ao Edwin, Lourenço, Renê, Bruno e a todos que me auxiliaram durante o mestrado, que muitas vezes tiveram que abrir mão do tempo com as suas famílias para tirar minhas dúvidas e me ajudar a resolver os diversos problemas enfrentados no caminho e a todos os colegas e amigos do LaSDPC.

Um saudoso agradecimento ao meus amigos, Arthur, Carlos (Carlota), Frederico (Fred), Venilton (Vernis), sem vocês não sei se teria chegado até o final, agradeço muito por tudo o que fizeram.

Um agradecimento especial direcionado a Universidade de São Paulo (USP) - Campus de São Carlos, pela oportunidade de realizar um grande sonho no curso de Pós-Graduação.

“O conhecimento serve para encantar as pessoas,
e não para humilhá-los.”
(Mário Sérgio Cortella)

RESUMO

SOUZA, F. L.. **Extensão da geração de carga do Bench4Q para benchmark de desempenho em regime transiente.**. 2016. 75 f. Dissertação (Mestrado em Ciências – Ciências de Computação e Matemática Computacional) – Instituto de Ciências Matemáticas e de Computação (ICMC/USP), São Carlos – SP.

Este trabalho de mestrado apresenta o desenvolvimento de uma extensão no *benchmark* Bench4Q. A extensão proposta é uma nova funcionalidade para o *benchmark*. O referido *framework* é utilizado para gerar carga sintética para um sistema *e-commerce* acoplado ao *benchmark*. Seu principal emprego na literatura tem sido em avaliação de desempenho sob carga estacionária. Contudo, recentes pesquisas tem apresentado interesse no estudo de arquiteturas adaptativas de autogerenciamento de recursos, o que implica em responder às perturbações e atender os requisitos de desempenho em regime transiente propostos para o sistema. No entanto, este *benchmark* não abrange os estados transiente do sistema. O presente trabalho tem por objetivo estender o *benchmark* Bench4Q acrescentando-lhe capacidade de excitar a resposta transiente do sistema mediante as perturbações da carga de trabalho. Para isso, o *software* foi acrescido de funcionalidade capaz de gerenciar a modulação da carga de trabalho. Os experimentos foram executados em um ambiente multicamadas que apresentou resultados compatíveis ao objetivo, representando contribuições para a área de avaliação de desempenho. A motivação da pesquisa, inserção em outros trabalhos em andamento e direções futuras são introduzidas.

Palavras-chave: Avaliação de Desempenho, *Benchmark*, Computação em Nuvem, Modelagem.

ABSTRACT

SOUZA, F. L.. **Extensão da geração de carga do Bench4Q para benchmark de desempenho em regime transiente.**. 2016. 75 f. Dissertação (Mestrado em Ciências – Ciências de Computação e Matemática Computacional) – Instituto de Ciências Matemáticas e de Computação (ICMC/USP), São Carlos – SP.

This master thesis introduces the development of an extension for the Bench4Q benchmark. The referred framework is utilized to generate sinthetic workload for a companion e-commerce benchmark. The software package Bench4q is a benchmark for cloud computing applications which simulates various aspects of conventional architectures and workloads in this kind of environment. It is mainly referenced in the literature in works on performance evaluation under stationary load. Recent research works have broaden its interest to the study of adaptive architectures of resource self-management, what implies in responding to disturbances and meeting performance requirements in transient regime. This work aims at extending Bench4q adding it capabilities to excite the transient response of the system by means of applying disturbances during execution time. To this end, the piece of software shall be enriched with functionalities for generating non-stationary workload and programmed disturbances. Experients have been carried out in a multi-layer enviroment and have yielded positive result, representing contributions to the state of the art. The motivation of this piece of work, insertion in other ongoing research and directions are introduced.

Key-words: Performance Evaluation, Benchmark, Cloud Computing, Model.

LISTA DE ILUSTRAÇÕES

Figura 1 – Tempo de resposta do *Black Friday* Brasil 2012 3
Figura 2 – Diagrama de blocos de um sistema de controle 8
Figura 3 – Dinâmica do aquecimento da água 9
Figura 4 – Dinâmica da distensão de uma mola 10
Figura 5 – Dinâmica de controlador (LUZ, 2014) 11
Figura 6 – Arquitetura conceitual MEDC 12
Figura 7 – Sinais em tempo discreto comuns, parte 1 13
Figura 8 – Sinais em tempo discreto comuns, parte 2 14
Figura 9 – Arquitetura Bench4Q 17
Figura 10 – Console Bench4Q 18
Figura 11 – SUT Bench4Q 19
Figura 12 – CBMG - perfil de navegação dos *brwosers* do Bench4Q 20
Figura 13 – Comparação da quantidade de requisições completadas com sucesso entre dois cenários: normal e otimizado não realisticamente. 21
Figura 14 – Carga de trabalho gerada pelo Bench4Q 22
Figura 15 – Tipo de sessões Bench4Q 23
Figura 16 – Possibilidade de cargas moduláveis pela extensão 26
Figura 17 – Arquitetura do experimento 27
Figura 18 – Comportamento de métrica transiente 29
Figura 19 – Configuração experimental Bench4Q. 32
Figura 20 – Diagrama de classes da extensão do Bench4Q. 34
Figura 21 – Console de programação da carga de trabalho. 38
Figura 22 – Teste de modulação da carga 39
Figura 23 – Carga gerada com base na configuração: Degrau Positivo 42
Figura 24 – Carga gerada com base na configuração: Degrau Negativo 43
Figura 25 – Carga gerada com base na configuração: Onda Quadrada 44
Figura 26 – Conexões por segundo vs. Tempo de resposta 45
Figura 27 – Conexões por segundo vs. Média de utilização de CPU nas VMs 46
Figura 28 – Conexões por segundo vs. Utilização de CPU no banco de dados 47
Figura 29 – Comparação da dinâmica entre Postgres e DB2 48
Figura 30 – Utilização da metodologia de avaliação de desempenho não-estacionário . . 49

LISTA DE CÓDIGOS-FONTE

Código-fonte 1 – Algoritmo calcula os tempos de inicialização e termino para cada um dos Clientes 33
Código-fonte 2 – Algoritmo de geração de carga modificado para modulação 36
Código-fonte 3 – Código para gerar a os parâmetros para a modulação 37
Código-fonte 4 – Algoritmo calcula os tempos de inicialização e termino para cada um dos Clientes 59
Código-fonte 5 – Algoritmo de geração de carga modificado para modulação 60
Código-fonte 6 – Código para gerar a os parâmetros para a modulação 64

LISTA DE TABELAS

Tabela 1 – Especificação do ambiente de execução dos experimentos 28
Tabela 2 – Fator e nível dos experimentos 30

LISTA DE ABREVIATURAS E SIGLAS

CBMG ... *Customer Behavior Model Graph*
EBs ..... *Emulations Browsers*
GNU ..... *Lesser General Public License*
HTTP .... *Hypertext Transfer Protocol*
LaSDPC .. Laboratório de Sistemas Distribuídos e Programação Concorrente
MEDC ... *Monitor, Effector, Demanda and Capacity*
QoS ..... Quality of service
S3 ..... Simple Storage Service
SPEC .... *Standard Performance Evaluation Corporation*
SUT ..... *System Under Test*
TPC-C ... *On-line Transaction Processing*
TPC-E ... *Complex on-line transaction processing*
TPC-H ... *Ad-hoc decision support system*
VMs ..... Virtual Machines
WIPS .... *Web Interaction Per Second*
WIRT .... *Web Interaction Response Time*

SUMÁRIO

1 INTRODUÇÃO 1
1.1 Motivação 4
1.2 Objetivo 5

2 ESTADO DA ARTE 7

3 METODOLOGIA 25

4 DESENVOLVIMENTO 31
4.1 Configuração da carga de trabalho 33
4.2 Geração da carga de trabalho 35
4.3 *Interface* gráfica 37
4.4 Teste de modulação 38

5 RESULTADOS 41
5.1 Impacto da Carga Modulada no ambiente 44
5.2 Contribuições 47

6 CONCLUSÃO 51
6.1 Trabalhos Futuros 53

REFERÊNCIAS 55

APÊNDICE A CÓDIGOS DA EXTENSÃO 59
A.1 Configuração da carga de trabalho 59
A.2 Geração da carga de trabalho 60
A.3 *Interface* gráfica 64

APÊNDICE B DOCUMENTAÇÃO DA EXTENSÃO DO BENCH4Q . 67

CAPÍTULO

1

INTRODUÇÃO

Para além da pesquisa acadêmica, a Computação em Nuvem, hoje se faz presente na vida de pessoas e empresas, integrando-se em suas rotinas e atividades do cotidiano e estabelecendo-se como um importante e fundamental meio computacional.

O conceito, Computação em Nuvem, tem se tornando um paradigma cada vez mais reconhecido e utilizado. Inúmeras empresas tem apoiado e incentivado o uso e o seu desenvolvimento, assim como o meio científico. Atualmente, os exemplos desse paradigma de computação oferecem diversas opções para armazenamento ou processamento de dados, tais como, a *Simple Storage Service (S3)* da Amazon, Bigtable do Google, ou PNUTS do Yahoo (BINNIG *et al.*, 2009). Vazquez, Krishnan e John (2014) exemplificam o crescimento do Instagram, que construiu a sua base de usuários com mais de 150 milhões de usuários em menos de quatro anos usando soluções em nuvem.

O aumento da popularidade da Computação em Nuvem é impulsionado pelas vantagens oferecidas e pelo modelo dinamicamente escalável e o rápido desenvolvimento nos últimos anos levou a uma grande quantidade de publicações. O interesse no meio acadêmico e da indústria tem levado a uma quantidade considerável de publicações de artigos científicos nos últimos anos (HEILIG; VOSS, 2014). Como apresentado nos trabalhos de Heilig e Voss (2014) e Yang e Tate (2012), o número de pesquisas tem concentrado esforços na resolução de problemas que em geral envolvem desempenho e/ou garantia de requisitos de Qualidade de Serviço (*Quality of service (QoS)*).

Nessa área existe uma grande preocupação por parte dos provedores de computação em garantir QoS de uma forma eficiente. Em termos de recursos, significa que os recursos virtuais (*Virtual Machines (VMs)*) e/ou físicos devem ser alocados de forma autônoma, para que possam responder às influências externas, como a carga de trabalho. Observa-se, por outro lado, os *data center* são muitas vezes subutilizados devido ao excesso de provisionamento de recursos, bem como as demandas de recursos que variam no tempo de acordo com os sistemas (PADALA

et al., 2007). Inomata *et al.* (2011) afirmam que tempo e dinheiro são investidos para projetar, construir, configurar, monitorar e manter recursos computacionais e que o futuro da Computação em Nuvem é o auto gerenciamento e a atribuição de recursos automáticos aos seus consumidores com base na carga de trabalho.

Atualmente, com o advento e crescimento das aplicações intensivas de dados, que lidam com grandes volumes em tempo real, a dinâmica do sistema passa ser apreciável, fenômeno frequentemente não perceptível claramente para os sistemas computacionais. Na prática, tais aplicações, de grande volume de dados, tendem a operarem com cargas de trabalho variante no tempo, causando inúmeras dificuldades. A Computação em Nuvem oferece uma infraestrutura elástica e escalável que pode ser utilizada para obter recursos sob demanda; no entanto, um problema em aberto é decidir sobre a correta alocação de recursos ao implantar um sistema na nuvem (CERVINO *et al.*, 2012). De modo geral, um sistema dinâmico não apresenta os efeitos das ações da carga de forma imediata. Por exemplo, a velocidade de um carro não atingem a sua velocidade máxima imediatamente ao acionamento do acelerador ou como a temperatura em uma sala muda instantaneamente quando um aquecedor é ligado.

Este comportamento, dinâmico, começa a ser apreciado em grandes sistemas computacionais, quando um súbito aumento de requisições para um *website* em que os servidores não conseguem-se ajustar à demanda de modo imediato.

O *burstiness* (rajada) no fluxo de requisições é frequentemente encontrado em sistemas cliente-servidor. Esse *burstiness* pode impactar de forma inesperada o desempenho de diferentes mecanismos de alocação de recursos projetados para o gerenciamento de um sistema adaptativo, e, portanto, testar e avaliar esses mecanismos sob cargas de trabalho reproduzíveis e controláveis é importante para o projeto do sistema. Como ilustração, durante o evento promocional de um *e-commerce* durante o *Black Friday*, na edição de 2012, foi constatado que o tempo de resposta médio aos dias que antecederam 23 de Novembro de 2012, era de 5.3 segundos. A partir das 7 horas do dia evento, observou-se um aumento significativo no tempo de resposta conforme apresentado no gráfico da Figura 1. Mesmo hospedado em um ambiente de Computação em Nuvem, o sistema não teve a expectativa de que o volume de requisições aumentaria drasticamente, de tal maneira a prejudicar e comprometer os níveis de QoS, como tempo de respostas.

As técnicas de gerenciamento de recursos são inúmeras, como as de inteligência artificial que utilizam lógica *fuzzy*, algoritmos de mineração de dados e aprendizado de máquina (NOBILE, 2013). As técnicas de aprendizado de máquina mostram-se interessantes para soluções imediatas, onde não há tempo hábil, inclusive, com abordagens não lineares empregadas para implementar um gerenciador de recursos responsável pela alteração dinâmica das capacidades computacionais. Outros trabalhos, como o Zhang *et al.* (2007), buscam identificar padrões de comportamento na carga de trabalho e otimizar a provisão dinâmica dos recursos nas máquinas virtuais com base em algoritmos reativos. Quiroz *et al.* (2009) buscam detectar, por monitoramento, a padronização e a tendência da carga de trabalho com o objetivo de otimizar os recursos. Nobile (2013) também

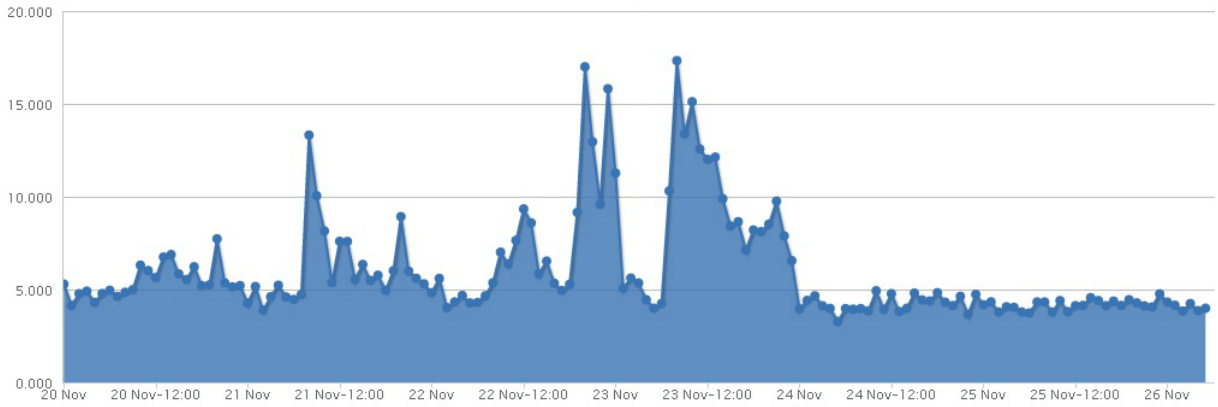


Figura 1 – Tempo de resposta do *Black Friday* Brasil 2012

Fonte: Adaptada de E-commerce Brasil (2012).

propõe o uso de modelos de séries temporais para modelagem de tráfego de tempo real e previsão de demanda, que é usada para criar partições de conexões para diferentes classes de prioridade. Zhang *et al.* (2011b), utiliza de técnicas de estatística e avaliação dos recursos disponíveis para tomar a decisão de qual recurso deve ser alocado.

Huang, He e Miao (2014) afirmam que atualmente a maioria das aplicações Web são concebidas com sistemas *mult-tiers* (multi-camadas) devido à flexibilidade de escalabilidade.

Júnior *et al.* (2015b) afirmam que a partir de um planejamento de capacidade, e a gestão de recursos em tempo real aumenta a sofisticação da arquitetura da solução nos diversos níveis e complexidade, as abordagens convencionais para análise de sistemas computacionais modernos; e concluí que as dinâmicas emergentes das interligações desses sistemas *mult-tiers* pode produzir efeitos transitórios sobre o desempenho, como resposta temporal, ou o amortecimento e/ou comportamento oscilatório. Uma vez que estes efeitos variáveis no tempo podem potencialmente afetar a capacidade de resposta, eficiência e até mesmo a estabilidade global, métodos e ferramentas para avaliação das propriedades dinâmicas de sistemas computacionais são de importância prática para fins da engenharia.

Há caso em que existe o interesse em controlar a dinâmica deste sistema, para tanto é necessário modelar o sistema em questão e projetar um controlador que manipulará a sua dinâmica. Para trabalhar com estes sistemas deve-se ser capaz de modelar a dinâmico do sistema, que em termos matemáticos é extrair um modelo matemático e analisar suas características dinâmicas. Um modelo matemático de um sistema dinâmico é definido como um conjunto de equações que representa a dinâmica de maneira relativamente razoável. Percebe-se que um modelo matemático não é exclusivo para um determinado sistema. Um sistema pode ser representando de muitas maneiras diferentes e, por conseguinte, podem ter diversos modelos matemáticos, dependendo da perspectiva, uns mais precisos que outros, e outros mais simplificados (OGATA, 2001).

Algumas vezes, é possível representar sistemas dinâmicos através de equações diferenci-

ais obtidas com base nas leis básicas da física. Esses sistemas de equações diferenciais descrevem uma determinada dinâmica do sistema no domínio do tempo e, com o desenvolvimento dessas equações, é possível identificar propriedades fundamentais do sistema (NOBILE, 2013). Quando a modelagem é muito dedutiva e muito complexa pode-se recorrer a métodos empíricos de identificação. Nesses, o modelo matemático é induzido mediante a comparação de entrada e saída.

A avaliação por aferição, comparação de entrada e saída, medição direta via instrumentação apropriada adequa-se nesse caso; no entanto, é necessário a existência e disponibilidade do sistema ou protótipo, pois a avaliação é feita através de estímulos às entradas. A leitura de suas saídas, possibilita testes de caixa preta em que não são exigidos conhecimentos sobre o funcionamento interno do sistema (NOBILE, 2013), como em um sistema dinâmico, no qual o comportamento do sistema evolui com o tempo, em resposta a estímulos externos.

1.1 Motivação

Embora amplamente aplicada e difundida em diversas áreas da engenharia e ciência, a avaliação de desempenho em regime transitório é pouco explorada e usada em sistemas computacionais, em função de que sistemas computacionais e suas aplicações não tem necessita dessa análise. O desenvolvimento, contudo, de sistemas distribuídos de larga escala e de alta complexidade altera essa realidade (MAMANI *et al.*, 2015; JúNIOR *et al.*, 2015b; JúNIOR *et al.*, 2015a), devido ao comportamento dinâmico que não se apresenta nitidamente em avaliações estacionarias convencionais na computação.

No *Laboratório de Sistemas Distribuídos e Programação Concorrente (LaSDPC)*[1], onde desenvolve este projeto, trabalhos anteriores lidam com essa temática. O trabalho de Nobile (2013), um sistema com características dinâmicas, hospedado em um ambiente de computação em nuvem gerencia recursos elásticos por meio de mecanismos de provisão de QoS, técnicas de teoria de controle. Em outro trabalho, Júnior *et al.* (2015b) apresenta uma especificação de uma arquitetura conceitual que separa responsabilidades de simulação dinâmica em um conjunto de aspectos básicos, e formaliza um modelo de referência abstrata para a concepção de ferramentas de simulação. O trabalho de Mamani (2015), em desenvolvido também no LaSDPC, estuda e define uma metodologia de análise transiente dedicado a sistemas computacionais dinâmicos reais utilizando a especificação arquitetural proposto por Júnior *et al.* (2015b). A principal contribuição pretendida é a formulação e definição de uma metodologia para seu emprego em sistemas, cuja a metodologia deverá ser capaz de descrever e especificar os passos para modelar o sistema, e analisar os resultados transiente mediante as variações na carga de trabalho. Para tanto, o trabalho de Mamani (2015) utiliza de um *benchmark* para a validação experimental de sua metodologia. No entanto, o *benchmark* escolhido por Mamani (2015), Bench4Q, não

[1] <http://www.lasdpc.icmc.usp.br>

contempla as especificações apresentadas por Júnior *et al.* (2015b).

Segundo Binnig *et al.* (2009) os *benchmarks* tradicionais não são suficientes para a análise desses novos serviços de elasticidade da Computação em Nuvem. O principal desafio dos novos *benchmarks* são fazer com que as métricas ofereçam informações relevantes a esses diferentes serviços e com diferentes capacidades e garantias desses serviços. Huang, He e Miao (2014) afirmam que, a maioria das aplicações Web são concebidas como sistemas de *mult-tiers*, devido à flexibilidade e capacidade de reutilização de *software*, porem é difícil de modelar o comportamento de aplicações Web de *multi-tiers*, devido ao fato de que a carga de trabalho estimula a dinâmica do sistema nos diferentes níveis da camada.

No âmbito da análise de desempenho em sistemas computacionais, definimos o *benchmarking* como o ato de medir e avaliar o desempenho computacional, protocolos de rede, dispositivos e redes, sob condições de referência, em relação a uma avaliação de referência. O objetivo deste processo de *benchmarking* é permitir a comparação equitativa por diferentes soluções, ou entre desenvolvimentos subsequentes de um *System Under Test (SUT)*. *Benchmarking* é o principal método para medir o desempenho de uma máquina ou sistema.

Apesar de existir diversos *benchmarks* e ferramentas para o estudo, nenhuma delas estimulam a dinâmica do sistema e permite uma avaliação em regime transiente, que se faz necessário para a pesquisa de Mamani (2015). A proposta deste trabalho é modificar o *benchmark* definido por Mamani (2015) e adequá-lo de maneira em que estimule a dinâmica do sistema possibilitando uma avaliação transiente.

1.2 Objetivo

Este trabalho tem por objetivo a extensão do *framework* de *benchmark* Bench4Q afim de atender os requisitos do modelo *Monitor, Effector, Demanda and Capacity (MEDC)*, proposto por Júnior *et al.* (2015b). O objetivo restringe-se ao módulo de modulação da carga de trabalho, gerado pelo *benchmark*, acrescendo-o de provisões nativas para gerar perturbações capazes de excitar e produzir o regime transiente do sistema SUT do *benchmark*, permitindo apreciar-se sua dinâmica. A contribuição almejada é a disponibilização de um *benchmark* que auxilie a análise de sistemas dinâmico e que possibilite a analise transiente.

CAPÍTULO

2

ESTADO DA ARTE

Nos últimos anos, com o aumento da popularidade, a Computação em Nuvem tem atraído a atenção da indústria e do mundo acadêmico, tornando-se cada vez mais comum na literatura cientifica e técnica, e com grande adoção por parte das empresas e instituições de pesquisa, impulsionadas pelas vantagens oferecidas pelo modelo dinâmico e escalável O *pay-as-you-go* é um modelo que permite uma aplicação cresça naturalmente com a demanda, possibilitando a adição de recursos de uma forma dinâmica e elástica (VAZQUEZ; KRISHNAN; JOHN, 2014). A elasticidade é uma característica no contexto Computação em Nuvem e, o que distingue este paradigma de computação dos demais. Como os recursos escaláveis da Computação em Nuvem, as aplicações reduzem os riscos de excesso de provisionamento, e o desperdício de recursos durante o horário de baixa utilização (VAZQUEZ; KRISHNAN; JOHN, 2014; GALANTE; BONA, 2012).

A elasticidade permite aos usuários contratante adquirirem recursos dinamicamente de acordo com respectivas demandas e necessidade, mas decidir a quantidade correta desses recursos não é uma tarefa trivial. Na verdade, o dimensionamento adequado de recursos para aplicações computacionais é uma questão crucial na Computação em Nuvem. Em situações previsíveis, os recursos podem ser provisionados com antecedência através de técnicas de planejamento de capacidade, com pouca divisibilidade, contudo seria desejável uma automatização da escalabilidade do sistema que ajusta os recursos disponíveis a uma aplicação com base em suas necessidades. O problema de dimensionamento automático que pode ser abordado utilizando-se diferentes abordagens (LORIDO-BOTRÁN; MIGUEL-ALONSO; LOZANO, 2012).

Nesse contexto, novos temas de pesquisas começam a chamar a atenção e muitos destes têm apresentado soluções de alocação dinâmica de recursos, principalmente, em situações onde há variação rápida da carga de trabalho ou das características internas do sistema.

Nobile (2013) apresenta uma solução que consiste na proposta de uma arquitetura de gerenciamento adaptativo de recursos. O trabalho inspira-se em técnicas de controle realimentado

para alocação dinâmica de recursos conforme apresentado na Figura 2. A Figura 2 é um diagrama de blocos que representa a malha fecha do sistema de controle retroalimento implantado no sistema afim de controlar os recursos conforme o impacto da variação da carga de trabalho (entrada) no próprio recurso. Cada figura geométrica e letras/símbolos tem significados específicos na teórica de controle, tema que não é de interesse deste trabalho, portanto, estes não serão explicados neste trabalho.

O que nos interessa saber é que a teoria de controle tem se mostrado proeminente nas áreas da engenharia e algumas ciências naturais, e conta um vasto conjunto de ferramentas de modelagem matemáticas que auxiliam na descrição do comportamento de sistemas dinâmicos frente a diferentes estímulos em regime estacionário e transiente (NOBILE, 2013). Nesse foco, a dinâmica do sistema passa a ser um ponto chave para o planejamento e aplicação de técnicas de teoria de controle em sistemas computacionais.

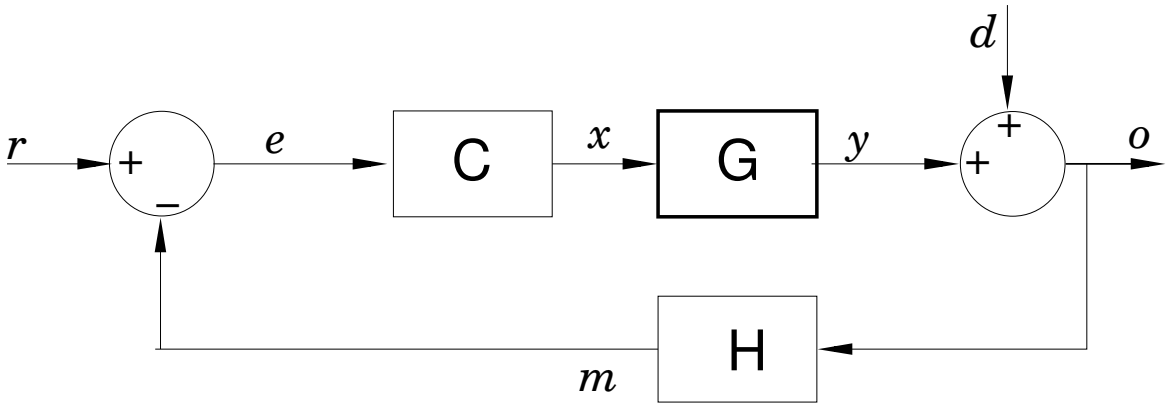


Figura 2 – Diagrama de blocos de um sistema de controle

Fonte: Nobile (2013).

Nobile (2013) foi ao encontro de soluções adaptativas para o gerenciamento de recursos e de alta disponibilidade do sistema computacional na Computação em Nuvem. O trabalho optou por aplicar técnicas de adaptação ainda pouco exploradas nas computação, mas de reconhecimento há décadas por outras áreas da ciência e engenharia. Um ferramental que auxilia na descrição do comportamento de sistemas dinâmicos, em termos de como eles respondem a diferentes estímulos em regime estacionário e transiente.

O adjetivo dinâmico refere-se aos fenômenos com uma reação retardada. Pela inercia intrínseca. É possível entender um sistema dinâmico no contexto lógico de definição simples como um modelo matemático, mas um modelo matemático em que os objetos de interesse estão em função do tempo: o universo é um espaço de funções.

Sistemas dinâmicos são usados para modelar fenômenos físicos cujo estado ou descrição instantânea muda com o tempo ainda que a entrada permaneça constante. Este modelo é utilizado, como por exemplo, na previsão econômica e financeira, modelagem ambiental, diagnóstico médico, o equipamentos industriais, e uma série de outras aplicações (DEAN; WELLMAN,

1991).

O comportamento dinâmico é observado em praticamente qualquer sistema físico. Por exemplo, considere-se:

Uma chaleira com água em um fogão convencional, sob ação de uma fonte de calor, a chama, a água presente dentro da chaleira não saltará imediatamente da temperatura ambiente para a temperatura final da chama; pelo contrário, a água do recipiente mostra um aquecimento gradual. O mesmo ocorrerá, mas de maneira inversa, quando a chama for apagada: a temperatura da água não sofrerá uma queda brusca imediatamente mas, lentamente diminuirá até a temperatura ambiente.

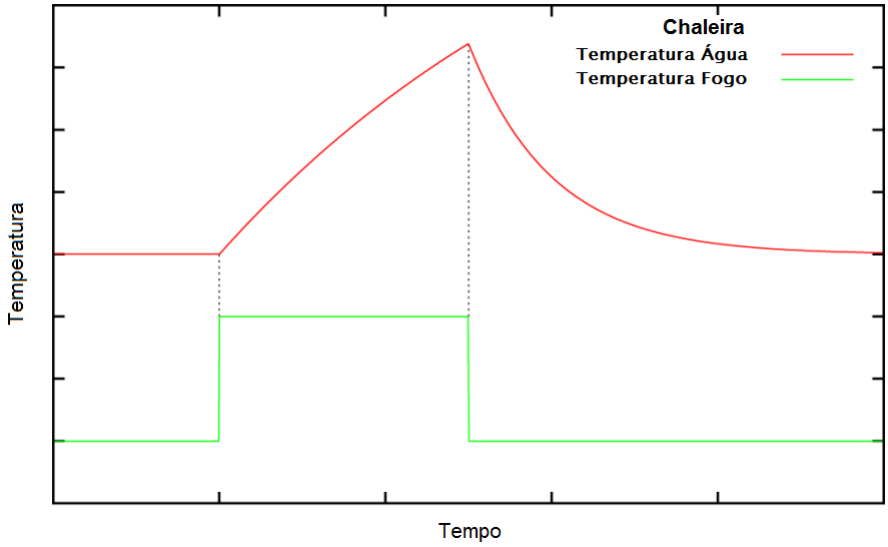


Figura 3 – Dinâmica do aquecimento da água

Fonte: Adaptada de Janert (2013).

Neste caso da chaleira em aquecimento, a constante temperatura da chama (enquanto o fogo ligado, fornecendo calor) representa a entrada do sistema, como indicado pela linha verde da Figura 3, e o aquecimento da água representa o comportamento do sistema mediante a entrada, conforme demonstrado pela linha vermelha da mesma Figura 3.

Considere-se, como outro exemplo, uma esfera maciça, em repouso, suspensa por uma mola, também em repouso, presa a certa altura. Ao deslocarmos o peso tirando-o do repouso e soltarmos após o tensionamento da mola, a massa iniciará uma oscilação amortecida até o ponto em que a massa e a mola voltaram ao equilíbrio inicial.

O deslocamento inicial (entrada do sistema), por nós provocados, é representado pela linha verde na Figura 4 e a oscilação subsequente em resposta à força aplicada é representado pela linha vermelha na mesma Figura 4.

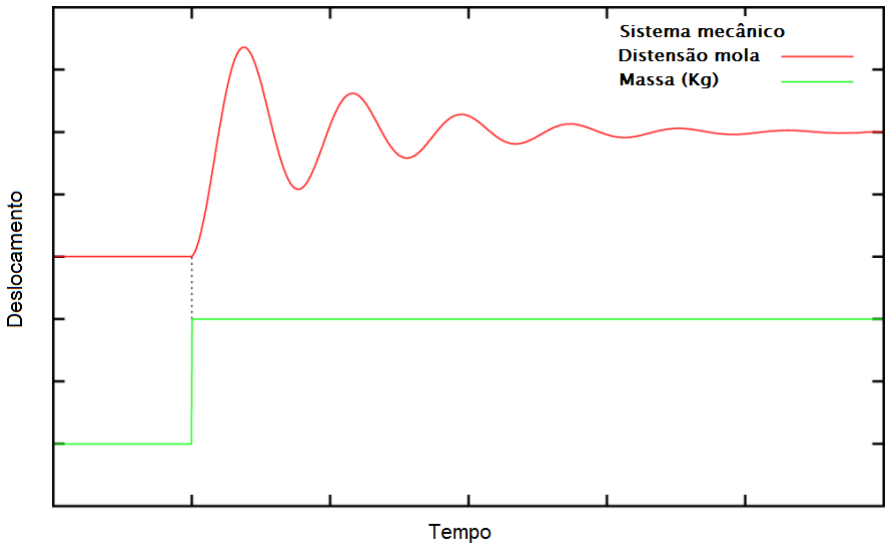


Figura 4 – Dinâmica da distensão de uma mola

Fonte: Adaptada de Janert (2013).

Os exemplos de sistemas dinâmicos físicos e mecânicos, apresentados anteriormente, e as suas respostas aos estímulos externos, são comportamento encontrados em sistemas computacionais e estes aspectos são negligenciados em muitos sistemas, pois até então os sistemas computacionais tem-se comportado com uma dinâmica imperceptível aos usuários. Entretanto, os atuais sistemas computacionais tem tomado tamanha proporção que já não tem respondido imediatamente a um estimulo, produzindo oscilações na saída, apresentando, assim, características de dinâmica (JANERT, 2013). Vale salientar e enfatizar, que estes comportamentos dinâmicos estão apreciáveis devido a variação na entrada dos sistema em avaliação.

A Figura 5 é um exemplo onde pode-se apreciar a dinâmica em sistemas computacionais. O gráfico **??**, representa a entrada do sistema, número de requisições por amostragem de tempo. No instante de 1000 amostragens de tempo, o número de requisições é dobrado, de 7 para 14 requisições por amostragem de tempo. Este abrupta aumento da entrada proporciona a nitidez da dinâmica do sistema na saída conforme apresentado pelo gráfico **??**, referente a quantidade de máquinas virtuais. Próximo a 1100 amostragens de tempo, o número de máquinas virtuais chega próximo a 24 máquinas virtuais, após algumas amostragens de tempo, o número de máquinas virtuais diminui para 21 se mantendo até o final.

Embora a análise transiente ainda seja pouco explorada frente as principais abordagens de avaliação de desempenho, Júnior *et al.* (2015b) discutem sobre o tema e sua importância para sistemas computacionais. A abordagem predominante na avaliação de desempenho dos sistemas computacionais considera a análise da capacidade no estado estacionário. Geralmente deseja-se descartar efeitos de inicialização e, em seguida, medir o desempenho quando a saída se estabilizou. A razão para a proeminência da análise estacionária é devido a não importância da dinâmica em sistemas computacionais tradicionais. Normalmente, as plataformas de compu-

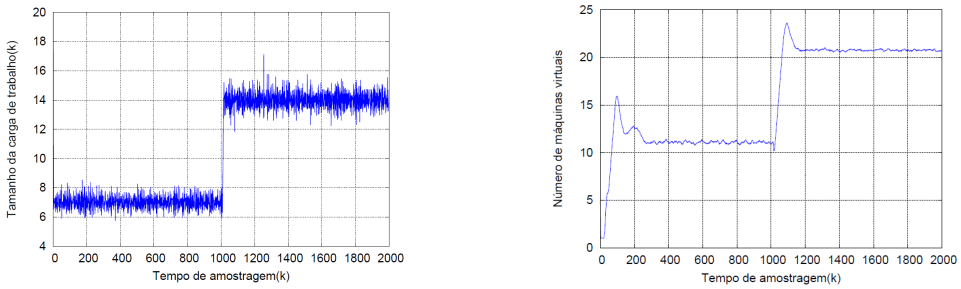


Figura 5 – Dinâmica de controlador (LUZ, 2014)

tação respondem suficientemente rápido as alterações operacionais que afetam o desempenho. Essas justificativas, no entanto, não são necessariamente verdadeiras para sistemas distribuídos emergentes, especialmente em ambientes de grande escala e aplicações de *mult-tiers*. O efeito de pequenos atrasos e a propagação para outras camadas podem render comportamento dinâmico importantes (JúNIOR *et al.*, 2015b).

Durante o regime transitório, o desempenho pode apresentar comportamentos diferentes, dependendo da dinâmica do sistema: ele pode variar abruptamente, ou lenta e progressivamente, e até mesmo apresentar componentes oscilatórios. Análise em regime transitório pode revelar capacidade dinâmica do sistema dependendo das suas propriedades inerciais (JúNIOR *et al.*, 2015b).

Segundo Janert (2013), a resposta de um sistema a uma perturbação externa muitas vezes consiste em um componente transiente, que desaparece ao longo do tempo, e um componente de estado estacionário. Os componentes transiente diluem no decorrer do tempo.

O trabalho de Júnior *et al.* (2015b) discute sobre a abordagem de análise de estado estacionário vigente para avaliação de desempenho de sistemas computacionais e apresenta uma abordagem para o planejamento de experimentos de simulação destinados a análise transitória, especialmente em aplicações *multi-tiers*. Júnior *et al.* (2015b) iniciam discutindo sobre as propriedades dinâmicas de sistemas computacionais de grande escala em ambientes distribuídos e como essas características podem afetar o desempenho do sistema. As dinâmicas emergentes da interligação de sistemas computacionais de *multi-tiers* e escaláveis podem produzir efeitos transitórios sobre o desempenho, como resposta temporal e comportamento oscilatório, sendo que estes efeitos variáveis no tempo podem afetar a capacidade de resposta, eficiência e até mesmo a estabilidade do mesmo.

Júnior *et al.* (2015b) apresentam um conjunto de requisito que especificam uma arquitetura conceitual, intitulada MEDC (*Monitor, Effector, Demanda and Capacity*) representado na Figura 6. O diagrama ilustra a arquitetura que tem quatro preocupações essenciais: *Demand*, *Capacity*, *Monitor* and *Effector*:

***Demand*(Demanda):** é uma referência para a capacidade de modulação da entrada e deve ser configurada de modo a especificar a forma como a carga de trabalho muda ao longo do

tempo;

***Capacity*(Capacidade):** é a disposição análoga no que diz respeito aos recursos do sistema;

***Monitor*(Monitor):** desempenha o papel de um sensor e aquisição de dados temporais, sendo seu dever de coletar os dados sobre o sistema e torná-lo disponível;

***Effector*(Efetor):** representam os mecanismos de atuação através modulação tem efeito, que serve como uma camada de abstração disponível para a modelagem dos componentes operacionais que afetam a dinâmica *Capacity* e *Demand*.

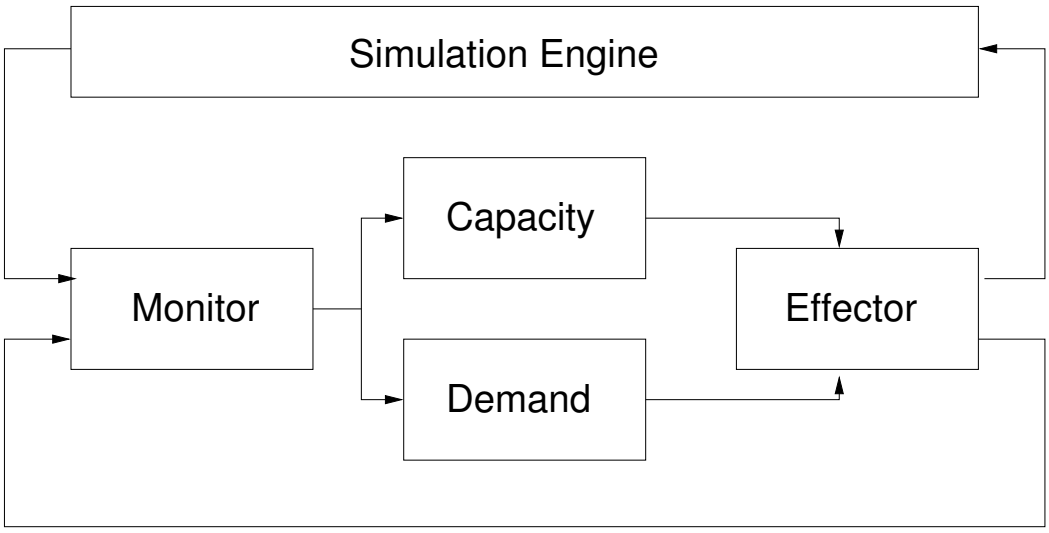


Figura 6 – Arquitetura conceitual MEDC

Fonte: Júnior *et al.* (2015b).

O fluxo de trabalho está associado a responsabilidade de controlar a geração de carga com os ciclos de simulação e, correspondentemente, no que diz respeito à capacidade. Este controle se traduz em uma política de tomada de decisão que aciona eventos que solicitam a criação ou o liberação de novas entidades de carga de trabalho e de recursos. A ordem é efetivamente realizada pelo *Effector*, que recebe as solicitações dos controladores *Demand* e *Capacity* os manipulam antes de transmitir os eventos para o mecanismo de núcleo. É através do *Effector* que pode-se modelar, por exemplo, atrasos associados a alocação e deslocação, de políticas de escalonamento, enchimento do *buffers*, os tempos de resposta do *hardware*, padrões de falha de recurso, etc. O dever do *Monitor* está em alimentar os módulos *Demand* e *Capacity* de dados, por exemplo, utilização média do sistema, tempo de permanência trabalho, relação taxa de perda de *deadlines*, etc. Para cada uma das quatro responsabilidades existe uma ação associada de forma iterativa em cada etapa do ciclo de simulação (JúNIOR *et al.*, 2015b).

A fim de expor essas propriedades dinâmicas, o experimento de simulação deve excitar o sistema com carga de trabalho não-estacionária sob condições controladas. Para que seja possível apreciar a dinâmica de um sistema e realizar a análise transitória do sistema é preciso utilizar

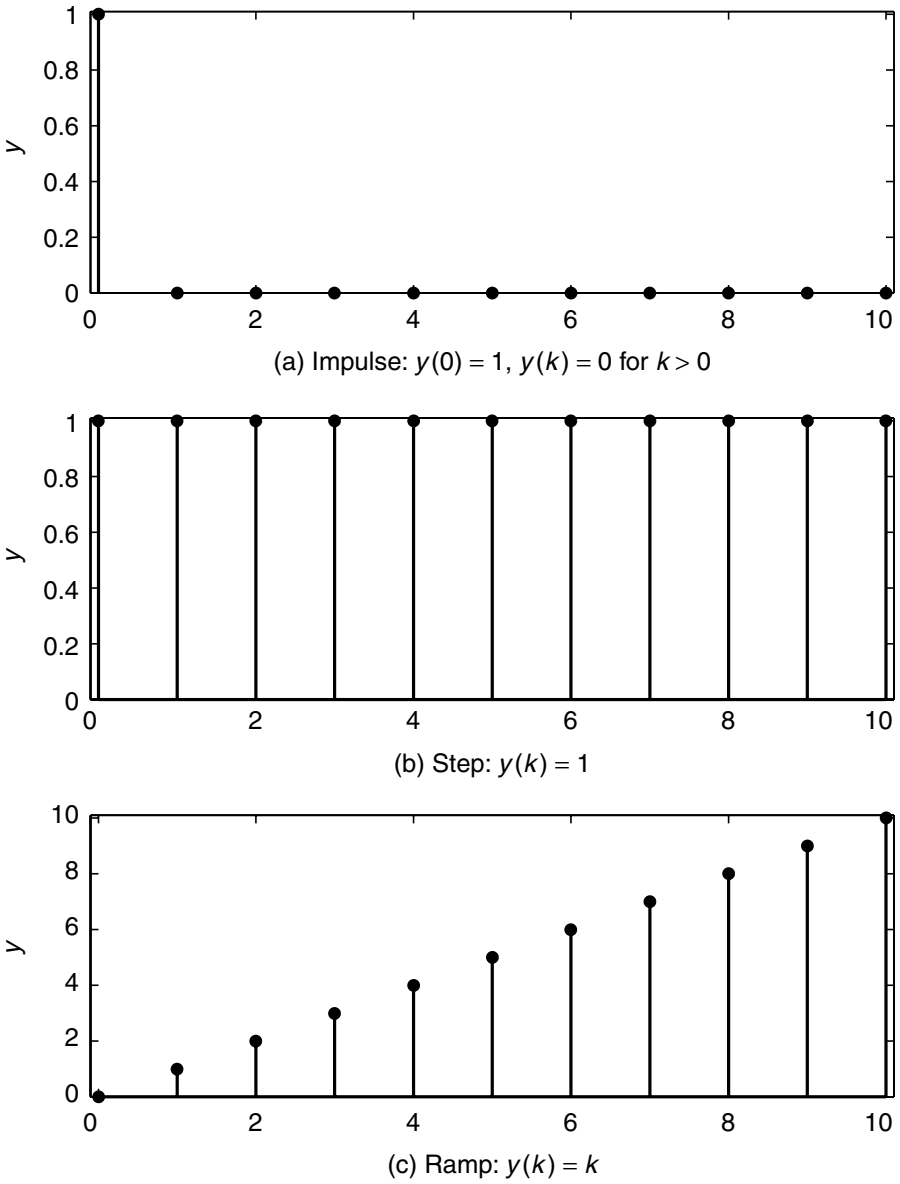


Figura 7 – Sinais em tempo discreto comuns, parte 1

Fonte: Hellerstein *et al.* (2004).

cargas de trabalho que possam provocar comportamentos propícios, onde as caraterísticas dinâmicas do sistema sejam claramente observadas Hellerstein *et al.* (2004), apresentam propostas de algumas funções, ou sinais, de perturbação, capazes de excitar o sistema a apresentar a sua dinâmica por exemplo: impulso, degrau, rampa, seno, exponencial, seno modulada por uma exponencial, etc. Essas funções são apresentadas nas Figuras 7 e 8. Cada uma das funções tem características distintas, exceto pelo fato de todas alterarem no decorrer do tempo, e esta característica é crucial e fundamental para este trabalho. Para este trabalho utilizaremos somente a segunda função (*Step*) da Figura 7, esta é suficiente para impactar o sistema afim de apresentar a sua dinâmica.

Mamani (2015) pretende utilizar a especificação de arquitetura proposta por Júnior *et*

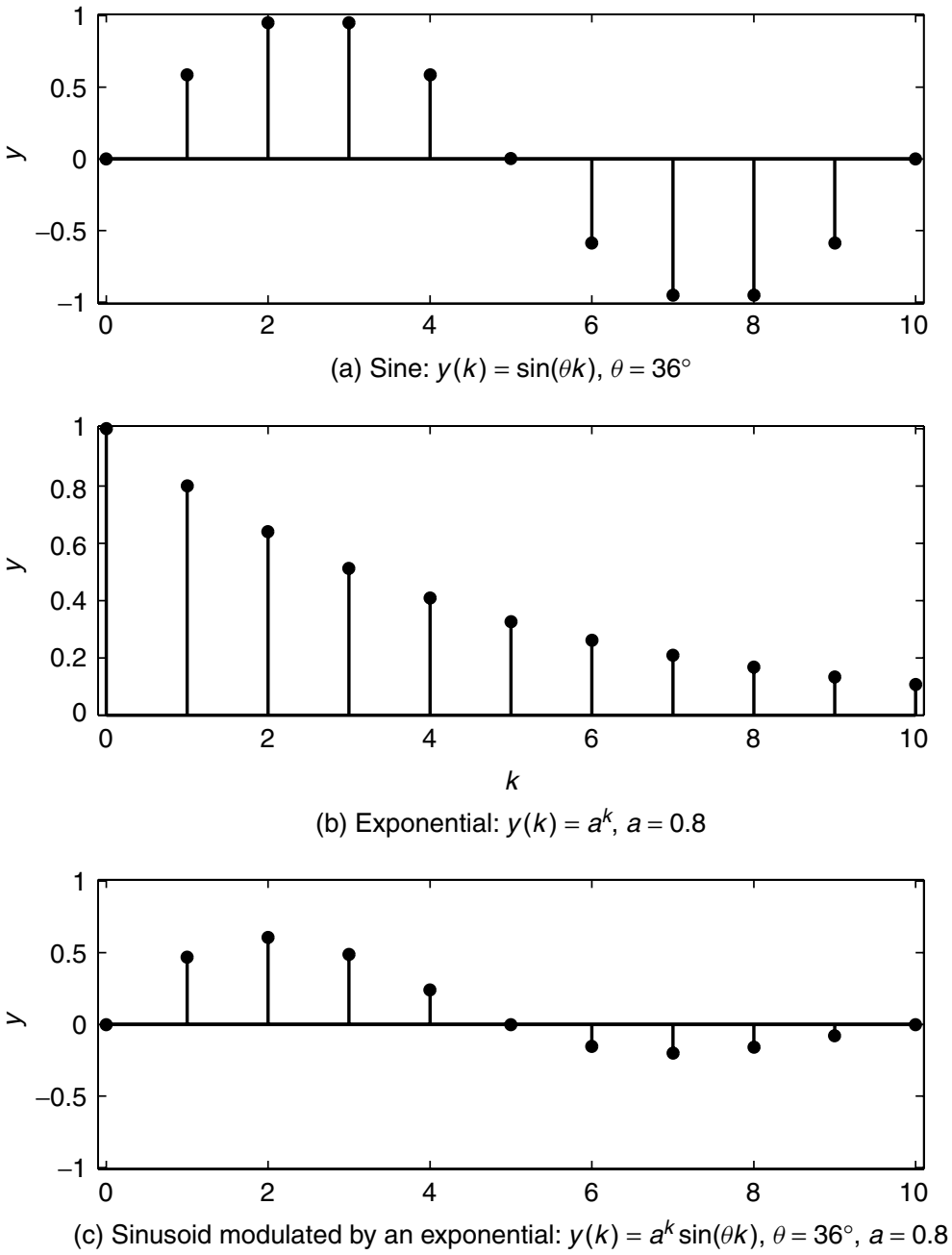


Figura 8 – Sinais em tempo discreto comuns, parte 2

Fonte: Hellerstein *et al.* (2004).

al. (2015b) em um ambiente real. Estes trabalho irá atuar diretamente no requisito *Demand* da arquitetura conceitual MEDC, implementado no *benchmark* Bench4Q.

A definição de um *benchmark* é tema de inúmeras pesquisas, uma série de trabalhos que tentam fornecer orientações e critérios de qualidade que devem ser considerados no projeto e execução de *benchmark*, (KISTOWSKI *et al.*, 2015; CHEN; RAAB; KATZ, 2014; FOLKERTS *et al.*, 2013; VIEIRA *et al.*, 2012; HUPPLER, 2009; GRAY, 1992) :

Relevância é, talvez, adequada a característica mais importante de um *benchmark*. Mesmo que a carga de trabalho seja perfeita em todos os outros aspectos, será de uso mínimo se ele não fornece informações relevantes para seus usuários. No entanto, a relevância é também uma característica de como os resultados do *benchmark* são aplicados; *benchmarks* podem ser altamente relevantes para alguns cenários e ter mínima relevância para outros. Para o usuário do *benchmark*, uma avaliação da relevância de um ponto de referência deve ser feita no contexto da utilização prevista desses resultados para o *benchmark* (KISTOWSKI *et al.*, 2015).

Reprodutibilidade é a capacidade de produzir os mesmos resultados de forma consistente para um ambiente de teste em particular. A capacidade de reproduzir os resultados em outro ambiente de teste é em grande parte ligada à capacidade de construir um ambiente equivalente. *Benchmarks* de indústria requere, além de resultados, uma descrição do ambiente de teste, geralmente incluindo hardware e componentes de software, bem como opções de configuração. *Hardware* deve ser descrito em suficiente em detalhe para uma outra pessoa possa obter um arranjo equivalente. As versões de *software* devem ser indicados de modo que seja possível usar as mesmas ao reproduzir o resultado. Opções de ajuste e configuração devem ser documentadas para versão do *firmware*, sistema operacional e aplicativos para que as mesmas opções possam ser usadas. (KISTOWSKI *et al.*, 2015)

Verificabilidade Dentro da indústria, *benchmarks* são normalmente executados por fornecedores que têm interesse nos resultados. Na academia, os resultados são submetidos a revisão por pares e resultados interessantes serão repetidos e desenvolvidos por outros pesquisadores. Em ambos os casos, é importante que os resultados do *benchmark* sejam verificáveis, de modo que os resultados possam ser considerados dignos de confiança (KISTOWSKI *et al.*, 2015).

Usabilidade A maioria dos usuários de *benchmarks* são normalmente técnicos, tornando a facilidade de uso uma preocupação menor do que é para aplicações pensadas e desenvolvidas para o consumidor. Existem, no entanto, várias razões pelos quais a facilidade de utilização é importante. Um aspecto da facilidade de utilização é ser capaz de construir configurações práticas para a execução do *benchmark*.

Escalabilidade deve ser apoiada em uma maneira que preserve a relação com o cenário de negócios próximo ao modelo real. Além disso, ao usuário deve ser oferecido a possibilidade de dimensionar a carga de trabalho de forma arbitrária pela definição de um conjunto próprio de pontos de escala (VIEIRA *et al.*, 2012).

Simplicidade Os elementos conceituais de um *benchmark* devem ser reduzidos ao mínimo e feitos de fácil compreensão. O *benchmark* também deve abstrair detalhes que representam configurações de caso a caso, ou escolhas de administração do sistema que não afetam o

desempenho (CHEN; RAAB; KATZ, 2014). Um *benchmark* com uma estrutura altamente complexa é muitas vezes difícil de entender e difícil confiar. Se as pessoas não confiam no *benchmark*, elas não irão usá-lo. *Benchmarks* devem, portanto, ser o mais simples possíveis. Complexidade necessária pode ser explicada em documentação auxiliar (WEBER, 2014).

Economia É muitas vezes negligenciada durante o desenvolvimento inicial do *benchmark*, uma vez que as fases iniciais do desenvolvimento estão focadas em imitar a realidade para fornecer a relevância necessária ao *benchmark*. Para ser relevante, um *benchmark* deve ser realístico; e ser realístico, muitas vezes significa ser complexo; e complexo pode ser caro. Esta é claramente uma outra oportunidade para o compromisso, para *benchmark* de sucesso. Por exemplo, os resultados da IBM, *On-line Transaction Processing* (TPC-C) ou *Complex on-line transaction processing* (TPC-E) ou *Ad-hoc decision support system* (TPC-H) e alguns da *Standard Performance Evaluation Corporation* (SPEC), SPECjAppServer2004 e SPECweb2005, todos eles anunciam característica como baixo custo para implementar, o apelo de ser barato para correr, fácil de executar e fácil de verificar. Enquanto esses não são usados fora do contexto da sua intenção, eles também atendem aos requisitos para a relevância, equidade e reprodutibilidade (HUPPLER, 2009).

Métrica Uma métrica ser claramente compreensível. Segundo Folkerts *et al.* (2013) as métricas do *benchmark* devem permitir caracterizar e quantificar o comportamento do sistema quando enfrenta perturbações (ou seja, falhas, ataques, e variações de ambiente operacional). Métricas de *benchmark* podem caracterizar o desempenho, confiabilidade e segurança (VIEIRA *et al.*, 2012).

O *benchmark* utilizado neste trabalho e objeto alvo da extensão proposta é o Bench4Q, que simula um *e-commerce* orientado a QoS e oferece recursos para deduzir uma representação controlável e flexível da carga de trabalho baseada em sessões complexas, e para reproduzem simular o comportamento do cliente (ZHANG *et al.*, 2011a).

O trabalho Cherkasova e Phaal (1998), apresentam o conceito de sessão, que define uma sequência de requisições de um único cliente. Krishnamurthy, Rolia e Majumdar (2006), apresentam a dependência de sessões em sistemas *e-commerce* e ressalta a importância de caracterizar a carga de trabalho sintética.

A maioria dos *benchamarks* de *e-commerce*, incluindo o TPC-W, são limitados para os sistemas sensíveis a QoS. As métricas utilizadas são baseadas em requisições *Hypertext Transfer Protocol* (HTTP), como a taxa de requisições atendidas com sucesso ou o tempo de resposta das requisições. Embora uma das características mais críticas de um sistema de *e-commerce* sensíveis a QoS seja a integridade do serviço prestado aos clientes, as métricas baseada em requisições podem levar a afirmações ineficientes e até mesmo erradas.

O Bench4Q é uma extensão do TPC-W (MENASCE, 2002), e tem como objetivo o *tuning* de servidores *e-commerce* orientados a fornecer QoS aos seus clientes. As principais

características do Bench4Q incluem: apoio à análise de métricas baseada em sessão que simula carga sensível a QoS para uma análise da capacidade.

O *benchmark* Bench4Q é distribuído de acordo com a *Lesser General Public License* (GNU), sendo um software livre. Seguindo muitas diretrizes da especificação do TPC-W, o Bench4Q usa principalmente em suas métricas de simulação a garantia de QoS (ZHANG *et al.*, 2011a).

O Bench4Q oferece uma arquitetura distribuída para a geração de carga através de seus agentes que são conectados a um único console que os gerencia, por meio do qual é possível ajustar separadamente as configurações para cada agente. Esses agentes geram carga (requisições HTTP) para o servidor de aplicação onde está hospedado o *e-commerce*, como ilustrado na Figura 9. Os resultados da avaliação de carga aplicada ao *e-commerce* são coletados pelo Console, que apresenta alguns gráficos, os quais facilitam a interpretação da avaliação mediante as diretrizes do TPC-W (ZHANG *et al.*, 2011a).

A ferramenta é composta por três partes: Console, Agente e SUT, conforme apresenta a Figura 9, e também disponibiliza interfaces para o monitoramento de recursos para o servidor de aplicação e para o banco de dados; este monitoramento inclue CPU, memória, rede, etc.

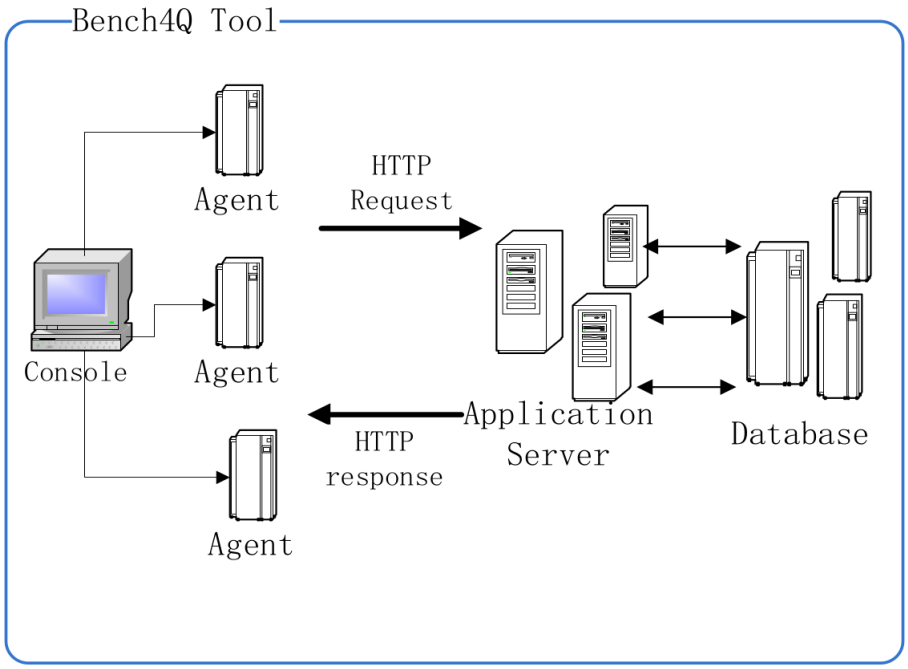


Figura 9 – Arquitetura Bench4Q

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

- **Console:** A Figura 10 apresenta a interface do console, onde configura-se o teste, coleta e exibição os resultados.

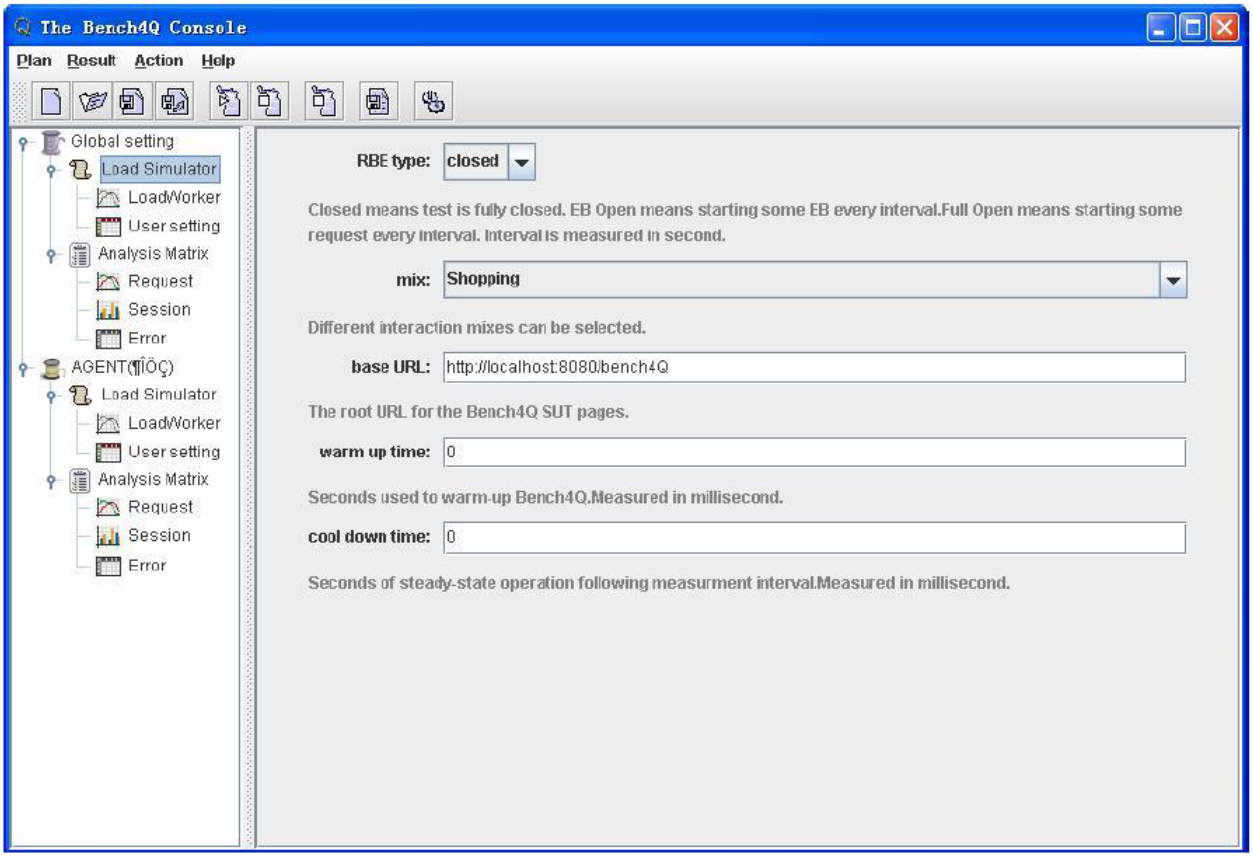


Figura 10 – Console Bench4Q

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

- **Agente:** É este que faz o trabalho real, pois gera a carga configurada no console. Simula o comportamento dos usuários no *website*, disparando diversas requisições para o *e-commerce*.

- **SUT:** O *e-commerce* SUT, conforme apresentado na Figura 11, é que fornece o *website* de compras e está organizado com um banco de dados, servidor web e servidor de aplicativos. O portal compreende todos os componentes que fazem parte de uma aplicação real. Isso inclui as conexões de rede, servidores web, servidores de aplicação, servidores de banco de dados, etc.

De uma maneira mais especifica, percebe-se que os *benchmarks* para aplicações computacionais típicas de Computação em Nuvem, como *e-commerce*, por exemplo, não são atualmente orientados a suportar totalmente QoS (ZHANG *et al.*, 2011b). Um exemplo desses recursos é a integralidade do serviço, que é geralmente expressa como uma sessão fornecida aos clientes. Nesse sentido o Bench4Q tenta abranger essa lacuna.

O TPC-W, estendido pelo Bench4Q, é (mesmo atualmente tido como obsoleto) um *benchmark* direcionado a *website* de *e-commerce* transacionais em que os consumidores são vinculados a sessões. Cada cliente possui sua própria e única sessão em determinado intervalo

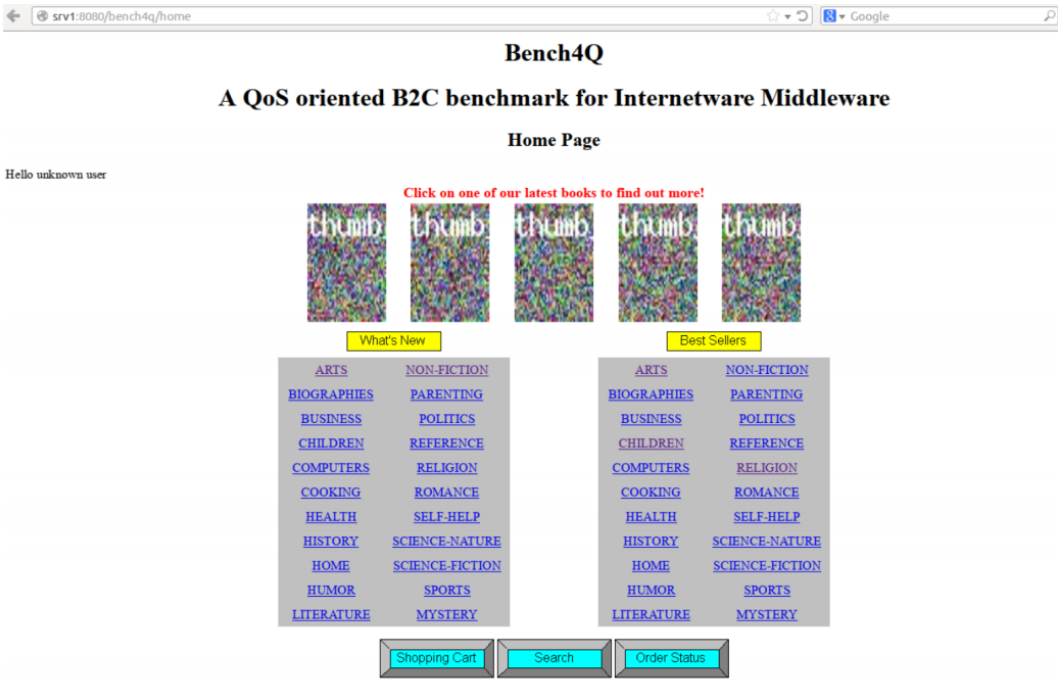


Figura 11 – SUT Bench4Q

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

de tempo. Trata-se de um *e-commerce* de venda de livros implementado em Java e hospedado em um servidor de aplicação Tomcat [1]. Clientes, intitulados pela ferramenta como *Emulations Browsers* (EBs), são os que emulam e comportam de consumidores dos produtos oferecidos pelo *e-commerce*. Basicamente, os clientes podem interagir de diferentes formas: acessar a página inicial do *site* (home), navegar e procurar por produtos, realizar operações que envolvam o carrinho de compras e finalizar uma compra. A carga de trabalho gerada pelo TPC-W pode ser representada por um *Customer Behavior Model Graph* (CBMG), um grafo orientado, em que os nós representam uma operação a ser realizada (procurar, navegar, comprar etc.) e os pesos nas arestas significam a probabilidade de transição de uma operação para outra (ZHANG *et al.*, 2011b). A Figura 12 modela o fluxo de requisições a páginas e operações que um cliente pode realizar em uma sessão, basicamente um comportamento estocástico de acesso às páginas.

A carga de trabalho gerada pelo TPC-W é formada por um CBMG com 14 tipos de interação *web* e três perfis de pesos para suas arestas. O resultado é uma carga em que a maioria dos clientes apenas navega nas páginas (navegação: 95% e compra: 5%); outra em que a maioria dos clientes realiza compras de forma moderada (navegação: 80% e compra: 20%); e a terceira composta por muitos clientes que finalizam as compras (navegação: 50% e compra: 50%). Vale ressaltar que existe um atraso entre as requisições de uma sessão. Ao iniciar uma sessão uma requisição é disparada e após receber a respectiva resposta, a próxima requisição acontece após um tempo também estocásticos e variante. Essa especificação tem o objetivo de emular melhor o

[1] Tomcat: <http://tomcat.apache.org/>

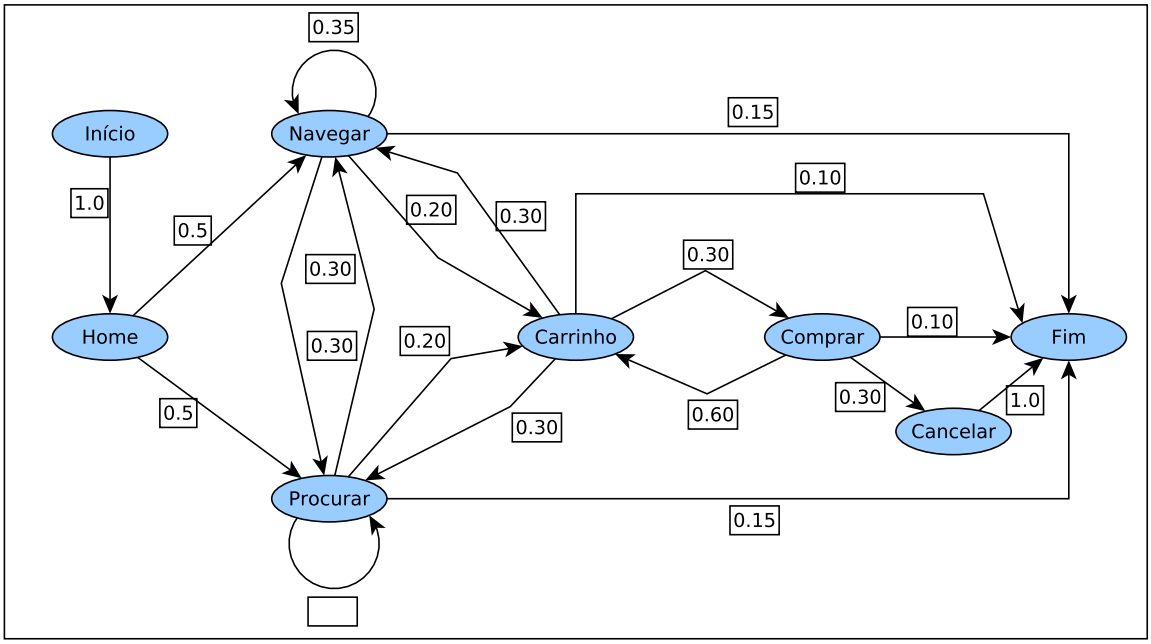


Figura 12 – CBMG - perfil de navegação dos *brwosers* do Bench4Q

Fonte: Adaptada de Mark e Csaba (2007).

comportamento humano ao acessar *website* de *e-commerce*.

Ao ser implantado em uma infraestrutura computacional que hospede todos seus componentes, os clientes emulados devem estar localizados fora de tal infraestrutura e conectados por uma rede. Esses clientes começam as séries de requisições. O TPC-W implementa um modo de geração de carga conhecido como fechado (*close mode*): um novo cliente chega somente após o antigo deixar o sistema (ZHANG *et al.*, 2011b). As métricas disponíveis concernem basicamente à quantidade de interações *web*, medida em interações por segundo (*Web Interaction Per Second* (WIPS)) e seu tempo de resposta (*Web Interaction Response Time* (WIRT)).

A motivação para a extensão do TPC-W e criação do Bench4Q começa porque, embora as referidas métricas aparentem descrever bem a quantidade de acessos ao sistema, a qualidade do serviço experimentada pelo usuário pode ser desproporcional a essas mesmas métricas. Em Ow2 e Trustie (2010) é descrito um ensaio que contempla dois cenários diferentes: um normal e outro otimizado não realisticamente. A otimização foi feita pela configuração de parâmetros no servidor Tomcat, a saber: `sessionTimeout`, `connectionTimeout` e `acceptCount`. Um das características da aplicação é a presença de operações *IO-bound*, por consequência, observar o tempo médio de resposta dessas operações permite diminuir os valores de estouro de tempo, e, com isso, forçar uma taxa de utilização de CPU. Assim, pode-se *"otimizar"* o ambiente da aplicação. Os resultados foram de $WIPS = 131$ para o cenário normal e $WIPS = 199$ para aquele otimizado não realisticamente, colaborando a hipótese. Porém, ao observa-se a quantidade de sessões completadas com sucesso, conforme apresentado na Figura 13, percebe-se que o cenário

normal permitiu uma quantidade de erros menor do que aquele otimizado não realisticamente. Por consequência, uma expectativa de lucro maior quando da implantação: maior quantidade de sessões realizadas sem erro implicaria em maior probabilidade de compras efetivas.

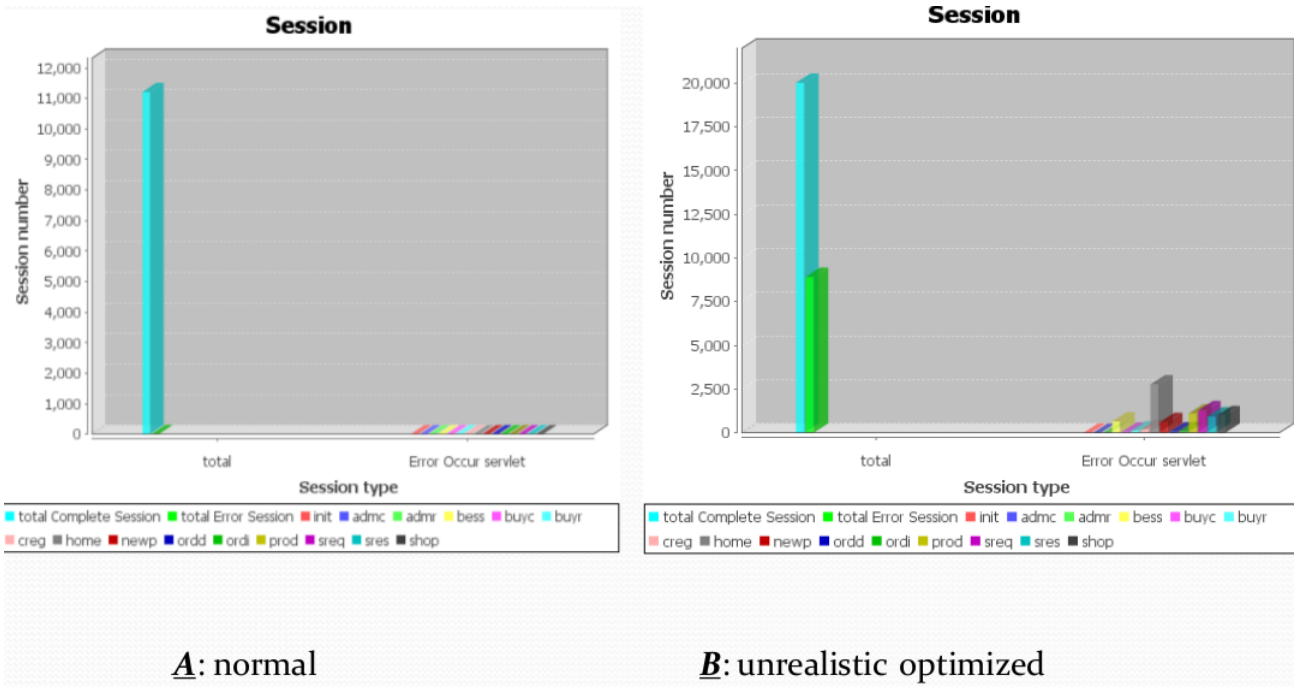


Figura 13 – Comparação da quantidade de requisições completadas com sucesso entre dois cenários: normal e otimizado não realisticamente.

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

Tanto o TPC-W quando Bench4Q tem como objetivo produzir resultados que possibilitem o *tuning* (sintonia do parâmetros) de servidores que compõem a arquitetura de um serviço *e-commerce* orientados a fornecer QoS aos seus clientes (MENASCE, 2002; ZHANG *et al.*, 2011b). A ideia principal é que seja construída uma banca de testes (*testbed*), haja a execução do *benchmark* e consecutiva geração dos traços de execução, seja feita a extração dos dados e, por fim, com base nos resultados, melhores valores para os parâmetros de configuração do sistema sejam aplicados. Especificamente sobre o Bench4Q, as principais funcionalidades implementadas nele tem como contexto a observância do estado das sessões.

O *e-commerce* é um modelo de negócio bastante comum e possível principalmente por sua implantação em ambiente *web*, orientado a negociação de bens e serviços. Igualmente como o que acontece na forma tradicional de negociação, disputas por clientes e/ou fatias de mercado surgem naturalmente, como podem ser observadas em promoções, ofertas, lançamentos de novos produtos etc. Portanto, um *website* alinhado a esses requisitos que proporcionem um desempenho melhor, ou seja, que possibilidade uma experiência melhor a seus usuários, certamente já possui uma vantagem competitiva. Nesse sentido, o desempenho da infraestrutura que hospedará o negócio e o ajuste fino dos parâmetros operacionais da solução *e-commerce* podem ser encarados como requisitos não-funcionais. Assim, seria possível realizar avaliações de desempenho que

fomentem o projeto de tais sistemas cujo resultado sejam sistemas mais eficientes, mesmo que sejam abstraídas algumas regras específicas do negócio a ser implementado. O Bench4Q é uma ferramenta que possibilita tais projetos.

A oscilação da carga de trabalho é uma característica fundamental. A simultaneidade dos acessos apresentam grandes efeitos sobre a escolha da política de ajuste de um servidor. O Bench4Q simula tal oscilação de carga através dos seus agentes com os seguintes parâmetros:

- *Base Load:* quantia fixa de *threads* por agentes.
- *Radom Load:* quantidade de *threads* que são geradas aleatoriamente.
- *Rate:* taxa de mudanças de carga; pode ser positivo, negativo ou nulo. Positivo significa que a carga de base está aumentando a cada segundo, se negativa significa que a carga de base está diminuindo a cada segundo e enquanto a taxa é zero, a carga é fixa.
- *Trigger Time:* o tempo para o agente de carga para começar a gerar sua carga.
- *Duration:* o tempo de execução do agente de carga.

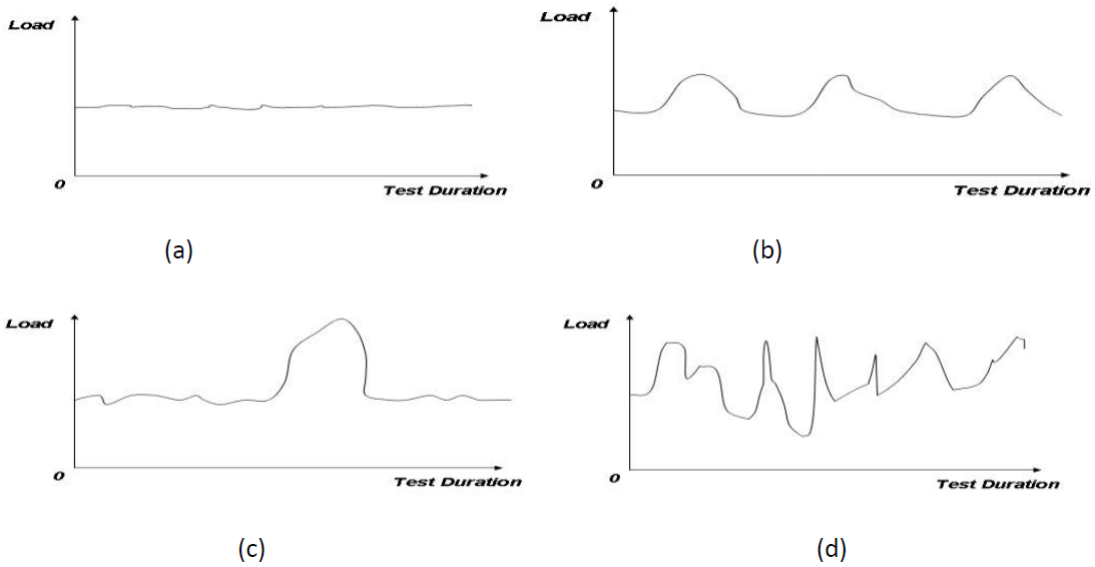
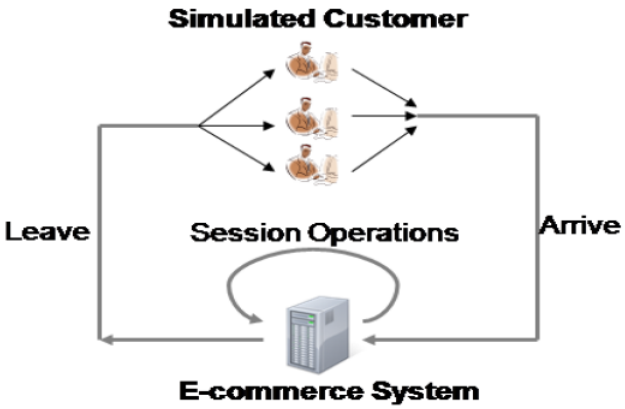


Figura 14 – Carga de trabalho gerada pelo Bench4Q

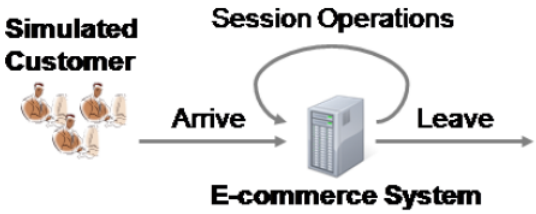
Fonte: Zhang *et al.* (2011a).

Existem dois modos de simulação de carga: o fechado e o aberto, conforme o serviço na Figura 15. No modo fechado, ilustrado na Figura 15 (a), um novo cliente só acessará depois do cliente antigo deixar o sistema. Já no modo aberto, ilustrado na Figura 15 (b), novos clientes vão acessar o sistema sem se importar com a saída dos antigos clientes. O TPC- W simula carga no modo fechado, o que faz uma suposição inexistente do mundo real (ZHANG *et al.*, 2011a).

Em sua utilização neste projeto, a sessão aberta é a que mais se adequa à finalidade do objetivo proposto.



(a) Closed Mode



(b) Open Mode

Figura 15 – Tipo de sessões Bench4Q

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

CAPÍTULO

3

METODOLOGIA

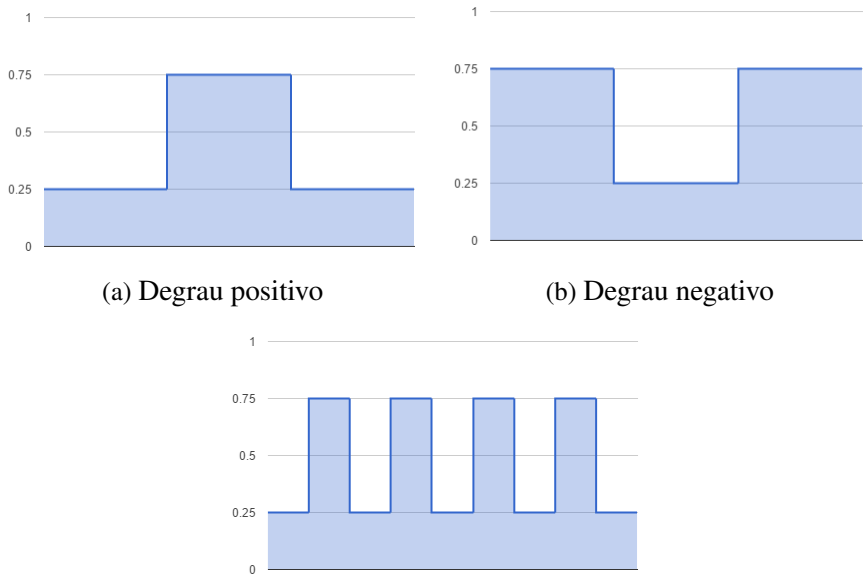
O objetivo deste trabalho é estender a carga de trabalho do *benchmark* Bench4Q para que seja possível estimular o sistema a apresentar a sua dinâmica, e assim possibilitando a analise transiente do sistema, incorporando-se o modulo *Demand* da arquitetura conceitual MEDC.

Conforme apresentado no Capitulo 2, o *benchmark* Bench4Q oferece uma interface gráfica para configuração e coleta dos dados, o que facilita a sua operação e análise. A proposta da extensão deste trabalho é manter o padrão de usabilidade e possibilitar a modulação da carga de trabalho atrás de uma interface gráfica. Sendo assim, com o preenchimento de um conjunto de parâmetros será possível a geração modulada da carga:

- **Tempo de planejamento de carga:** Um período de tempo em que a carga de trabalho é modulada, caracterizando a mudança do comportamento das requisições de maneira programada;
- **Tipo de modulação:** conforme apresentado no Capitulo 2, a modulação será apresentada conforme as funções ou sinais propostos por Hellerstein *et al.* (2004);
- **Tempo de interrupções:** Período de interrupções/pausa após o *Tempo de planejamento de carga*;
- **Quantidade de clientes na modulação:** reservar uma quantidade de clientes EBs, que estão com dedicação exclusiva para a modulação da carga.

Através da nova interface, que recebe os parâmetros para a modulação da carga, espera-se modular a carga conforme os exemplos apresentados na Figura 16. Esses exemplos são derivados das funções apresentadas por Hellerstein *et al.* (2004). Com esses tipos de variações na carga, no decorrer do tempo, é possível estimular o sistema de maneira a expor a sua dinâmica. O objetivo é ser capaz de criar cargas de trabalho que possam ser utilizadas em estudos de avaliação de desempenho não estacionária. A carga de trabalho em formato de um pulso conforme a Figura

16a é de interesse para o trabalho de Mamani (2015). Este formato de carga se inicia estacionária e tem um crescimento e brusco repentinamente, mantendo-se por um período, antes de retomar ao patamar inicial. Já carga representada pela Figura 16b tem comportamento oposto a carga anterior da Figura 16a. Por fim, a carga modulada conforme a Figura 16c, é a combinação de ambos os formatos de carga apresentados anteriormente, oscilando entre níveis de baixa intensidade à máxima intensidade, sempre com alterações bruscas e repentinas, nos mesmos intervalos de unidade de tempo.



(a) Degrau positivo

(b) Degrau negativo

(c) Onda quadrada

Figura 16 – Possibilidade de cargas moduláveis pela extensão

O Bench4Q simula um *website* de um *e-commerce* de livros, utilizando uma arquitetura *multi-tier*, particionando o processo da aplicação em níveis, onde cada camada fornece uma determinada funcionalidade. Uma vantagem de tal arquitetura é a possibilidade de proporcionar um elevado nível de escalabilidade e a flexibilidade. No entanto, a alocação de recursos entre esses níveis será mais difícil devido à interdependência entre as camadas. Para a discussão deste trabalho, assumimos um sistema de *multi-tiers* que consiste nos seguintes componentes:

- Gerador de Carga (*Workload*)
- Balanceador de carga (*Load Balancer*)
- Servidor Físico (*Hypervisor*)
- Servidor de dados (*Data base*)

A descrição e especificação da arquitetura anterior foi implementada em ambiente real e utilizada para a execução dos experimentos definidos e descritos a seguir. Foram executados das

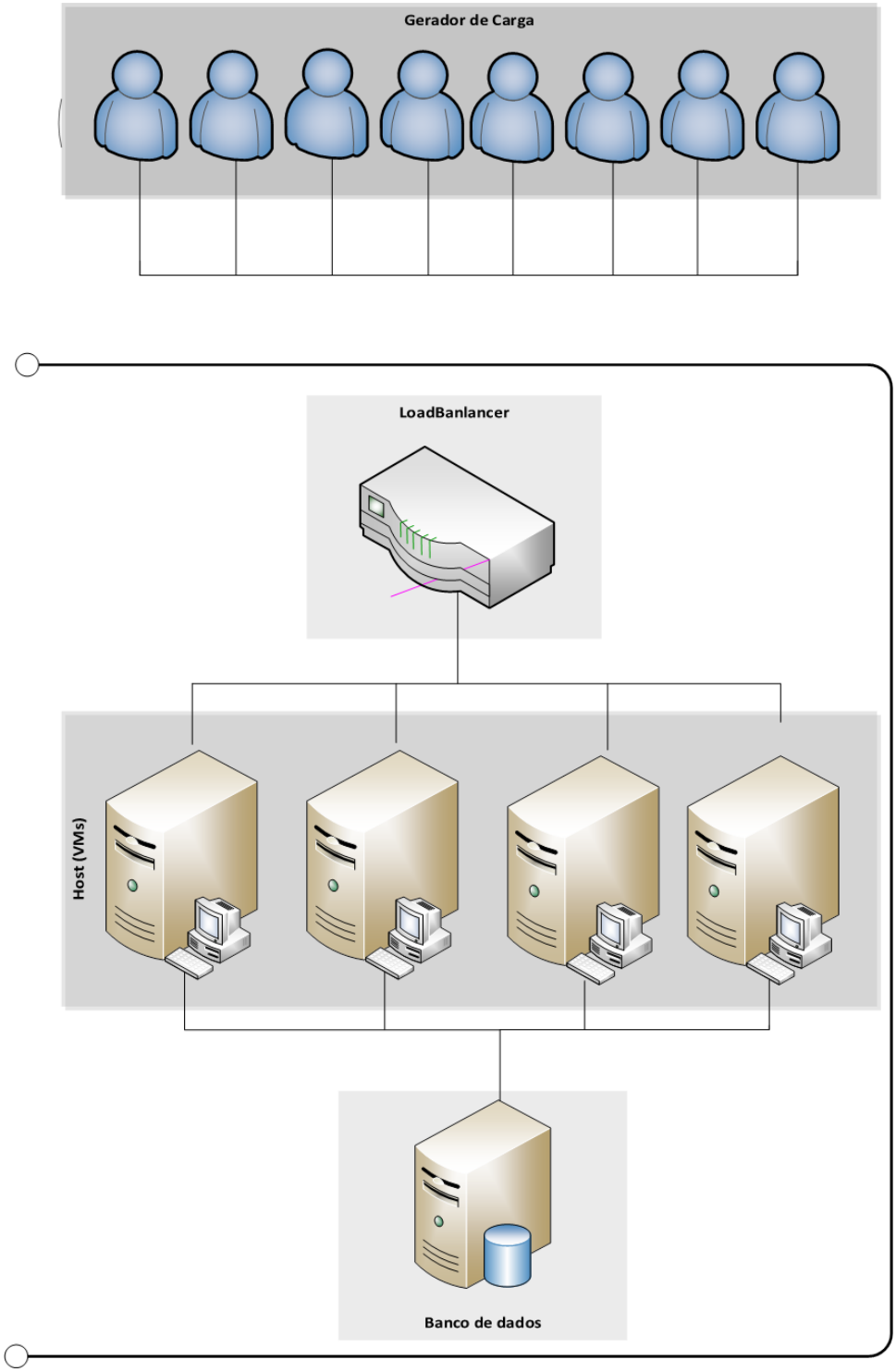


Figura 17 – Arquitetura do experimento

Fonte: Elaborada pelo autor.

baterias de experimentos em objetivos distintos. O primeiro, é no caráter de certificar e verificar a correta modulação da carga conforme a programada atrás dos parâmetros da extensão. Esses experimentos tem por objetivo reproduzir a forma da carga de trabalho representada na Figura 16. A segunda bateria de experimentos, tem por objetivo verificar o impacto dessa modulação

da carga de trabalho em um ambiente real controlado de *multi-tiers* conforme a Figura 17. A configuração do ambiente utilizado para execução dos experimentos são apresentados na Tabela 1. Foram utilizadas oito unidades para a geração de carga de trabalho (*workload*) que atuam como clientes dos serviços, 1 unidade de balanceamento de carga, 4 servidores atuando como provedor de serviços e 1 unidade executando o banco de dados a ser consultado pelo provedor de serviços.

Tabela 1 – Especificação do ambiente de execução dos experimentos

Componente	Quantidade	Configuração
Workload	8 Unidades	Intel Core 2 Quad Q9400, 8 GB RAM
Load Balancer	1 Unidade	Intel Core I7 3.60GHZ, 32 GB RAM, HD 2TB Sata III
Hosts	4 Unidades	Intel Core I7 3.60GHZ, 32 GB RAM, HD 2TB Sata III
Data base	1 Unidade	AMD Vishera 4.2 Ghz, 32 GB RAM, HD 2TB Sata III

Fonte: Dados da pesquisa.

Nesta segunda bateria de experimentos é necessário a definição de métricas para se observar o impacto da carga no ambiente. Para a execução dos experimentos foram definidos e utilizados um conjunto de métricas a fim de validar a modulação da carga de trabalho e observar o impacto provado no sistema. A métrica é uma função que transforma resultados medidos em uma forma facilmente compreendida (FOLKERTS *et al.*, 2013). As métricas de referência devem permitir caracterizar e quantificar o comportamento do sistema quando enfrenta perturbações (ou seja, falhas, ataques, e variações de ambiente operacional) (VIEIRA *et al.*, 2012). As métricas tradicionais, de analise estacionaria, não podem capturar os comportamentos transitórios do sistema em resposta a modulação da carga de trabalho.

Rosu *et al.* (1997) apresentam as características e os comportamentos de uma métrica transiente, conforme ilustrado pela Figura 18, que são:

- ***Reaction Time* (Tempo de reação)** - o período entre a ocorrência da variação crítica e a conclusão da promulgação realocação de correção;
- ***Recovery Time* (Tempo de Recuperação)** - o intervalo entre a conclusão, promulgação e da restauração de um nível de desempenho aceitável;
- ***Performance Laxity* (Frouxidão desempenho)** - a diferença entre o *required vs performance*, e o desempenho em estado estacionário, após a redistribuição;

A métrica em questão deve ser identificada dentro da realidade e necessidade em que se encontra o sistema a ser avaliado. Logo seria uma ingenuidade fixar um conjunto de métricas para um sistema desconhecido, o importante é que ela tenha o comportamento e as características apresentadas por Rosu *et al.* (1997). Existem diversos trabalhos dedicados a identificação de métricas transientes em vários contextos como apresentado por Binnig *et al.* (2009), Lu *et al.*

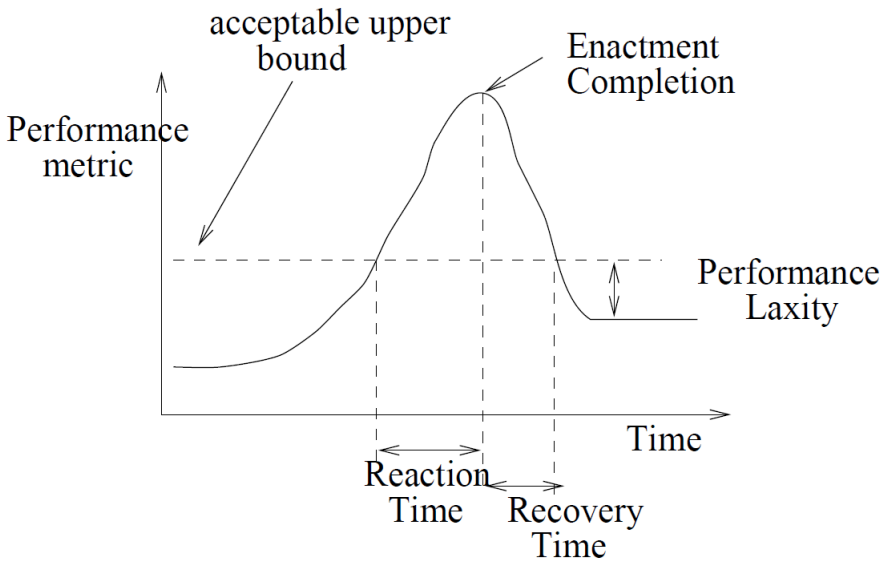


Figura 18 – Comportamento de métrica transiente

Fonte: Rosu *et al.* (1997).

(2000) e Rosu *et al.* (1997). Binnig *et al.* (2009) afirmam que os *benchmarks* tradicionais estão principalmente preocupados com o desempenho e o custo de sistemas estáticos e essas métricas ainda têm relevância para as aplicações em Computação em Nuvem, mas é necessário medir diferentes métricas para sistemas escaláveis (ou seja, dinâmicos) onde os recursos vêm e vão. Ainda Binnig *et al.* (2009) enfatiza que os novos *benchmarks* devem relatar métricas diferentes do que os *benchmarks* existentes: Ao invés de medir o desempenho médio de um sistema estático em carga máxima, as novas métricas devem refletir a capacidade dos serviços em nuvem para se adaptar a uma mudança de carga com relação ao desempenho e custos. Além disso, uma métrica adicional também deve cobrir a robustez desses serviços contra falhas de nós individuais.

As métricas definidas precisam refletir o cenário de dinâmica do sistema, é interessante também cobrir cada um dos níveis da arquitetura proposta a fim de observar o impacto da carga de trabalho em cada um dos níveis. Neste projeto é interessante salientar:

- **Conexões por segundo (*Load Balancer*):** Conforme sugerido por Binnig *et al.* (2009), medir a escalabilidade através do aumento dos interações web emitidos por segundo ao longo do tempo e de forma contínua contando a interação web que são respondidas em um intervalo de tempo de resposta,

- **Tempo de resposta (*browsers*):** Luz (2014), em um sistema dinâmico, cuja transformação entrada-saída não ocorre em tempo zero, mas é sujeita a uma inércia advinda dos processos físicos associados, possuí uma inércia intrínseca que atrasa o efeito que uma entrada terá na saída. Esses efeitos refletem no consequentemente nos comportamentos diversos que

incluem retardo no tempo de resposta e possíveis oscilações;

- **Taxa de utilização da CPU (VMs):** Nobile (2013) afirma que diversas métricas podem ser analisadas para verificar o desempenho das máquinas virtuais, e cita alguns exemplos, como o tempo de inicialização, a taxa de utilização de CPU, o tempo médio de resposta e o *throughput*, e usualmente, número de máquinas virtuais que hospedam serviços de interesse ao cliente e que respondem a uma carga de trabalho imposta por usuários através de requisições;

- **Taxa de utilização da CPU:** O trabalho apresentado por Wang *et al.* (2009), que lida com uma carga de trabalho variante no tempo e intensiva, demonstra que a CPU e I/O podem ser utilizadas para prever as necessidades dos recursos de um banco de dados e para orientar a alocação de recursos *on-demand* de acordo com a exigência de carga de trabalho. Entretanto iremos somente considerar em nossos experimento a taxa de utilização do banco de dados.

Uma vez que a extensão é desenvolvida e a configuração física do ambiente é estabelecida, a configuração lógica do ambiente de medição tem de ser especificada. Nesse contexto, é de interesse analisar o impacto da carga de trabalho no ambiente. Logo, se faz necessário efetuar um experimento com uma carga de trabalho modelada, bem como a formulação subsequente do experimento.

Considerando a utilização de um ambiente físico real e controlado juntamente com o objetivo experimental, utilizamos um único fator, a carga de trabalho (*workload*) conforme apresentado na Tabela 2.

Tabela 2 – Fator e nível dos experimentos

Fator	***Workload***	
Nível 1	60 EBs	

Fonte: Dados da pesquisa.

O único fator refere-se à quantidade de clientes simultâneos requisitando o serviço do *e-commerce*. Esses clientes são modulados conforme a configuração feita. Neste estudo, 3 replicações foram executadas para o experimento. Durante a execução dos experimentos, pequenas aplicações, monitores, em cada um dos níveis da arquitetura são responsáveis por coletar algumas informações de interesse disponíveis (métricas), como a taxa de utilização de CPU e a quantidade de requisições simultâneas. Os valores produzidos pela experimentação foram adicionados em arquivos Excel (*.xls*) e processados pelo R [1].

[1] <https://www.r-project.org/>

CAPÍTULO

4

DESENVOLVIMENTO

Nesta seção são descritos de maneira detalhada a implementação realizada no *benchmark* Bench4Q e a abordagem utilizada no processo de desenvolvimento da extensão. Descrevemos também as implementações das distribuições de carga de trabalho e as disponibilizamos sob uma licença de código aberto, com o intuito de que outros possam reutilizar a plataforma desenvolvida e estender o *benchmark* segundo as suas necessidades para contribuir com outros tipos de modulação de cargas de trabalho num ambiente de experimentos controlado [1].

Dentro do processo de experimentos de sistemas de Computação em Nuvem onde é analisado o comportamento da resposta do sistema a uma perturbação no regime transiente, foram identificadas algumas faltas de funcionalidades fundamentais, principalmente na geração da carga de trabalho. Especificamente, o Bench4Q possui algumas limitações na sua versão original, principalmente pela definição do próprio projeto de *software* para avaliação de desempenho no regime estacionário. Esta limitação foi uma barreira que dificultou a experimentação e analise nos trabalhos relacionados com o presente projeto de Mamani (2015) e Júnior *et al.* (2015b).

No processo de alteração do Bench4Q, foram encontradas classes disponíveis originalmente no *benchmark* que permitem a modulação de carga de trabalho no instante inicial dos experimentos (processo conhecido como processamento em lote). A simulação de uma carga de trabalho em que há a alteração introduzida ao longo do experimento é o foco deste trabalho, as limitações encontradas, por exemplo, dificultam projetar um controlador para o gerenciamento de recursos em tempo real, pois para essa atividade é necessário uma análise de resultados transientes mediante a modulação da carga de trabalho, como foi realizado por Nobile *et al.* (2007).

O modelo de referência MEDC define um conjunto de requisitos ao qual a presenta proposta de extensão deve-se segui-la e respeitá-la, entre os requisitos, o *Demand*. Segundo Mamani (2015), o requisito *Demand* visa a avaliação de desempenho de sistemas computacionais

[1] O código fonte está disponível em http://gitlab.lasdpc.icmc.usp.br/edwin/bench4q

considerando mudanças na carga de trabalho que chega ao sistema. Para o contexto do Bench4Q, a carga de trabalho são requisições Web que são submetidas à camada de aplicação do *benchmark*.

O Console do Bench4Q é o módulo que gerencia EBs para gerar uma série sequencial programada de requisições que são submetidas para o SUT. O tópico aqui é controlar a taxa em que os EBs geram as requisições, para que seja possível controlar diretamente e de forma programada a carga de trabalho. Nativamente o Bench4Q, coleta informações sobre a taxa requisições e o comportamento do SUT, e relata esses dados no final da experimento. A Figura 19 ilustra de maneira geral o fluxo desse contexto.

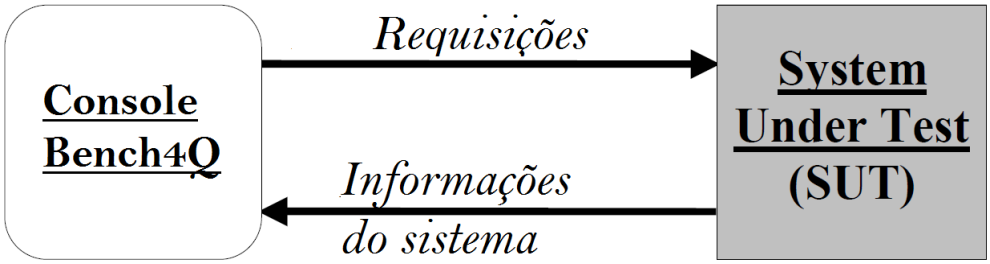


Figura 19 – Configuração experimental Bench4Q.

Fonte: Adaptada de Vieira e Madeira (2003).

A extensão implementada na geração de carga do Bench4Q segue a orientação do requisito *Demand* do MEDC, criando novas classes e modificando algumas já existentes da versão original do *benchmark*. Conforme o diagrama de classes na Figura 20, é possível ter uma ideia de alto nível da extensão realizada no *benchmark*. Vale salientar que o Bench4Q é uma ferramenta completa e extensa, por tais motivos são somente apresentadas algumas das classes mais representativas que passaram por modificações para atender aos requisitos da proposta juntamente às novas classes que foram necessárias para cumprir o objetivo. A Figura 20 mostra o diagrama de classes da extensão do Bench4Q. Como pontos de destaque, as classes sinalizadas na cor azul representam as já existentes, mas que passaram por adaptações e modificações, já as classes na cor verde referem-se as novas classes criadas para possibilitar a modulação da carga do *benchmark*.

O Bench4Q fornece uma estrutura e componentes compartilhados para a comunicação entre os dois módulos da carga de trabalho: **Console** e **Agente**. A extensão foi construída inicialmente sob a classe MLoadSimulatorPanel, que orquestra toda a interatividade gráfica do Bench4Q. O novo painel de configuração, que modula a carga, MLoadFrequencyPanel estende a classe original Bench4QTreeModel, adicionando os parâmetros para a modulação como: tipo da carga, o instante em que a carga se inicia, o tempo de atuação da carga e a quantidade de EBs que atuaram nessa carga. O parâmetro *"tipos de carga"*, utiliza da classe enum TypeFrequency que define as constantes dos tipos de modulações programadas para esta extensão. Todos os parâmetros inseridos na MLoadSimulatorPanel são armazenados na classe TestFrequency que se tornou uma propriedade da classe nativa TestPhase, e que posteriormente são repassadas para a

classe PropertiesEB através da FrequencySettings. Já as classes Agent, EB, EBClose, EBOpen, Workers, WorkersClosed e WorkersOpen foram modificadas para receber os novos parâmetros da PropertiesEB gerando a carga programada durante a execução dos experimentos.

O trabalho de desenvolvimento da extensão compreendeu extensivas modificações no código-fonte original do Bench4Q e a criação de novas funcionalidades. Algumas das principais modificações são descritas nas próximas seções.

4.1 Configuração da carga de trabalho

O módulo Agente do Bench4Q está diretamente ligado ao módulo Console do *benchmark* onde são definidas as configurações inicias para os experimentos. Essa caraterística da arquitetura do *benchmark* permite que diversos clientes (Agentes), sejam configurados e gerenciados de maneira organizada, assim como, fornecendo uma potencial escalabilidade para a execução dos experimentos, permitindo variados ambientes com alta ou baixa concorrência ou quantidade de requisições.

O Cliente é um programa Java para gerar as operações que compõem a carga de trabalho. Cada EB é representado por uma *thread* que executa uma série requisições com diferentes *think times* governados por uma função exponencial que finalmente são submetidas ao SUT. Para distribuir e controlar a submissão da carga de trabalho ao longo da simulação, uma estratégia utilizada é a modulação por meio de parâmetros que configuram o comportamento da carga. O cliente tem uma série de propriedades que definem o seu funcionamento e o comportamento resultante da carga de trabalho, apresentados no Capítulo 3.

O pseudo-código 1 refere-se a classe FrequencySettings criada para a extensão, que contém o algoritmo responsável por calcular os tempos de inicialização, pause e termino de cada um dos clientes. Este código é o ponto central que resulta no comportamento final da modulação da carga de trabalho. Entre as tarefas encontra-se definir os períodos (*think time*) de execução e interrupção para cada EBs, como um planejamento de tarefas. Para calcular os períodos de execução e interrupção são utilizados outros parâmetros nativo ao Bench4Q como o Tempo de Experimento. O código fonte apresentado na integra pode ser conferido no Apêndice A código 4.

No princípio um conjunto de variáveis (timeStart, timeEnd, timePause e timeExperiment) são inicializadas, em geral com valores informados previamente e mesurados em milesegundos. A condição IF verifica o tipo de modulação a ser gerada. Em seguida, outra condicional IF verifica, através do index, se o EBs é configurável para a modulação e calculando os tempos de inicializações e interrupções de cada EBs.

Código-fonte 1: Algoritmo calcula os tempos de inicialização e termino para cada um dos Clientes

```
1   ParametrosExperimento createProperties(Time tempoExperimento) {
```

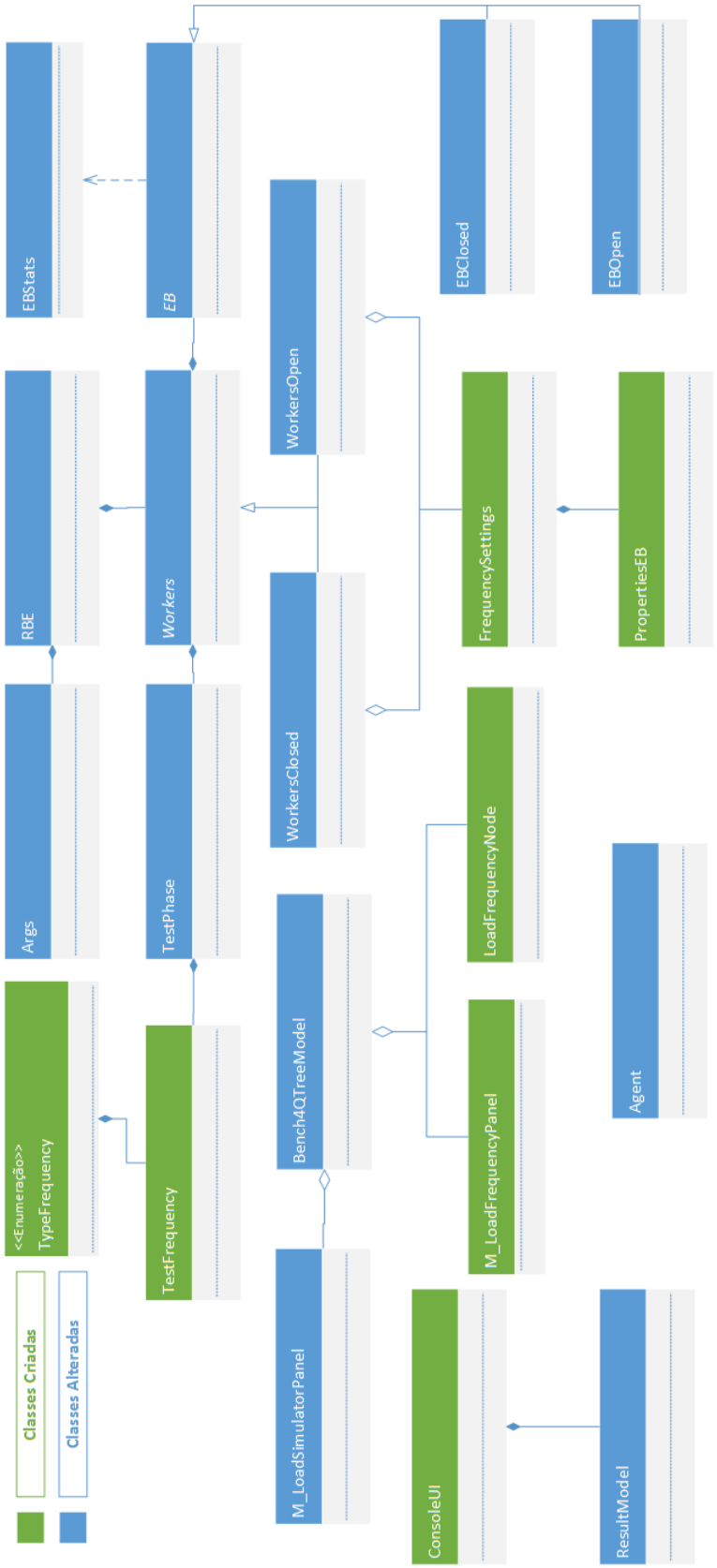


Figura 20 – Diagrama de classes da extensão do Bench4Q.

Fonte: Elaborada pelo autor.

```
2
3     ParametrosExperimento parametros = new ParametrosExperimento();
4
5     parametros.TempoIncial = calculaTempoInicial();
6     parametros.TempoFinal = calculaTempoFinal();
7     parametros.TempoPausa = calculaTempoPausa();
8
9     retorna parametros;
10   }
11 }
```

Novos tipos de modulações podem ser adicionadas a esta versão do Bench4Q. Isto é especialmente útil para *benchmarks* que têm requisitos muito especiais para a sua validação e/ou análise de desempenho. Por esse motivo foi criada a classe FrequencySettings para que outros tipos de modulações sejam implementadas com os seus cálculos necessários para a sua execução tornando simples a criação de novos tipos de carga de trabalho para outros desenvolvedores. As novas modulações a serem implementadas deve ser incluídas neste método, juntamente com o nome do novo tipo de carga na classe Enum *TypeFrequency*.

4.2 Geração da carga de trabalho

A princípio foi identificado o modulo de geração de carga do Bench4Q e este passou por alterações para gerar a carga de trabalho esperada. Logo, esta classe é nativa do *benchmark*. O Bench4Q tem dois tipos de sessões, *Open* e *Close*, e estas tem grande influência na geração da carga de trabalho. Consequentemente existem duas classes que tratam cada umas das sessões (*WorkersOpen* e *WorkersClose*) com o mesmo métodos, mas com implementações diferentes. O pseudocódigo 2, apresentado na a seguir, ilustra o *core* da geração da carga referente a construção da modulação da carga já com as modificações da extensão, este método é o encarregado do gerenciamento da geração dos EBs. Através dos parâmetros e dados calculados anteriormente é possível controlar as requisições afim de gerar as modulações desejas.

Para maiores detalhes o código-fonte é apresentado na integra no Apêndice A código 5. O código é extenso, aqui iremos descrever somente os trecho de interesse a modulação da carga de trabalho. O loop nativo *while* repete em quanto o valor parametros.ExperimentoEmExecucao for verdadeiro (parametros.ExperimentoEmExecucao, é o número de transações a serem efetuadas). A primeira condicional IF verifica se o tempo corrente é maior que o tempo de experimento, compreendendo assim a finalização do experimento a interrupção de novas requisições, assim a variável nativa é atribuída com o valor false encerrando as execução (código-fonte 2: linha 34). Já na segunda condicional IF, é verificado se o tempo corrente é maior que o tempo final de execução do EB somente para os EBs que são moduláveis. Neste caso é verificado, na sub-condicional IF, se existem pausa programadas e calculados novos tempos de execução e interrupção do EB, caso

contrário é encerrada a execução do EB com a atribuição false para a variável (código-fonte 2: linha 14). A terceira condicional IF (código-fonte 2: linha 25), trata se o tempo corrente é maior ou igual ao tempo de início de execução do EB; para este caso será gerada uma requisição, caso a sub-condicional IF (código-fonte 2: linha 34) não seja verdadeira, pois esta encerra a execução do EB.

Código-fonte 2: Algoritmo de geração de carga modificado para modulação

```
1   ParametrosExperimento parametros;
2   ebCorrente.EmExecucao = true;
3   while (parametros.ExperimentoEmExecucao) {
4     tempoCorrente = System.pegaTempoCorrente();
5
6     if (tempoCorrente > parametros.TempoExperimento){
7       ebCorrente.EmExecucao = false;
8     }
9
10    if (tempoCorrente > ebCorrente.TempoDuracao && ebCorrente.
      EbMarcado) {
11      if(parametros.TempoPausa > 0){
12        long novoInicio = ebCorrente.TempoFinal + parametros.
      TempoPausa ;
13        long periodo = ebCorrente.TempoFinal - ebCorrente.
      TempoInicial;
14
15        ebCorrente.TempoInicial = novoInicio;
16        ebCorrente.TempoFinal = periodo + novoInicio;
17      } else if (tempoCorrente > ebCorrente.TempoInicial) {
18      ebCorrente.EmExecucao = false;
19    }
20  }
21
22  if (tempoCorrente >= ebCorrente.TempoInicio) {
23    if (!ebCorrente.EmExecucao) {
24      return;
25    }
26    if (ebCorrente.TemProximaPagina) {
27      // fluxo de acesso a pagina do SUT (recurso nativo do Bench4Q)
28    } else {
29    ebCorrente.EmExecucao = false;
30  }
31  if (!ebCorrente.EmExecucao = false;) {
32    return;
33  } else {
34  ebCorrente.Dorme(500);
35 }
36
```

```
37 }
```

Modelagem de carga de trabalho é uma tentativa de criar um modelo simples, que pode então ser utilizada para gerar cargas sintéticas. O objetivo é de ser capaz de criar cargas de trabalho que podem ser utilizados em estudos de avaliação de desempenho com semelhanças as cargas de sistemas reais.

4.3 *Interface* gráfica

No Console são configurados os parâmetros iniciais para a execução do experimento, foi incluída uma nova opção *LoadFrequency* no menu do Bench4Q, que referente aos parâmetros da extensão da geração da carga. Por esta opção, *LoadFrequency*, deve-se preencher os campos (*Start Time*, *Duration Step*, *Pause* e *Quantity*) que irão gerar a carga conforme a programação. O pseudo-código 3 apresenta a criação da interface gráfica, o método *private* presente na classe *MLoadFrequencyPanel* que cria a *interface* gráfica gerada com a biblioteca *Swing* do Java, toda a parte gráfica do Bench4Q utiliza da mesma biblioteca. Todos os resultados de desempenho de cada agente de carga são agregados no console de carga para análise e demonstração, conforme a versão original. O código fonte pode ser apreciado na integra no Apêndice A código 6.

Código-fonte 3: Código para gerar a os parâmetros para a modulação

```
1   private void createPanelFunction(Menu menu) {
2
3     Panel panelLoadFrequency = menu.addItem(novo Panel());
4     panelLoadFrequency.setNome("LoadFrequency");
5
6     panelLoadFrequency.addCampo("Start Time");
7     panelLoadFrequency.addCampo("Duration Step");
8     panelLoadFrequency.addCampo("Pause");
9     panelLoadFrequency.addCampo("Quantity");
10
11    menu.atualizar();
12    menu.reconstruir();
13
14  }
```

Este conjunto de classes as quais lidam, manipulam, gerenciam e modulam a carga de trabalho gerada pelo Bench4Q são utilizadas pelo Console para configurar, monitorar e analisar todo o experimento. Todo o desenvolvimento, referente à modificação e implementação de novas classes, mantiveram e respeitaram o padrão de desenvolvimento do *benchmark*. A Figura 21 ilustra a interface gráfica por onde é possível modular a carga de trabalho do Bench4Q.

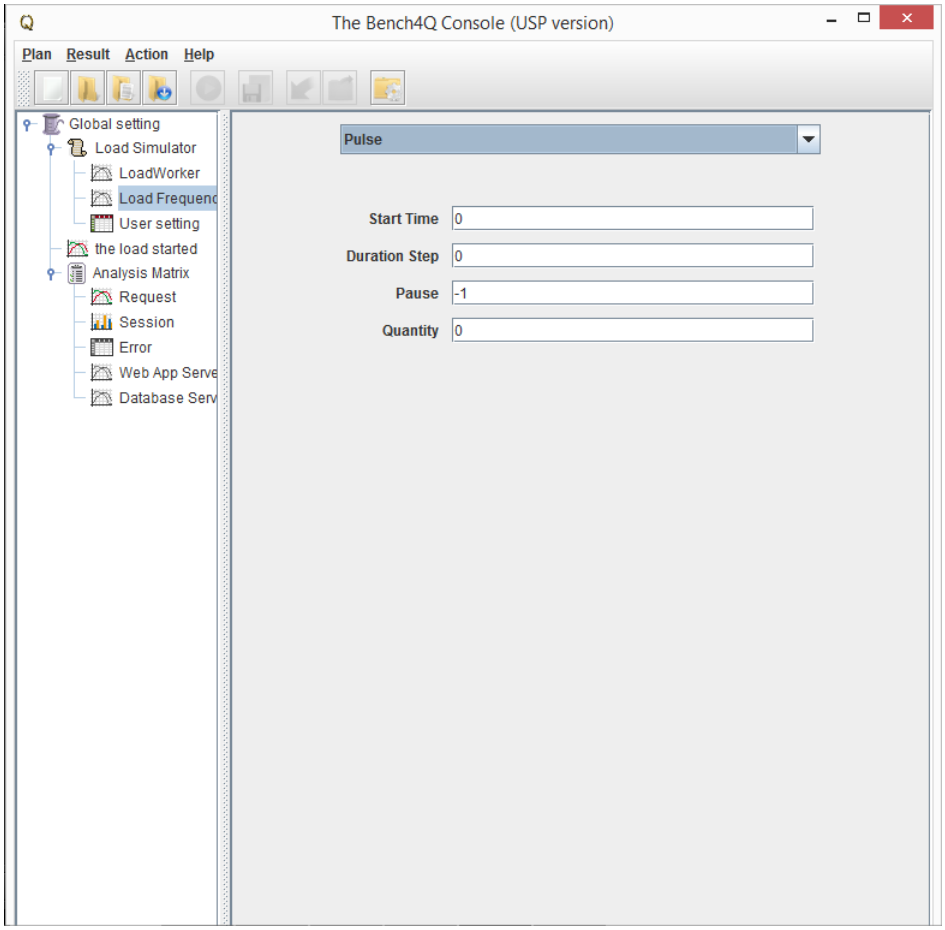


Figura 21 – Console de programação da carga de trabalho.

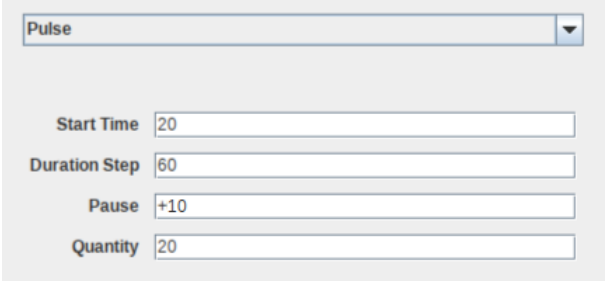
Fonte: Elaborada pelo autor.

4.4 Teste de modulação

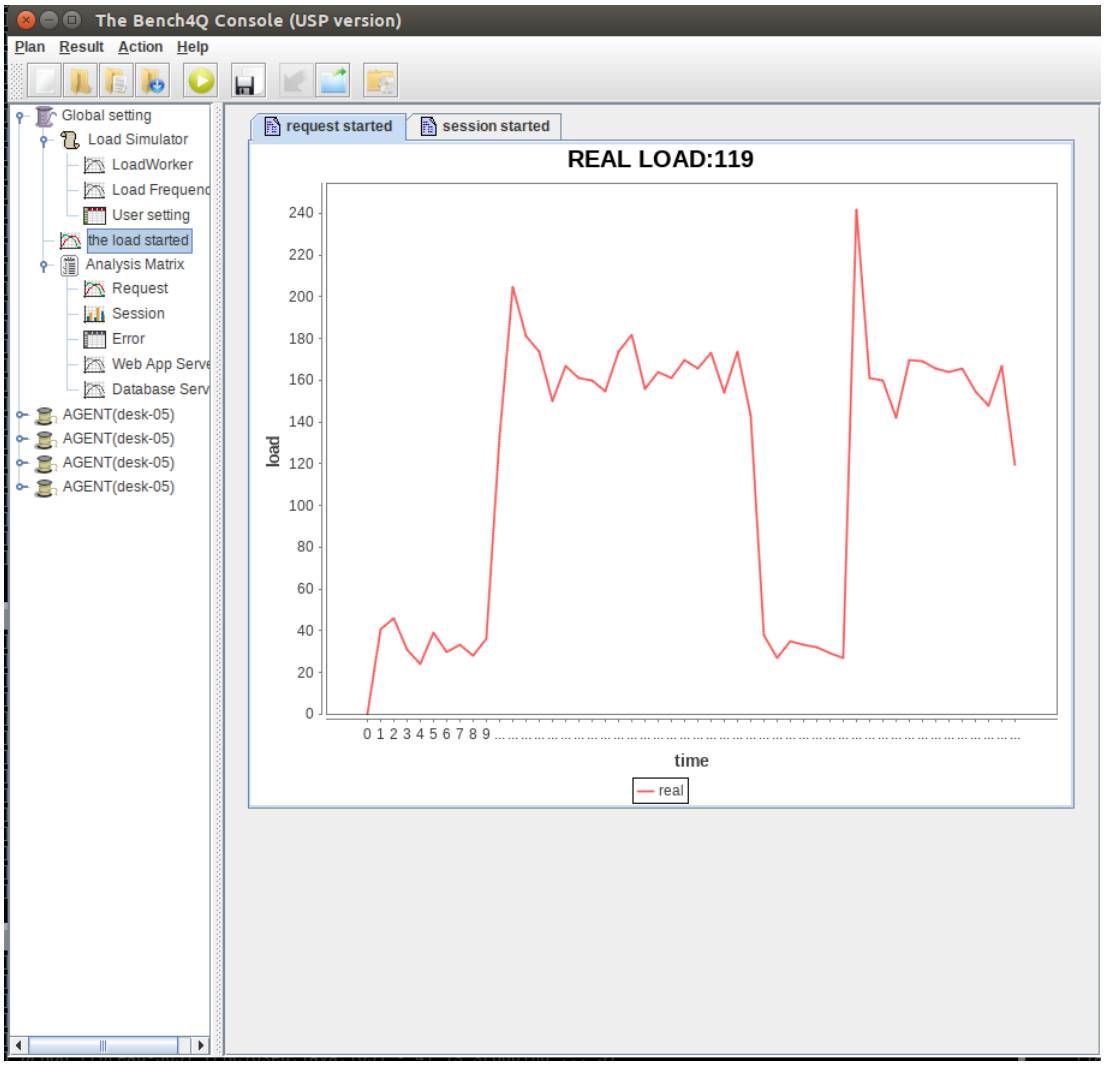
A carga de trabalho é imposta ao sistema por meio de requisições HTTP enviadas pelos EBs ao SUT que são processadas nos servidores de aplicação das máquinas virtuais instanciadas no *host*. Essas requisições exigem que as máquinas virtuais se ocupem pelo tempo necessário para processá-las, alterando o desempenho experimentado pelo sistema. Segundo Nobile (2013), existem dois fatores associados a uma requisição e que afetam diretamente o desempenho do sistema: o tempo de processamento e a quantidade de carga imposta pelas requisições, são dados pelo tempo de processamento e pela taxa de chegada de novas requisições, respectivamente. Com o tempo, a quantidade e o tamanho das requisições podem se alterar, dependendo do perfil de utilização dos usuários que utilizam o serviço naquele momento. Havendo um aumento em algum desses fatores é possível que o desempenho do sistema sofra degradação, podendo em casos extremos, entrar em colapso.

É possível informar previamente a execução dos parâmetros da modulação. Por exemplo, ao escolher a opção degrau, é necessário informar quantos EBs geram o degrau, em que instante de tempo, e qual o tempo de duração e por fim qual a sua polaridade (com base em um pulso

elétrico a positiva sairia de zero e chega a um, a negativa, sairia de um e chegaria a zero), é possível obter resultados conforme a Figura 22b.



(a) Teste de configuração da carga a ser modulada



(b) Carga gerada com base na configuração teste

Figura 22 – Teste de modulação da carga

Fonte: Elaborada pelo autor.

A Figura 22 ilustra uma carga teste modulada já pela extensão. A Figura 22a apresenta os parâmetros utilizado para realizar o teste. A carga modulada atuará a partir do décimo segundo de experimentação e com uma duração de 20 segundos; com 30 segundos de experimentação

ocorrerá uma pausa de 7 segundos e um novo degrau será gerado em seguida, o qual se manterá até o final do experimento. Para este exemplo foram fixados 40 EBs por Agente para modular o comportamento da carga. Este comportamento pode ser apreciado no item 22b da mesma Figura 22. Vale salientar que o gráfico gerado e apresentado na figura 22 de item 22b, é uma característica nativa ao *benchmark*.

Foi elaborada uma documentação seguindo os padrões da última versão original do *benchmark* e esta pode ser conferida no apêndice B que traz informações do programa e qual seu objetivo, entradas suportadas e saídas esperadas, exemplo de como executar o programa e tabela descrevendo as principais características.

CAPÍTULO

5

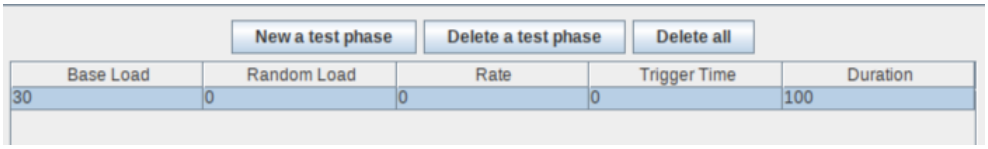
RESULTADOS

As análises apresentadas nessa seção resumem os resultados experimentos executados na extensão feita no Bench4Q. Os resultados aqui mostrados são proporcionados como exemplos de aplicação da metodologia. Dessa forma, para uma melhor abordagem, divide-se os exemplos em partes:

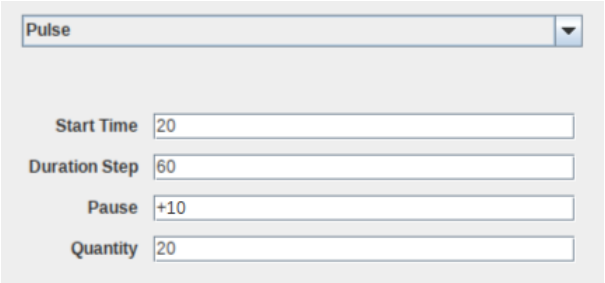
- Configuração da carga no Bench4Q;
- Configuração para modular a carga;
- Resultado da carga gerada.

A Figura 23 apresenta a primeira exemplificação gerada com o Bench4Q já com a extensão desenvolvida neste trabalho. A figura 23a demonstra os parâmetros de configuração utilizados para gerar a carga. Dois são os principiais: *Base Load* e *Duration*. O primeiro define a quantidade e EBs envolvidos no experimento, neste caso 30 EBs; já o segundo define o tempo de duração do experimento em segundos, neste caso 100 segundos. Na Figura 23b são apresentados os parâmetros para modular a carga *Start Time* de valor 20, referindo-se ao tempo de espera para o início do restante da carga que se mostra presente e ativa na modulação. De corrido 20 segundos, 20, dos 30 EBs, definido pelo campo *Quantity* iniciam a geração de carga para o sistema. Essa carga se manterá ativa durante 60 segundos conforme fixado no parâmetro *Duration Step*. Nesse exemplo, o parâmetro *Pause* não apresenta influência devido ao seu valor 0. O resultado pode ser analisado através da Figura 23c. Esse gráfico é nativo do próprio Bench4Q, que demonstra o comportamento da carga no decorrer do tempo. Apesar da estocasticidade a carga, essa modulou-se conforme programado. Essa estocasticidade é característica do Bench4Q, afim de manter reproduzir comportamento realístico como os de clientes acessando um *e-commerce*.

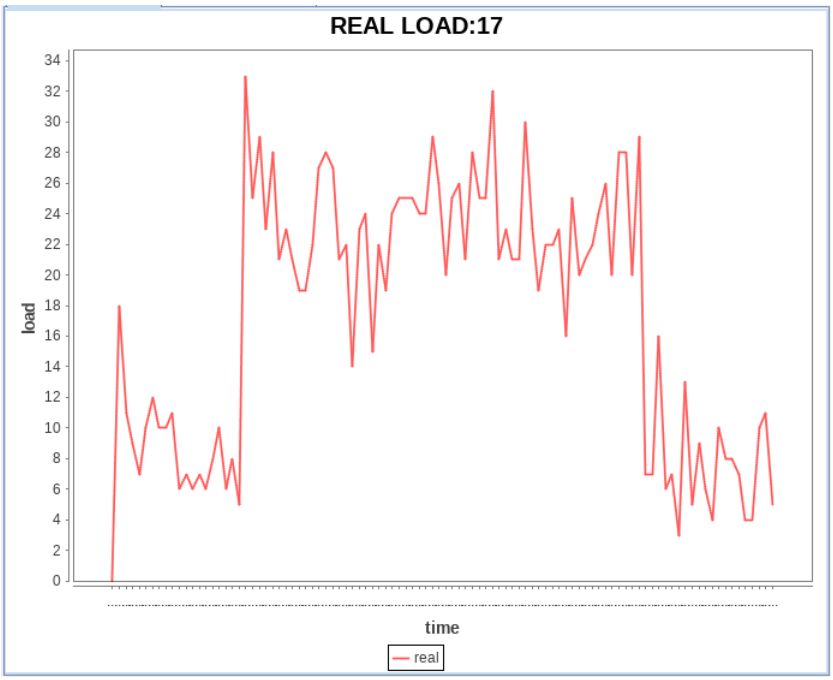
A Figura 24 apresenta os resultados dos parâmetros para o objetivo do Degrau Negativo. Os parâmetros de *Base Load* e *Duration* são os mesmos do experimento anterior: 30 EBs e 100



(a) Configuração da carga no Bench4Q, para um degrau positivo



(b) Configuração para modular a carga como um degrau positivo

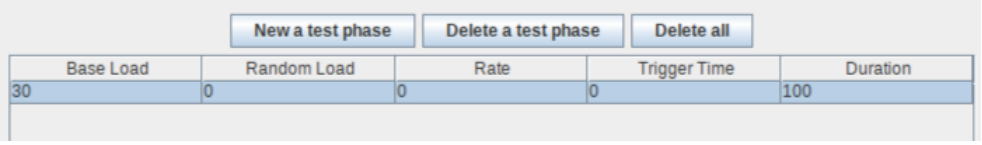


(c) Carga gerada com base nas configurações

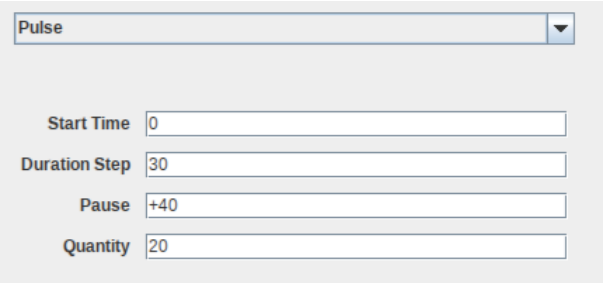
Figura 23 – Carga gerada com base na configuração: Degrau Positivo

Fonte: Elaborada pelo autor.

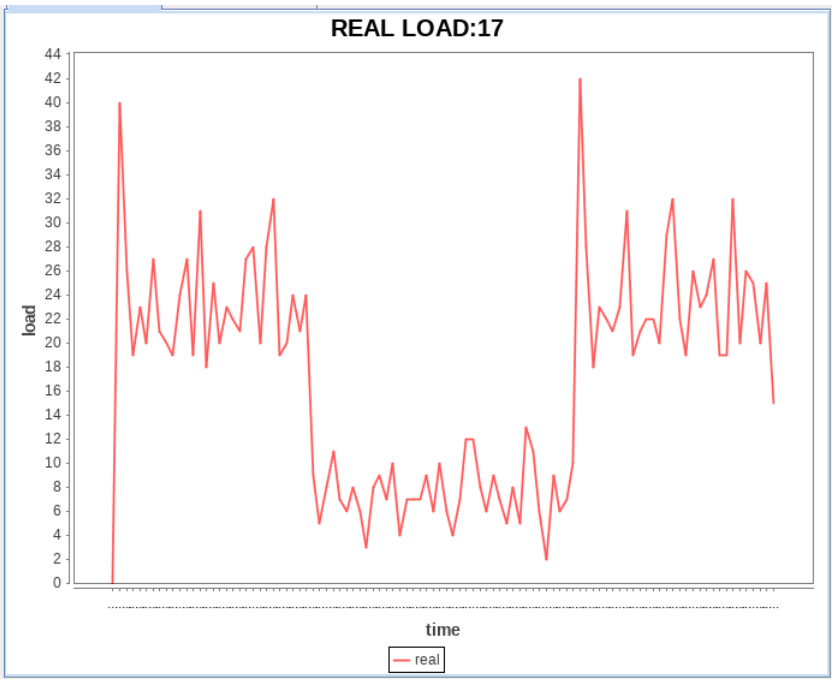
segundos de execução respectivamente, conforme apresentado na Figura 24a. Já na 25b que demonstra os parâmetros utilizados para modular a carga, o *Start Time* recebe o valor 0, assim a carga modulado iniciar com potência máxima utilizando os 30 EBs sendo 20 EBs atribuído no *Quantity* para reservá-los para a modulação, o tempo de carga máxima é de 30 segundos como é possível ver no parâmetro *Duration Step*, neste caso o *Pause* é parametrizado com 40 segundos, este valor é o que fará a interrupção brusca dos 20 EBs caindo o nível da geração de carga, gerando o degrau negativo. Passado esse período *Pause*, a carga retorna ao seu nível máximo e atua por mais 30 segundos, o resultado final pode ser visto na Figura 24c.



(a) Configuração da carga no Bench4Q, para um degrau negativo



(b) Configuração para modular a carga como um degrau negativo

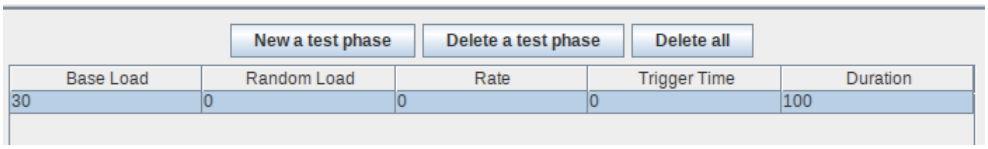


(c) Carga gerada com base nas configurações

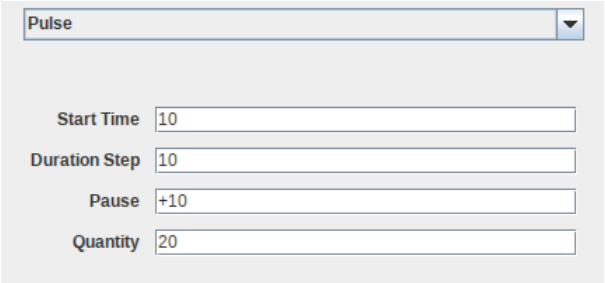
Figura 24 – Carga gerada com base na configuração: Degrau Negativo

Fonte: Elaborada pelo autor.

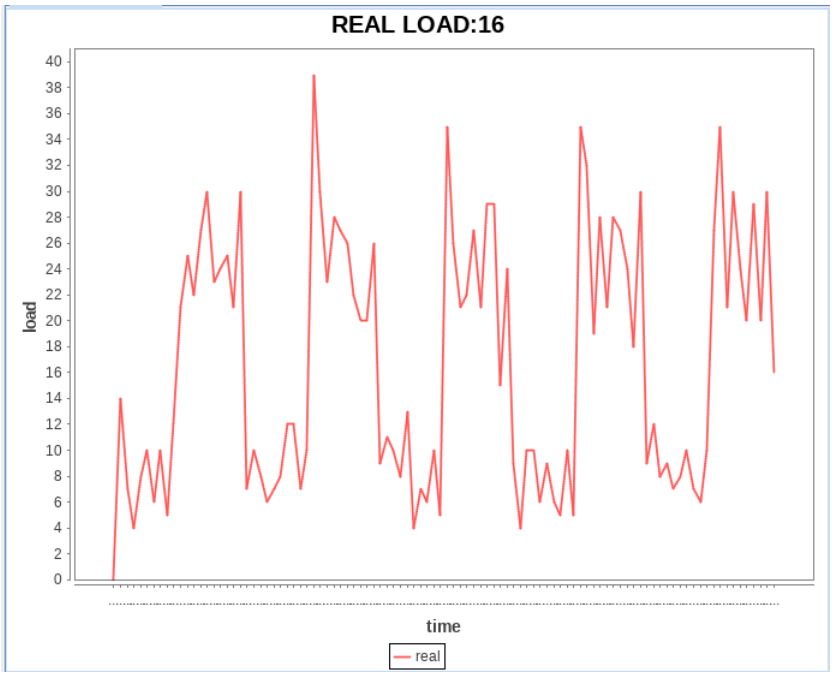
A Figura 25 apresenta o resultado da modulação de uma Onda Quadrada. Os parâmetros iniciais do Bench4Q referente a Figura 25a são os mesmos valores dos outros dois exemplos anteriores. Para gerar uma carga modulada com comportamento oscilatório como a de uma onda quadrada, os parâmetros 25b são definidos com 10 segundos para *Start Time*, 10 segundo de duração para *Duration Step* e 10 para o *Pause*, também configurado com 20 EBs. Para este exemplo vale salientar que para modular a carga como uma onda quadrada dois parâmetros são importantes. *Duration Step* e *Pause*. Esses devem ter os mesmos valores, pois são eles que manterão durante o período definido a carga em níveis baixos e alto.



(a) Configuração da carga no Bench4Q, para uma onda quadrada



(b) Configuração para modular a carga como uma onda quadrada



(c) Carga gerada com base nas configurações

Figura 25 – Carga gerada com base na configuração: Onda Quadrada

Fonte: Elaborada pelo autor.

5.1 Impacto da Carga Modulada no ambiente

Com a possibilidade da modulação de carga, conforme apresentado na seção anterior, é possível verificar o impacto da carga modulada no ambiente projetado para o trabalho de Mamani (2015). Seguindo o planejamento de experimento apresentado no Capítulo 3 foi possível coletar dos dados que geram os gráficos a seguir. Vale lembrar que a cargas de trabalho utilizada, com 60 EBs e um único degrau positivo, que inicialmente utiliza 40% da carga até o instante de 30 segundos de experimentação, quando ocorre um crescimento brusco na carga utilizando os 60% restantes da carga, e gerando um degrau positivo. A carga mantém-se máxima, em 100%,

durante 40 segundos (decorridos 70 segundos de experimentação), quando tem uma queda súbita voltando a trabalhar com 40% da carga até o final do experimento. O objetivo é poder identificar a relação de transformação da entrada na saída, mediante a variação degrau gerada pela carga de trabalho aplicada no sistema.

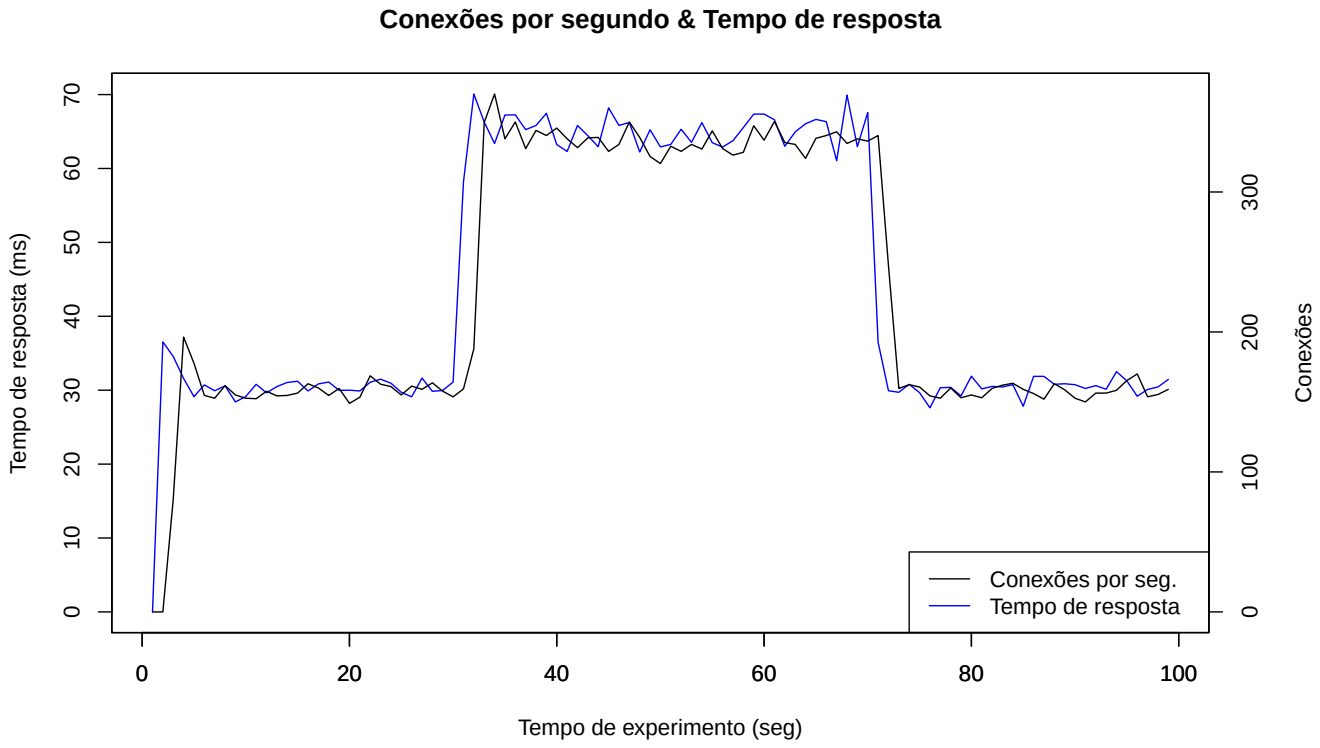


Figura 26 – Conexões por segundo vs. Tempo de resposta

Fonte: Dados da pesquisa.

É possível observar a Figura 26, onde são comparados os desempenhos da conexão por segundo e Tempo de resposta. O Bench4Q monitora os tempos de resposta e, no *LoadBalance*, as conexões por segundo. Ao iniciar o experimento é possível verificar a dinâmica da carga em ambas as métricas, seguida por, um assentamento e uma estabilização o instante de 30 segundos de experimentação, neste instante um aumento de carga é aplicado elevando o tempo de resposta e o número de conexões por segundo. Nesse instante é possível verificar novamente a dinâmica do aumento carga no sistema seguido por um assentamento. Essa elevação das métricas se mantem aproximadamente por 70 segundos de experimento quando uma queda acentuada baixando o nível de operação das métricas que se mantem constantes até o final da execução.

A Figura 27 ilustra as conexões por segundos junto com a média da taxa de utilização de CPU nas VMs. No início dos experimentos a utilização de CPU apresenta uma dinâmica acentuada, estacionando em seguida próximo a 13%. Esse aumento transiente é tão significativo que neste instante chega a atinge a maior média da CPU em todo o experimento, ultrapassando os 40% de utilização da CPU. Da mesma maneira que o volume de conexões por segundo, a

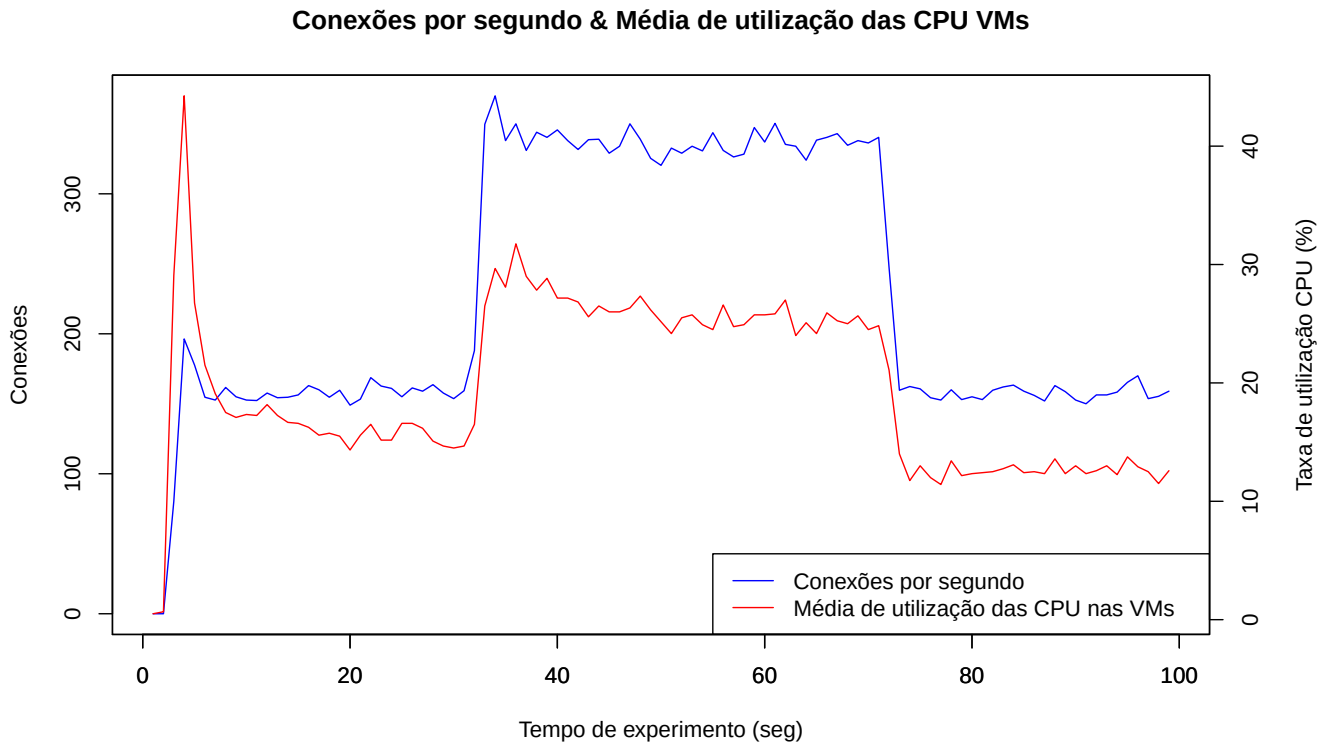


Figura 27 – Conexões por segundo vs. Média de utilização de CPU nas VMs

Fonte: Dados da pesquisa.

média da taxa de utilização de CPU nas VMs sofre alteração no impacto com o aumento da carga de trabalho próximo aos 30 segundos de experimentação. De corrido 40 segundos de carga máxima a utilização inicia de maneira mais enfática, atingindo os 30%. Com o de correr do período de máximo da carga, a taxa de CPU diminui a atingir aproximadamente os 23%. O que chama a atenção na Figura 27, são os trechos de mudança de carga, 0-7, 30-35 e 70-75 segundos aproximadamente. No primeiro instante, de 0 a 7 segundos iniciais, a taxa de utilização sob de maneira busca apresentando uma dinâmica mais acentuada em comparação com o número de conexões por segundo, mesmo que proporcionalmente. Isso é decorrente da inicialização do SUT em cada um dos servidores. A quantidade de tarefas ao iniciar uma aplicação pela primeira vez é maior quando a mesma aplicação é iniciada nas vezes seguinte, devido a inúmeras iterações com outras aplicações para que o serviço fique disponível. Já no intervalo, de 30 a 35 segundos aproximadamente, é possível observar a dinâmica inerente em sistemas multicamadas. O número de conexões por segundo reage a alteração de carga primeiro quando comparada a taxa de utilização da CPU reage primeiro a mudança quando comparada com o número de conexões por segundo.

Referente a Figura 28, onde se compara o número de conexões por segundo com a Taxa de utilização da CPU no banco de dados, podemos observar uma estocásticidade na taxa de utilização da CPU no banco de dados que está relacionada ao ser o gargalo do sistema. É possível

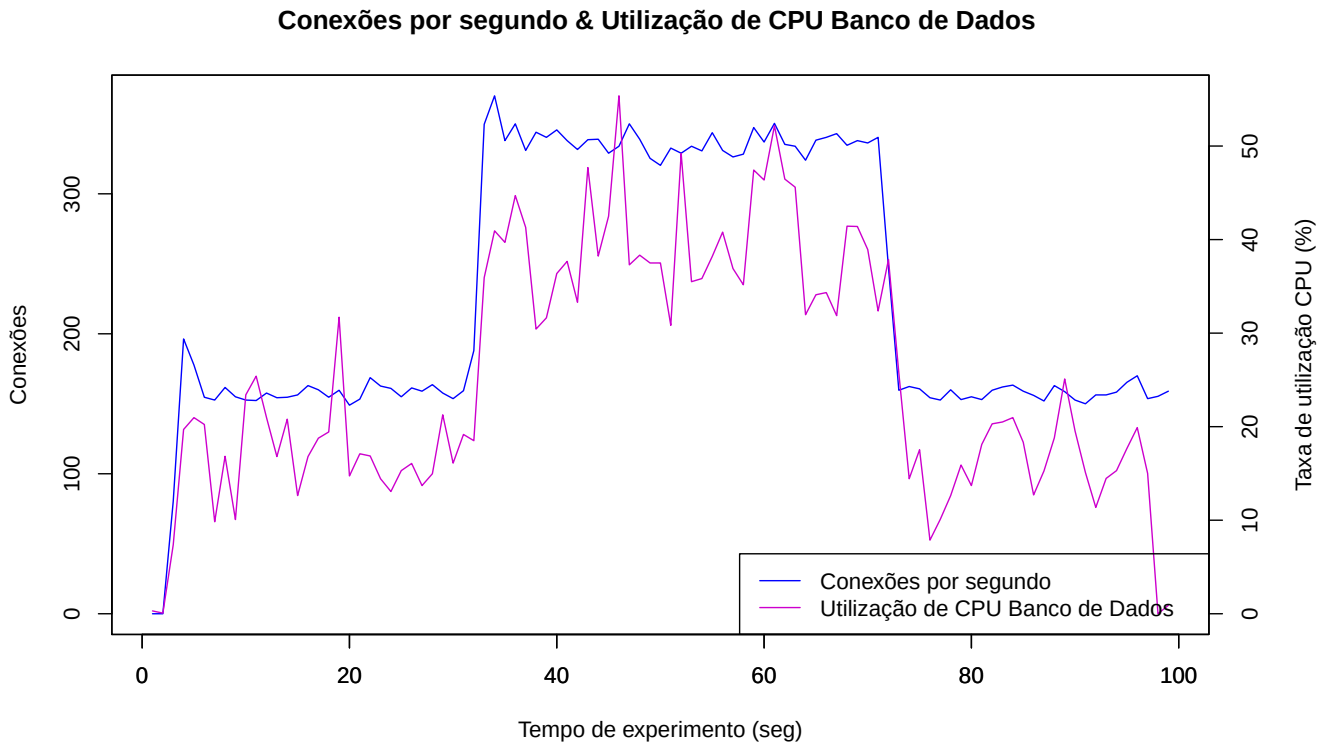


Figura 28 – Conexões por segundo vs. Utilização de CPU no banco de dados

Fonte: Dados da pesquisa.

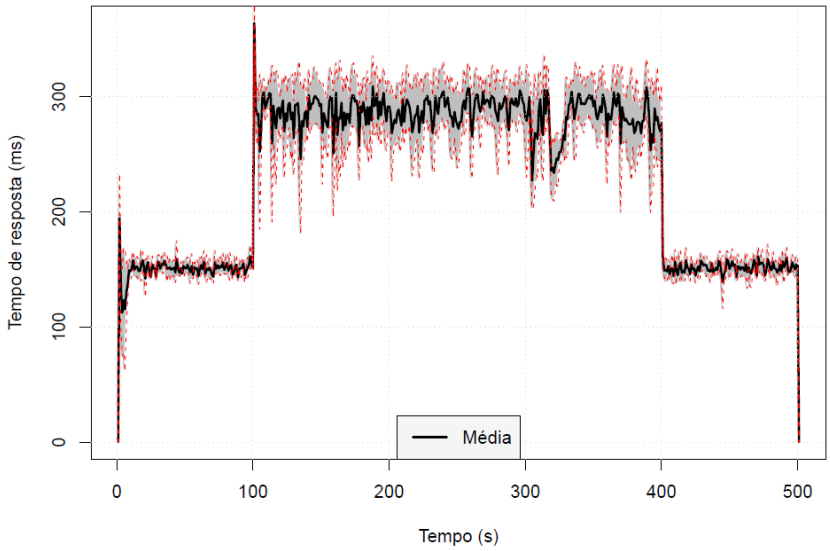
observar a dinâmica de um sistema *multi-tiers*. Do instante 0 até a metade do experimento, o comportamento foi semelhante ao apresentado pela Figura 27, onde o número de conexões por segundo corresponde a taxa de utilização de CPU no banco de dados. Entretanto existe uma convergência no instante da redução da carga de trabalho.

5.2 Contribuições

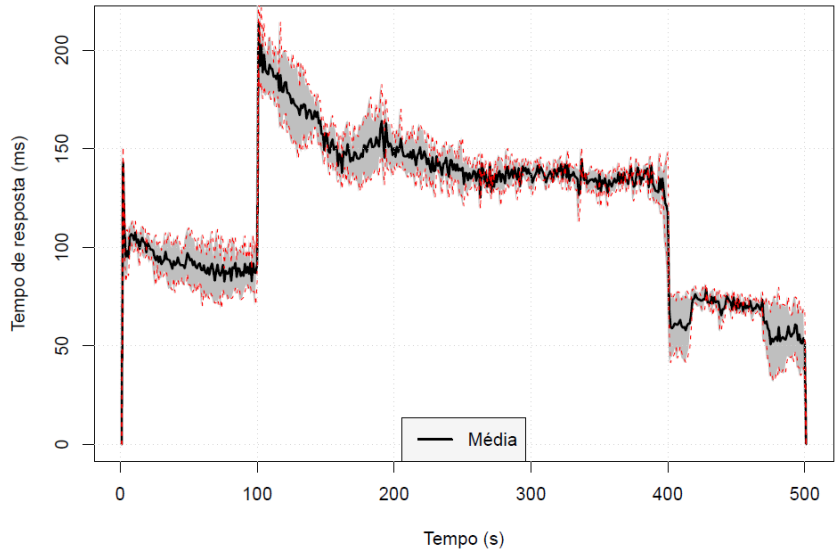
Nesta seção são apresentados resultados indiretos as este trabalho. Resultados que só foram possiveis mediante a extensão desenvolvida por este trabalho e que demonstram uma aplicação e necessidade real da modulação da carga de trabalho.

A Figura 29 apresenta resultados dos experimentos realizados no trabalho do Mamani (2015) que representam a comparação da métrica do tempo de resposta em dois bancos de dados (PostgreSQL 29a e DB2 29b). Na realização deste experimento foi utilizado a extensão no Bench4Q que este trabalho desenvolveu. Desta forma, permite-se compara a dinâmica entre banco de dados. Em ambos foi aplicado a mesma carga modulada, contudo cada banco de dados teve uma resposta diferente.

O PostgreSQL apresentou, conforme a Figura 29a, uma reação a mudança abrupta da carga de trabalho elevando o tempo de resposta de 150 milissegundos para 280 milisse-



(a) Tempo de resposta das requisições utilizando o banco de dados PostgreSQL



(b) Tempo de resposta das requisições utilizando o banco de dados DB2

Figura 29 – Comparação da dinâmica entre Postgres e DB2

Fonte: Mamani (2015).

gundos aproximadamente. Isto indica que a versão do PostgreSQL usada não apresenta um comportamento adaptativo para mudanças abruptas.

Na Figura 29b mostra o comportamento do tempo de resposta do DB2. Aplicando a mesma carga de trabalho do experimento no PostgreSQL com o mesmo nível de carga e durante o mesmo tempo, podemos observar um primeiro nível estacionário em torno de 87 milissegundos passando para um novo patamar em torno de 142 milissegundos.

Como afirmado por Mamani (2015), este experimento evidencia a utilidade de um *benchmark* em regime não-estacionário, permitindo o desenvolvimento de um modulo adaptativo

na camada de aplicação. Principalmente este é um caso comum na Computação em Nuvem, onde existem diversos comportamentos dinâmicos.

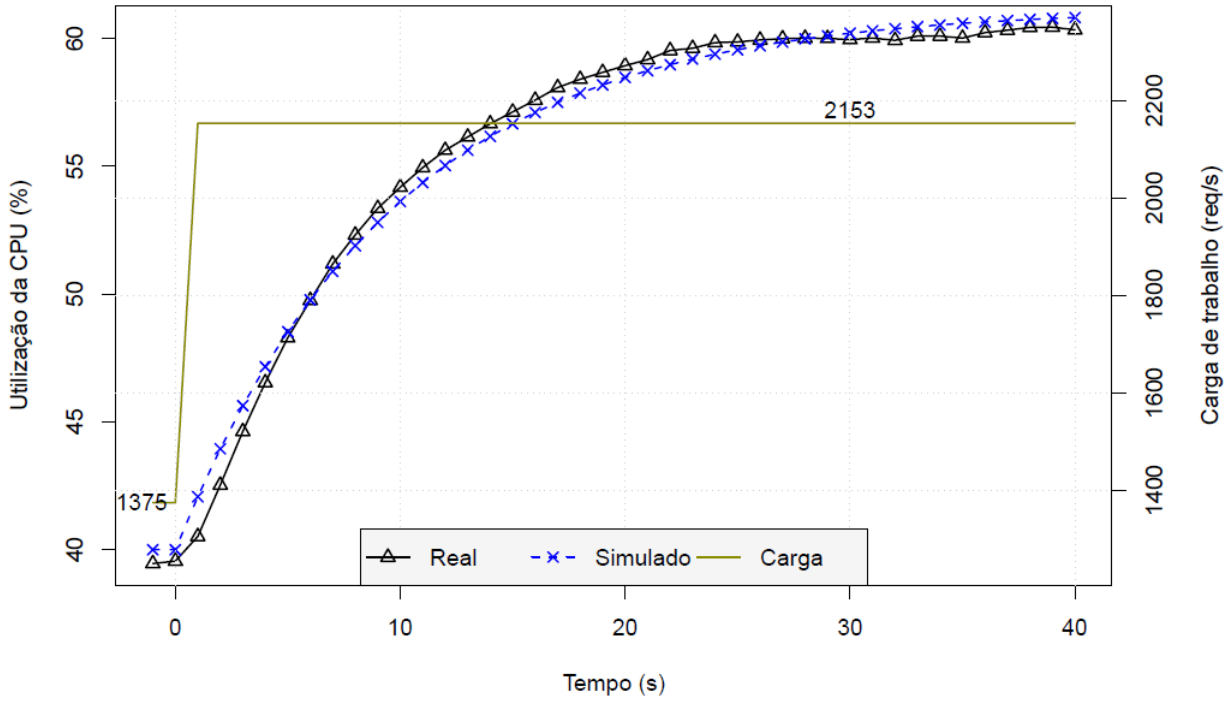


Figura 30 – Utilização da metodologia de avaliação de desempenho não-estacionário

Fonte: Mamani (2015).

O resultado do experimento também apresentado por Mamani (2015) demostra a aplicação real da extensão do Bencha4Q em um ambiente típico de Computação em Nuvem, o que foi possível medir o desempenho que varia ao longo do tempo, utilizando a metodologia de *benchmarking* de desempenho não-estacionário proposto por Mamani (2015).

A Figura 30 apresenta as taxas de utilização do sistema real com base nos valores da entrada (carga de trabalho), coletadas em requisições por segundo, juntamente com a saída simulada do modelo extraído no sistema real. A carga de trabalho utilizada no experimento é do tipo degrau, no instante inicial a execução do experimento, a carga está próximo a 1376 requisições por segundo, no instante zero, quando inicia o degrau da carga, o número de requisições por segundo aumenta de maneira brusca chegando a 2153 em 1 segundo, um aumento superior a 50% da carga inicial.

A métrica de desempenho a essa carga é a taxa de utilização da CPU, em preto no gráfico da Figura 30. Inicialmente a taxa de utilização está abaixo de 40% antes do instante zero da execução do experimento. Instantes seguintes após a perturbação da carga a taxa de utilização apresenta um comportamento transiente chegando a ultrapassar os 80%.

$$G(z) = \frac{b}{z-a} \tag{5.1}$$

$$G(z) = \frac{0.002682}{z - 0.9057} \tag{5.2}$$

A partir dos dados produzidos no sistema real, é possível fazer uma identificação do sistema atrás de técnicas de modelagem de sistemas. Estima-se os parâmetros a e b para o modelo ARX da equação 5.1 e derivar uma função de transferência equivalente G. A função transferência é uma aproximação analítica empírica da dinâmica do sistema que pode ser validada por comparação com a saída do sistema (simulação) conforme observado na Figura 30 pela linha em azul. As saídas simulada mostraram-se bem aproximadas e representam bem o desempenho do sistema.

$$y(k) = -ay(k-1) + bu(k-1) \tag{5.3}$$

$$y(k) = -0.9057y(k-1) + 0.002682u(k-1) \tag{5.4}$$

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

CAPÍTULO

6

CONCLUSÃO

A computação em nuvem popularizou um serviço de comercialização se de capacidade computacional na qual a infraestrutura, plataforma ou *software* são ofertados como produto sob demanda e onde os recursos são elásticos, despertando o interesse tanto da comunidade acadêmica quanto da indústria. Atualmente a maioria das grandes soluções de sistemas computacionais são compostas por *mult-tiers*, inclusive quando se refere a aplicações web, devido à flexibilidade de escalabilidade. Para essas aplicações, o planejamento de capacidade é um requisito crítico para determinar a quantidade de recursos exigido para garantia de QoS. No entanto, o planejamento de capacidade é usualmente uma decisão de longo prazo em que, e os recursos são determinados por critérios estáticos. Desta forma, os recursos podem revelar uma sobrecarga em situações de perturbação, mesmo que os níveis de QoS esteja dentro da faixa aceitável para a carga estacionária.

Em sistemas que apresentam dinâmica acentuada, a avaliação de desempenho deve considerar que os períodos de regime transiente são importantes. Junto aos mecanismos de elasticidade dos recursos sob demanda, vem a necessidade do autogerenciamento dos recursos. Existem vários trabalhos disponíveis na literatura que lidam e tratam da gestão dos recursos computacionais. Neste trabalho há interesse na modelagem do sistema na forma de uma representação analítica capaz de reproduzir o comportamento dinâmico do sistema, onde o período transiente tem grande colaboração e impacto na política de gerenciamento dos recursos. Um diferencial em relação as abordagens convencionais o objetivo de determinar como a capacidade do sistema em lidar com a variação da carga de trabalho, ao invés de, o desempenho apenas com a carga de trabalho estacionaria.

Mediante a uma arquitetura conceitual proposta por Júnior *et al.* (2015b) e Mamani (2015) propõe uma metodologia que descreve e especifica os passos para modelar um sistema computacional por meio de um *benchmark*. No entanto, não foram encontrados *benchmarks* que estimulem a dinâmica do sistema e que permitem uma avaliação em regime transiente, bem

como não foram identificados *benchmarks* que sigam a especificação de requisitos proposta por Júnior *et al.* (2015b). Este trabalho apresentou uma extensão de um *benchmark*, o Bench4Q, capaz de modelar a sua carga de trabalho de tal maneira a estimular o sistema a apresentar sua dinâmica através da carga. Essa extensão segue um dos requisitos MESC, o modulo *Demand*, proposto por Júnior *et al.* (2015b), que se restringe a modulação da carga de trabalho, acrescendo-a de provisões para gerar perturbações programadas. Essa extensão, obedece ao padrão de implementação e usabilidade nativas do *benchmark* Bench4Q. A extensão é provida de uma interface gráfica que possibilita a modelagem da carga através da inserção de parâmetros.

Para atingir o objetivo proposto, foi necessária a alteração da carga de trabalho nativa do Bench4Q. Essa modificação resultou na modulação da carga, possibilitando a geração de Degrau Positivo com a carga. Este modelo de carga tem como característica a alteração da sua potência de maneira brusca e repentina. Também é possível gerar um Degrau Negativo que tem efeito oposto ao Degrau Positivo, ou seja, o modelo da carga tem por característica a queda repentina de sua potência. Outra modelagem de carga que a extensão permite é a geração de uma Onda Quadrada, onde existe uma alternância entre os dois modelos descritos anteriormente. Por se tratar de um trabalho de extensão de *benchmark* difundido e grande complexidade, as alterações efetuadas trouxeram dificuldades relacionadas a implementação. A falta de uma documentação técnica gerou grande esforço no entendimento e compreensão da implementação original, necessitando muitas vezes a depuração do código para o claro entendimento do seu fluxo de funcionamento tão quando os módulos envolvidos.

O projeto apresentou exemplos práticos das modelagem das cargas propostas (Degrau Positivo, Degrau Negativo e Onda Quadrada) através dos resultados gerados pelo próprio *benchmark*. Os resultados da presente pesquisa foram adequado, na medida em que responderam positivamente ao objetivo do trabalho. O trabalho contemplou uma bateria de experimentos práticos em um ambiente controlado *mult-tier*. Na execução das fases de experimentos, foi elaborado o planejamento do experimento, a coleta de dados e juntamente a análises dos resultados obtidos, resultando em um conjunto de contribuições para a área de pesquisa:

- A elaboração de uma documentação padronizada da mesma forma que a original do Bench4Q, contando com versão em inglês que se encontra no apêndice B;
- Impacto da carga modulada: mediante a modulação da carga através da extensão, é possível excitar o sistema a apresentar a sua dinâmica, contribuindo para trabalhos que tem por necessidade a modelagem do sistema e do seu comportamento dinâmico;
- Dinâmica entre camadas: ao se tratar de um sistema de multicamadas, foi possível perceber, juntamente com a modulação da carga, a dinâmica intrínseca entre as camadas do sistema. Os efeitos combinados de atrasos intrínsecos, ainda que pequenos, e sua propagação por todo as camadas interligados geraram um comportamento dinâmico significante e apreciável;

- Métrica que mascara: por consequência da dinâmica inerente ao sistema multicamadas, foi possível vislumbrar que nem toda métrica apresenta a realidade quando se lida com um sistema multicamadas. Neste trabalho, foi possível observar esse fato na métrica tempo de resposta.

O presente trabalho foi desenvolvido em consequência aos interesse de estudo da pesquisa de Mamani (2015), e em paralelo a outra iniciativa de Júnior *et al.* (2015b) que especificam a identificação de capacidade dinâmica em sistemas computacionais e como tratá-las com técnicas de modelagem de sistemas. Os resultados do presente trabalho contribuem para ambos os projetos com a disponibilização de um *benchmark* que auxilia na modelagem de sistemas computacionais dinâmicos.

O presente trabalho implementa um dos requisitos do modelo conceitual de referência MEDC. A extensão permite que um *benchmark*, focado na avaliação de desempenho estacionária, seja capaz de uma avaliação não-estacionaria sendo está a principal contribuição do trabalho.

6.1 Trabalhos Futuros

O presente trabalho de mestrado contribuiu para o desenvolvimento de técnicas e estudos interessados na dinâmica do sistema. Todavia, existe uma gama de trilhas a serem exploradas até que a importância da dinâmica de sistemas computacionais tenha maior apreço, como nos casos das ciências e engenharias. Consequentemente este trabalho não finaliza as possibilidades de estudo relacionadas e outros estudos podem ser desenvolvidos a partir dos resultados e constatações identificadas dentre os são:

- **Novas formas de pertubação:** com base nas proposta de (HELLERSTEIN *et al.*, 2004) existem outras funções que auxiliam a excitar o sistema a apresentarem a sua dinâmica.

- **Avaliação da dinâmica em sistemas multicamadas:** com o presente trabalho, foi possível observar a dinâmica entre as camadas do sistema, entretanto o presente trabalho não cobre com um planejamento de experimentos utilizando métodos estatísticos por meio de avaliações de desempenho

- **Avaliação de métricas de planejamento de recursos em ambiente multicamadas:** este trabalho também revelou que existem métricas que podem mascarar a realidade, neste contexto é importante uma análise das principais métricas de diferentes técnicas para o planejamento de recursos sob um ambiente de multicamadas

REFERÊNCIAS

BINNIG, C.; KOSSMANN, D.; KRASKA, T.; LOESING, S. How is the weather tomorrow? towards a benchmark for the cloud. In: **Proceedings of the Second International Workshop on Testing Database Systems**. New York, NY, USA: ACM, 2009. (DBTest '09), p. 9:1–9:6. ISBN 978-1-60558-706-6. Disponível em: <http://doi.acm.org/10.1145/1594156.1594168>. Citado 4 vezes nas páginas 1, 5, 28 e 29.

CERVINO, J.; KALYVIANAKI, E.; SALVACHUA, J.; PIETZUCH, P. Adaptive provisioning of stream processing systems in the cloud. In: **Data Engineering Workshops (ICDEW), 2012 IEEE 28th International Conference on**. [S.l.: s.n.], 2012. p. 295–301. Citado na página 2.

CHEN, Y.; RAAB, F.; KATZ, R. From tpc-c to big data benchmarks: A functional workload model. In: RABL, T.; POESS, M.; BARU, C.; JACOBSEN, H.-A. (Ed.). **Specifying Big Data Benchmarks**. Springer Berlin Heidelberg, 2014, (Lecture Notes in Computer Science, v. 8163). p. 28–43. ISBN 978-3-642-53973-2. Disponível em: <http://dx.doi.org/10.1007/978-3-642-53974-9_4>. Citado 2 vezes nas páginas 14 e 16.

CHERKASOVA, L.; PHAAL, P. **Session based admission control: a mechanism for improving the performance of an overloaded web server**. [S.l.]: Hewlett-Packard Laboratories, 1998. Citado na página 16.

DEAN, T. L.; WELLMAN, M. P. **Planning and Control**. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1991. ISBN 1-55860-209-7. Citado na página 9.

E-commerce Brasil. **Tempo de resposta em sites de e-commerce aumenta em mais de 300% no Black Friday 2012**. 2012. Https://www.ecommercebrasil.com.br/noticias/tempo-de-resposta-em-sites-de-e-commerce-aumenta-em-mais-de-300-no-black-friday-2012/. Citado na página 3.

FOLKERTS, E.; ALEXANDROV, A.; SACHS, K.; IOSUP, A.; MARKL, V.; TOSUN, C. Benchmarking in the cloud: What it should, can, and cannot be. In: NAMBIAR, R.; POESS, M. (Ed.). **Selected Topics in Performance Evaluation and Benchmarking**. Springer Berlin Heidelberg, 2013, (Lecture Notes in Computer Science, v. 7755). p. 173–188. ISBN 978-3-642-36726-7. Disponível em: <http://dx.doi.org/10.1007/978-3-642-36727-4_12>. Citado 3 vezes nas páginas 14, 16 e 28.

GALANTE, G.; BONA, L. C. E. de. A survey on cloud computing elasticity. In: IEEE. **Utility and Cloud Computing (UCC), 2012 IEEE Fifth International Conference on**. [S.l.], 2012. p. 263–270. Citado na página 7.

GRAY, J. **Benchmark Handbook: For Database and Transaction Processing Systems**. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1992. ISBN 1558601597. Citado na página 14.

HEILIG, L.; VOSS, S. A scientometric analysis of cloud computing literature. **Cloud Computing, IEEE Transactions on**, v. 2, n. 3, p. 266–278, July 2014. ISSN 2168-7161. Citado na página 1.

HELLERSTEIN, J. L.; DIAO, Y.; PAREKH, S.; TILBURY, D. M. **Feedback Control of Computing Systems**. 1. ed. Hoboken, NJ, USA: John Wiley & Sons, Inc., 2004. 429 p. ISBN 047126637X. Citado 4 vezes nas páginas 13, 14, 25 e 53.

HUANG, D.; HE, B.; MIAO, C. A survey of resource management in multi-tier web applications. **Communications Surveys Tutorials, IEEE**, v. 16, n. 3, p. 1574–1590, Third 2014. ISSN 1553-877X. Citado 2 vezes nas páginas 3 e 5.

HUPPLER, K. Performance evaluation and benchmarking. In: NAMBIAR, R.; POESS, M. (Ed.). Berlin, Heidelberg: Springer-Verlag, 2009. cap. The Art of Building a Good Benchmark, p. 18–30. ISBN 978-3-642-10423-7. Disponível em: <http://dx.doi.org/10.1007/978-3-642-10424-4_3>. Citado 2 vezes nas páginas 14 e 16.

INOMATA, A.; MORIKAWA, T.; IKEBE, M.; OKAMOTO, Y.; NOGUCHI, S.; FUJIKAWA, K.; SUNAHARA, H.; RAHMAN, S. M. M. Proposal and Evaluation of a Dynamic Resource Allocation Method Based on the Load of VMs on IaaS. In: **2011 4th IFIP International Conference on New Technologies, Mobility and Security**. [S.l.]: IEEE, 2011. p. 1–6. ISBN 978-1-4244-8705-9. Citado na página 2.

JANERT, P. K. **Feedback Control for Computer Systems**. [S.l.]: O'Reilly Media, Inc., 2013. ISBN 1449361692, 9781449361693. Citado 3 vezes nas páginas 9, 10 e 11.

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: **Proceedings of the 2015 Winter Simulation Conference**. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc., 2015. p. (*to appear in*). Citado na página 4.

____. Non-stationary simulation of computer systems and dynamic performance evaluation: a concern-based approach and case study on cloud computing. In: **Computer Architecture and High Performance Computing (SBAC-PAD), 2015 IEEE 27th International Symposium on**. [S.l.: s.n.], 2015. p. (to be printed). ISSN 1550-6533. Citado 11 vezes nas páginas 3, 4, 5, 10, 11, 12, 14, 31, 51, 52 e 53.

KISTOWSKI, J. v.; ARNOLD, J. A.; HUPPLER, K.; LANGE, K.; HENNING, J. L.; CAO, P. How to build a benchmark. In: JOHN, L. K.; SMITH, C. U.; SACHS, K.; LLADÓ, C. M. (Ed.). **Proceedings of the 6th ACM/SPEC International Conference on Performance Engineering, Austin, TX, USA, January 31 - February 4, 2015**. ACM, 2015. p. 333–336. Disponível em: <http://doi.acm.org/10.1145/2668930.2688819>. Citado 2 vezes nas páginas 14 e 15.

KRISHNAMURTHY, D.; ROLIA, J.; MAJUMDAR, S. A synthetic workload generation technique for stress testing session-based systems. **Software Engineering, IEEE Transactions on**, v. 32, n. 11, p. 868–882, Nov 2006. ISSN 0098-5589. Citado na página 16.

LORIDO-BOTRÁN, T.; MIGUEL-ALONSO, J.; LOZANO, J. A. **Auto-scaling Techniques for Elastic Applications in Cloud Environments**. [S.l.], 2012. Citado na página 7.

LU, C.; STANKOVIC, J. A.; ABDELZAHER, T. F.; TAO, G.; SON, S. H.; MARLEY, M. Performance specifications and metrics for adaptive real-time systems. In: **Proceedings of the 21st IEEE Conference on Real-time Systems Symposium**. Washington, DC, USA: IEEE Computer Society, 2000. (RTSS'10), p. 13–23. ISBN 0-7695-0900-2. Disponível em: <http://dl.acm.org/citation.cfm?id=1890629.1890632>. Citado na página 29.

LUZ, H. J. F. d. **Um método para previsão de sobrecarga transiente em sistemas computacionais por meio de modelos dinâmicos obtidos empiricamente**. Dissertação (Mestrado) — Instituto de Ciências Matemáticas e de Computação, University of São Paulo, 2014. Citado 3 vezes nas páginas 15, 11 e 29.

MAMANI, E.; PEREIRA, L.; SANTANA, M.; SANTANA, R.; NOBILE, P.; MONACO, F. Transient performance evaluation of cloud computing applications and dynamic resource control in large-scale distributed systems. In: **High Performance Computing Simulation (HPCS), 2015 International Conference on**. [S.l.: s.n.], 2015. p. 246–253. Citado na página 4.

MAMANI, E. L. C. **Metodologia de benchmark para avaliação de desempenho não-estacionária: um estudo de caso baseado em aplicações de computação em nuvem**. Tese (Doutorado) — Universidade de São Paulo, 2015. Citado 11 vezes nas páginas 4, 5, 13, 26, 31, 44, 47, 48, 49, 51 e 53.

MARK, K.; CSABA, L. Analyzing customer behavior model graph (cbmg) using markov chains. In: **Intelligent Engineering Systems, 2007. INES 2007. 11th International Conference on**. [S.l.: s.n.], 2007. p. 71–76. Citado na página 20.

MENASCE, D. TPC-W: a benchmark for e-commerce. **IEEE Internet Computing**, v. 6, n. 3, p. 83–87, maio 2002. ISSN 1089-7801. Citado 2 vezes nas páginas 16 e 21.

NOBILE, P. N. **Projeto de um Broker de Gerenciamento Adaptativo de Recursos em Computação em Nuvem Baseado em Técnicas de Controle Realimentado**. Tese (Doutorado) — Instituto de Ciências Matemáticas e de Computação (ICMC), Universidade de São Paulo - USP, 2013. Citado 6 vezes nas páginas 2, 4, 7, 8, 30 e 38.

NOBILE, P. N.; LOPES, R. R. F.; MORON, C. E.; TREVELIN, L. C. QoS Proxy Architecture for Real Time RPC with Traffic Prediction. In: **11th IEEE International Symposium on Distributed Simulation and Real-Time Applications (DS-RT'07)**. [S.l.]: IEEE, 2007. p. 261–267. ISBN 0-7695-3011-7. Citado na página 31.

OGATA, K. **Modern Control Engineering**. 4th. ed. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2001. ISBN 0130609072. Citado na página 3.

OW2; TRUSTIE. **Bench4Q: A QoS-oriented E-Commerce Benchmark Tool**. [S.l.]: http://www.ow2.org/.../20100925-Bench4Q-1.3.0-ISCAS.pdf, 2010. Citado na página 20.

PADALA, P.; SHIN, K. G.; ZHU, X.; UYSAL, M.; WANG, Z.; SINGHAL, S.; MERCHANT, A.; SALEM, K. Adaptive control of virtualized resources in utility computing environments. **SIGOPS Oper. Syst. Rev.**, ACM, New York, NY, USA, v. 41, n. 3, p. 289–302, mar. 2007. ISSN 0163-5980. Disponível em: <http://doi.acm.org/10.1145/1272998.1273026>. Citado na página 2.

QUIROZ, A.; KIM, H.; PARASHAR, M.; GNANASAMBANDAM, N.; SHARMA, N. Towards autonomic workload provisioning for enterprise Grids and clouds. In: **Grid Computing, 2009 10th IEEE/ACM International Conference on**. [S.l.: s.n.], 2009. p. 50–57. Citado na página 2.

ROSU, D.; SCHWAN, K.; YALAMANCHILI, S.; JHA, R. On adaptive resource allocation for complex real-time applications. In: **Real-Time Systems Symposium, 1997. Proceedings., The 18th IEEE**. [S.l.: s.n.], 1997. p. 320–329. ISSN 1052-8725. Citado 2 vezes nas páginas 28 e 29.

VAZQUEZ, C.; KRISHNAN, R.; JOHN, E. Cloud computing benchmarking: A survey. In: THE STEERING COMMITTEE OF THE WORLD CONGRESS IN COMPUTER SCIENCE, COMPUTER ENGINEERING AND APPLIED COMPUTING (WORLDCOMP). **Proceedings of the International Conference on Grid Computing and Applications (GCA)**. [S.l.], 2014. p. 1. Citado 2 vezes nas páginas 1 e 7.

VIEIRA, M.; MADEIRA, H. A dependability benchmark for {OLTP} application environments. In: FREYTAG, J.-C.; LOCKEMANN, P.; ABITEBOUL, S.; CAREY, M.; SELINGER, P.; HEUER, A. (Ed.). **Proceedings 2003 {VLDB} Conference**. San Francisco: Morgan Kaufmann, 2003. p. 742 – 753. ISBN 978-0-12-722442-8. Disponível em: <http://www.sciencedirect.com/science/article/pii/B9780127224428500719>. Citado na página 32.

VIEIRA, M.; MADEIRA, H.; SACHS, K.; KOUNEV, S. Resilience benchmarking. In: **Resilience Assessment and Evaluation of Computing Systems**. [s.n.], 2012. p. 283–301. Disponível em: <http://dx.doi.org/10.1007/978-3-642-29032-9_14>. Citado 4 vezes nas páginas 14, 15, 16 e 28.

WANG, L.; XU, J.; ZHAO, M.; TU, Y.; FORTES, J. **Autonomic Resource Management for Virtualized Database Hosting Systems**. [S.l.], 2009. Citado na página 30.

WEBER, A. **Resource Elasticity Benchmarking in Cloud Environments**. Dissertação (Master Thesis) — Karlsruhe Institute of Technology (KIT), Am Fasanengarten 5, 76131 Karlsruhe, Germany, August 2014. Citado na página 16.

YANG, H.; TATE, M. A descriptive literature review and classification of cloud computing research. **Communications of the Association for Information Systems**, v. 31, n. 2, p. 35–60, 2012. Citado na página 1.

ZHANG, J.; KIM, J.; YOUSIF, M.; CARPENTER, R.; FIGUEIREDO, R. J. System-level performance phase characterization for on-demand resource provisioning. **2007 IEEE International Conference on Cluster Computing**, IEEE, Los Alamitos, CA, USA, v. 0, p. 434–439, 2007. Citado na página 2.

ZHANG, W.; WANG, S.; WANG, W.; ZHONG, H. Bench4q: A qos-oriented e-commerce benchmark. In: **COMPSAC**. [S.l.]: IEEE Computer Society, 2011. p. 38–47. ISBN 978-0-7695-4439-7. Citado 7 vezes nas páginas 16, 17, 18, 19, 21, 22 e 23.

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**. [S.l.]: IEEE, 2011. p. 266–273. ISBN 978-1-4577-1637-9. Citado 5 vezes nas páginas 3, 18, 19, 20 e 21.

APÊNDICE

A

CÓDIGOS DA EXTENSÃO

Utilizado para descrever e compreender melhor a extensão do *benchmark* na forma de um algoritmo, ou código, este apêndice apresenta na integra trechos dos principais códigos da extensão que permite ao usuário obter uma ideia de como o *benchmark* foi implementado.

A.1 Configuração da carga de trabalho

Código-fonte 4: Algoritmo calcula os tempos de inicialização e termino para cada um dos Clientes

```
1  public static PropertiesEB createProperties(int index, TestPhase
     testPhase, TypeFrequency type, long beginTime) {
2
3    PropertiesEB propertiesEB = new PropertiesEB();
4    Logger.getLogger().debug(type.getName());
5    propertiesEB.isFrenquency = true;
6    propertiesEB.setIndexEB(index);
7
8    long timeStart = testPhase.getFrequency().getStartTime() * 1000 +
     beginTime;
9    long timeEnd   = testPhase.getFrequency().getEndTime() * 1000 +
     timeStart;
10   long timePause = testPhase.getFrequency().getPauseTime() * 1000;
11   long timeExperiment = testPhase.getExperimentTime() * 1000 +
     beginTime;
12
13   if (TypeFrequency.STEP.equals(type)) {
14     if (index >= testPhase.getFrequency().getQuantity()) {
15       propertiesEB.setStartTime(beginTime);
16       propertiesEB.setEndTime(timeExperiment);
17       propertiesEB.setEndExperimentTime(timeExperiment);
```

```
18          Logger.getLogger().debug("Normal: " + index);
19        } else {
20        propertiesEB.setStartTime(timeStart);
21        propertiesEB.setEndTime(timeEnd);
22        propertiesEB.setEndExperimentTime(timeExperiment);
23        Logger.getLogger().debug("To Step: " + index);
24      }
25    } else if (TypeFrequency.PULSE.equals(type)) {
26    if (index >= testPhase.getFrequency().getQuantity()) {
27      propertiesEB.setStartTime(beginTime);
28      propertiesEB.setEndTime(timeExperiment);
29      propertiesEB.setEndExperimentTime(timeExperiment);
30      Logger.getLogger().debug("Normal: " + index);
31    } else {
32    propertiesEB.setStartTime(timeStart);
33    propertiesEB.setPauseTime(timePause);
34    propertiesEB.setEndTime(timeEnd);
35    propertiesEB.setEndExperimentTime(timeExperiment);
36    Logger.getLogger().debug("To Pulse: " + index);
37  }
38  }
39
40  return propertiesEB;
41
42  }
```

A.2 Geração da carga de trabalho

Código-fonte 5: Algoritmo de geração de carga modificado para modulação

```
1   public void test() {
2     long tt = 0L; // Think Time.
3     boolean sign = true;
4     long startGet = System.currentTimeMillis();
5     long currentTimeMillis = System.currentTimeMillis();
6     this.sessionStart = startGet;
7
8
9     while ((this.maxTrans == -1) || (this.maxTrans > 0)) {
10
11      //avaliando o EB segundo o tempo percorrido do experimento
12      currentTimeMillis = System.currentTimeMillis();
13
14      if (currentTimeMillis > this.propertiesEB.getEndExperimentTime
      ()){
```

```
15        Logger.getLogger().debug(propertiesEB.getIndexEB() + " is
    stopping ... " + (currentTimeMillis - startExp)/1000);
16        this.test = false;
17      }
18
19      if (currentTimeMillis > this.propertiesEB.getEndTime() && this.
    propertiesEB.isFrenquency()) {
20        //desactivado = -1
21        if(this.propertiesEB.getPauseTime() > 0){
22          long newInit = this.propertiesEB.getEndTime() + this.
    propertiesEB.getPauseTime();
23          long period = this.propertiesEB.getEndTime() - this.
    propertiesEB.getStartTime();
24
25          this.propertiesEB.setStartTime(newInit);
26          this.propertiesEB.setEndTime(period + newInit);
27
28          Logger.getLogger().debug(propertiesEB.getIndexEB() + " was
    restarted  ... ");
29        }else{
30        Logger.getLogger().debug(propertiesEB.getIndexEB() + " is
    ending ... " + (currentTimeMillis - startExp)/1000);
31        this.test = false;
32      }
33    }
34
35    // alguns EBs nao iniciam inmediatamente, porque foram marcados
    para esperar
36    if (currentTimeMillis >= this.propertiesEB.getStartTime()) {
37      if (this.terminate || !this.test) {
38        this.sessionEnd = System.currentTimeMillis();
39        EBStats.getEBStats().sessionRecorder(this.sessionStart, this.
    sessionEnd, this.sessionLen,
40        this.Ordered, this.isVIP);
41        return;
42      }
43
44      long endGet;
45      if (this.nextReq != null) {
46        // Check if user session is finished.
47        if (this.toHome) {
48          // User session is complete. Start new user session.
49          this.sessionEnd = System.currentTimeMillis();
50          EBStats.getEBStats().sessionRecorder(this.sessionStart,
    this.sessionEnd, this.sessionLen,
51          this.Ordered, this.isVIP);
52          initialize();
```

```
53            return;
54          }
55          if (this.nextReq.equals("")) {
56            EBStats.getEBStats().addErrorSession(this.curState, this.
      isVIP);
57            initialize();
58            continue;
59          }
60          // Receive HTML response page.
61          if (this.rate > 0) {
62            if (isVIP) {
63              if (this.nextReq.contains("?")) {
64                this.nextReq += "&bench4q_session_priority=10";
65              } else {
66              this.nextReq += "?bench4q_session_priority=10";
67            }
68          } else if (this.nextReq.contains("?")) {
69          this.nextReq += "&bench4q_session_priority=1";
70        } else {
71        this.nextReq += "?bench4q_session_priority=1";
72      }
73    }
74
75    // additional load
76    if(this.addLoad > 0 && this.addLoadOpt >= 0) {
77      if (this.nextReq.contains("?")) {
78        this.nextReq += "&bench4q_add_load=" + this.addLoad + "&
      bench4q_add_load_opt=" +this.addLoadOpt;
79      } else {
80      this.nextReq += "?bench4q_add_load=" + this.addLoad + "&
      bench4q_add_load_opt=" +this.addLoadOpt;
81    }
82  } else {
83  if (this.nextReq.contains("?")) {
84    this.nextReq += "&bench4q_add_load=0&bench4q_add_load_opt=0";
85  } else {
86  this.nextReq += "?bench4q_add_load=0&bench4q_add_load_opt=0";
87  }
88  }
89
90  if (this.first) {
91    this.m_Client = HttpClientFactory.getInstance();
92    this.m_Client.getParams().setCookiePolicy(CookiePolicy.RFC_2965);
93  }
94
95  startGet = System.currentTimeMillis();
```

```
96 sign = getHTML(this.curState, this.nextReq, (currentTimeMillis -
      startExp)/1000);
97
98 endGet = System.currentTimeMillis();
99
100 if (!sign) {
101   EBStats.getEBStats().addErrorSession(this.curState, this.isVIP);
102   initialize();
103
104   continue;
105 }
106 this.first = false;
107
108 // Compute and store Web Interaction Response Time (WIRT)
109 EBStats.getEBStats().interaction(this.curState, startGet, endGet, tt,
       this.isVIP);
110 this.sessionLen++;
111 if (this.curState == 4) {
112   this.Ordered = true;
113 }
114 this.curTrans.postProcess(this, this.html);
115 } else {
116 this.html = null;
117 endGet = startGet;
118 }
119
120 if (!nextState()) {
121   return;
122 }
123 if (this.nextReq != null) {
124   // Pick think time (TT), and compute absolute request time
125   tt = MAP();
126   startGet = endGet + tt;
127   if ((this.terminate) || (!this.test)) {
128     return;
129   }
130   try {
131     sleep(tt);
132   } catch (InterruptedException inte) {
133   Thread.currentThread().interrupt();
134   return;
135 }
136 if (this.maxTrans > 0) {
137   this.maxTrans--;
138 }
139 } else {
140 EBStats.getEBStats().addErrorSession(this.curState, this.isVIP);
```

```
141 initialize();
142 }
143 } else {
144 try {
145   // libera de sobrecarga
146   Thread.sleep(500L);
147 } catch (InterruptedException e) {
148 // TODO Auto-generated catch block
149 e.printStackTrace();
150 }
151 }
152
153 }
154 }
```

A.3 *Interface* gráfica

Código-fonte 6: Código para gerar a os parâmetros para a modulação

```
1   private void createPanelFunction(final TypeFrequency type) {
2
3     this.functionPanel.removeAll();
4     this.m_configModel.getArgs().setTypeFrenquency(type.getName());
5     int row = 0;
6
7     lb_startTime = new JLabel("Start Time");
8     tf_startTime = new JTextField(String.valueOf(dataSet.get(0).
    getFrequency().getStartTime()));
9     tf_startTime.getDocument().addDocumentListener(new
    StartTimeListener());
10    functionPanel.add(lb_startTime, new GridBagConstraints(0, row, 1,
     1, 0.0, 0.0, GridBagConstraints.EAST,
11    GridBagConstraints.NONE, new Insets(5, 5, 5, 5), 1, 1));
12    functionPanel.add(tf_startTime, new GridBagConstraints(1, row++,
    1, 1, 100.0, 0.0, GridBagConstraints.WEST,
13    GridBagConstraints.HORIZONTAL, new Insets(5, 5, 5, 5), 1, 1));
14
15    lb_endTime = new JLabel("Duration Step");
16    tf_endTime = new JTextField(String.valueOf(dataSet.get(0).
    getFrequency().getEndTime()));
17    tf_endTime.getDocument().addDocumentListener(new EndTimeListener
    ());
18    functionPanel.add(lb_endTime, new GridBagConstraints(0, row, 1,
    1, 0.0, 0.0, GridBagConstraints.EAST,
19    GridBagConstraints.NONE, new Insets(5, 5, 5, 5), 1, 1));
```

```
20     functionPanel.add(tf_endTime, new GridBagConstraints(1, row++, 1,
       1, 100.0, 0.0, GridBagConstraints.WEST,
21     GridBagConstraints.HORIZONTAL, new Insets(5, 5, 5, 5), 1, 1));
22
23     if (type.getName().compareTo("Pulse") == 0) {
24       lb_pauseTime = new JLabel("Pause");
25       tf_pauseTime = new JTextField(String.valueOf(dataSet.get(0).
       getFrequency().getPauseTime()));
26       tf_pauseTime.getDocument().addDocumentListener(new
       PauseTimeListener());
27       functionPanel.add(lb_pauseTime, new GridBagConstraints(0, row,
       1, 1, 0.0, 0.0, GridBagConstraints.EAST,
28       GridBagConstraints.NONE, new Insets(5, 5, 5, 5), 1, 1));
29       functionPanel.add(tf_pauseTime, new GridBagConstraints(1, row
       ++, 1, 1, 100.0, 0.0, GridBagConstraints.WEST,
30       GridBagConstraints.HORIZONTAL, new Insets(5, 5, 5, 5), 1, 1));
31     }
32
33     if (type.getName().compareTo("Step") == 0) {
34       lb_polarity = new JLabel("Polarity");
35       tf_polarity = new JTextField(String.valueOf(dataSet.get(0).
       getFrequency().getPolarity()));
36       tf_polarity.getDocument().addDocumentListener(new
       PolarityListener());
37       functionPanel.add(lb_polarity, new GridBagConstraints(0, row,
       1, 1, 0.0, 0.0, GridBagConstraints.EAST,
38       GridBagConstraints.NONE, new Insets(5, 5, 5, 5), 1, 1));
39       functionPanel.add(tf_polarity, new GridBagConstraints(1, row++,
        1, 1, 100.0, 0.0, GridBagConstraints.WEST,
40       GridBagConstraints.HORIZONTAL, new Insets(5, 5, 5, 5), 1, 1));
41     }
42
43     lb_quantity = new JLabel("Quantity");
44     tf_quantity = new JTextField(String.valueOf(dataSet.get(0).
       getFrequency().getQuantity()));
45     tf_quantity.getDocument().addDocumentListener(new
       QuantityListener());
46     functionPanel.add(lb_quantity, new GridBagConstraints(0, row, 1,
       1, 0.0, 0.0, GridBagConstraints.EAST,
47     GridBagConstraints.NONE, new Insets(5, 5, 5, 5), 1, 1));
48     functionPanel.add(tf_quantity, new GridBagConstraints(1, row++,
       1, 1, 100.0, 0.0, GridBagConstraints.WEST,
49     GridBagConstraints.HORIZONTAL, new Insets(5, 5, 5, 5), 1, 1));
50
51     this.functionPanel.updateUI();
52     this.functionPanel.repaint();
53
```

```
54 }
```

APÊNDICE

B

DOCUMENTAÇÃO DA EXTENSÃO DO BENCH4Q

Este Apêndice traz a documentação produzida para o usuário afim de facilitar o entendimento e uso da extensão realizada no *benchmark*, aderindo às características da documentação original do Bench4Q.

Bench4Q Tool 1.3 – OnlineBench4Q

Workload Model

USER'S MANUAL

Revision Sheet

Release No.	Date	Revision Description
Rev. 1.0	09/24/2009	For Bench4Q Tool 1.0.0
Rev. 1.1	11/26/2009	For Bench4Q Tool 1.1.0
Rev. 1.2	05/10/2010	For Bench4Q Tool 1.2.X
Rev. 1.3	07/11/2010	For Server-side Resourcer Monitoring
Rev. 1.3 – OnlineBench4Q	07/11/2015	For Workload Model

1 Introduction

1.1 Basic information

The Bench4Q, is a benchmarking using an e-commerce oriented QoS, provided features that allow the simulation of a controllable and flexible environment. Furthermore, the Bench4Q can be used to evaluate system performance scalability.

The Bench4Q is an extension of TPC-W, and aims to tuning servers ecommerce oriented to provide QoS to their customers. The main features of Bench4Q include: supporting the analysis of session-based metrics that simulates sensitive cargo QoS for a capacity analysis.

The Bench4Q benchmark, is distributed under the Lesser General Public License (GNU), and free software, it can be redistributed and / or modified under the terms of public license by the Free Software Foundation. Following many directives of the TPC-W specification, Bench4Q mainly uses in his simulation metrics QoS guarantee

USP by the need of some academic work, need to extend the Bench4Q for the execution and completion of some work, the version used in extension was 1.3. This pape is extation the Bench4Q's paper original.

Bench4Q is available on the Internet at http://forge.ow2.org/projects/jaspte. You can find latest version there. Bench4Q is distributed the zip file Which you shouldnt expand using unzip, WinZip (http://www.winzip.com/) or similar.

1.2 Targeted Audience

This document is targeting two types of audience:

- People who just want to use right away the Bench4Q Tool. This is for those who will use the Bench4Q Tool to benchmarking the middleware.
- People who would like to modify Bench4Q Tool to fit their particular needs. You may want to change a little bit our Bench4Q Tool to add some functionality or replace a component with another one.
- People wishing to undertake study and transient analysis or a modulated load

1.3 Structure of the document

This document will guide you on:

- A brief introduction to Bench4Q in section 2, and must justify the extension done.
- A brief introduction to Bench4Q tool in Section 3, followed by som practical examples of the extension.

2 Bench4Q: Workload Model

2.1 Overview

Bench4Q is available on the Internet at http://forge.ow2.org/projects/jaspte. You can find latest version there. Bench4Q is distributed the zip file Which you shouldnt expand using unzip, WinZip (http://www.winzip.com/) or similar. All necessary information on this benchmarking can be found in its original documentation distributed with the tool.

2.2 USP Extension

The main challenge of the new benchmarks is to make the results presented, provide relevant information to these different services with different capacities and guarantees these services.

Most aplicaçõesWeb are designed as Multi-tiers systems, due to the flexibility and software reusability, however it is difficult to model the Web application behavior multi-tiers, due to the fact that the workload stimulates the dynamics of system in different levels of the layer.

As part of performance analysis in computer systems, we set the benchmark as the act of measuring and evaluating performance computing, networking protocols, devices and networks, under reference conditions relative to a reference evaluation.

However, despite the existence of various benchmarks and tools for the study, none of them stimulate the transient dynamics of the system and allow an evaluation in transient

The proposed and implemented extension in Bench4Q aims to meet the requirements of MEDC model, which restricts the modulation magnitude of the workload generated by the benchmark. Craving analysis of dynamic systems and that enables the transient analysis of sitema SUT

3 Bench4Q Tool

3.1 QUICK Introductions

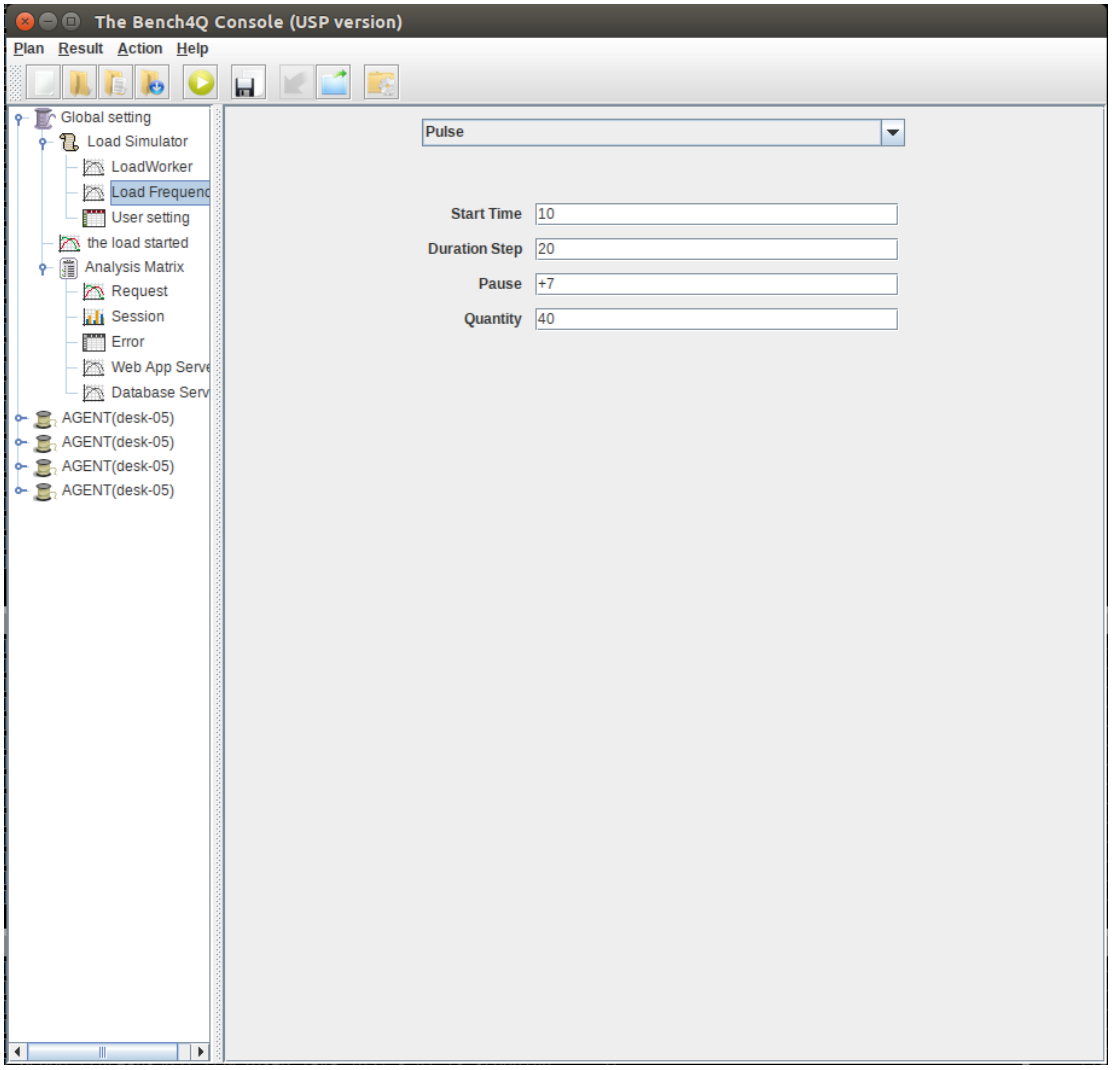
Bench4Q Tool is designed for Bench4Q benchmarking. Bench4Q Tool offers a convenient way to configure the test and analysis the test result. Now, it is possible Workload Modeling

3.2 Bench4Q Tool Design

Bench4Q tool composed of three parts, console, agent and SUT(system under test), this had no modification.

3.2.1 Cosole

The console configures the test, collate and display result. The main frame of console windows has a new tab "Load Frequency" is Showed Following in the picture.



The new tab, which configures the running of the experiment relating to the parameters of the extent of generation of the modulated load. For this option, you must fill in the fields (Start Time, Duration Step, Pause and Quantity) that will generate the modulated load as planned.

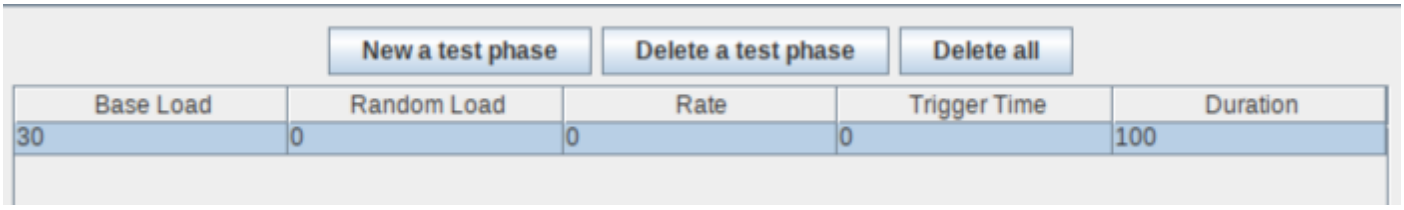
- **Start Time:** A period of time that the workload is modulated, characterizing the behavior of the change requests programmed manner;
- **Step Duration:** as shown in Chapter 2, the modulation will be displayed on Degray way;
- **Pause:** Period of interruption / pause after the load planning time;
- **Quantity:** book a number of EBs customers in case of Bench4Q) that are dedicated exclusively to the load modulation.

4 Getting Started

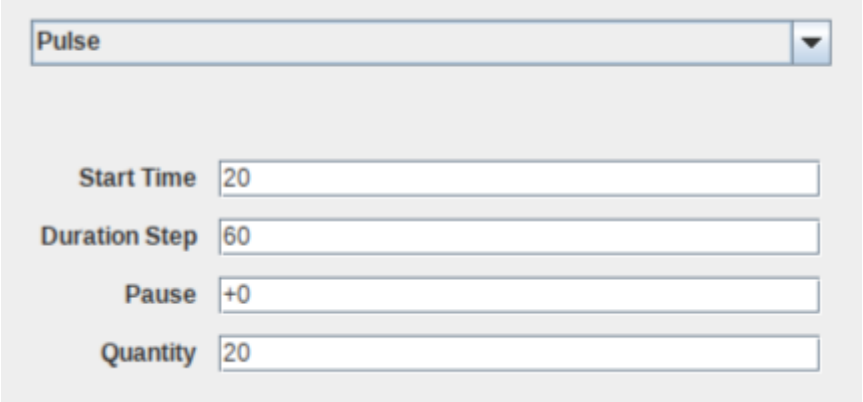
In order to run a Bench4Q test, you will need to get some basic information of the Bench4Q Tool.

4.1 Execution workload model

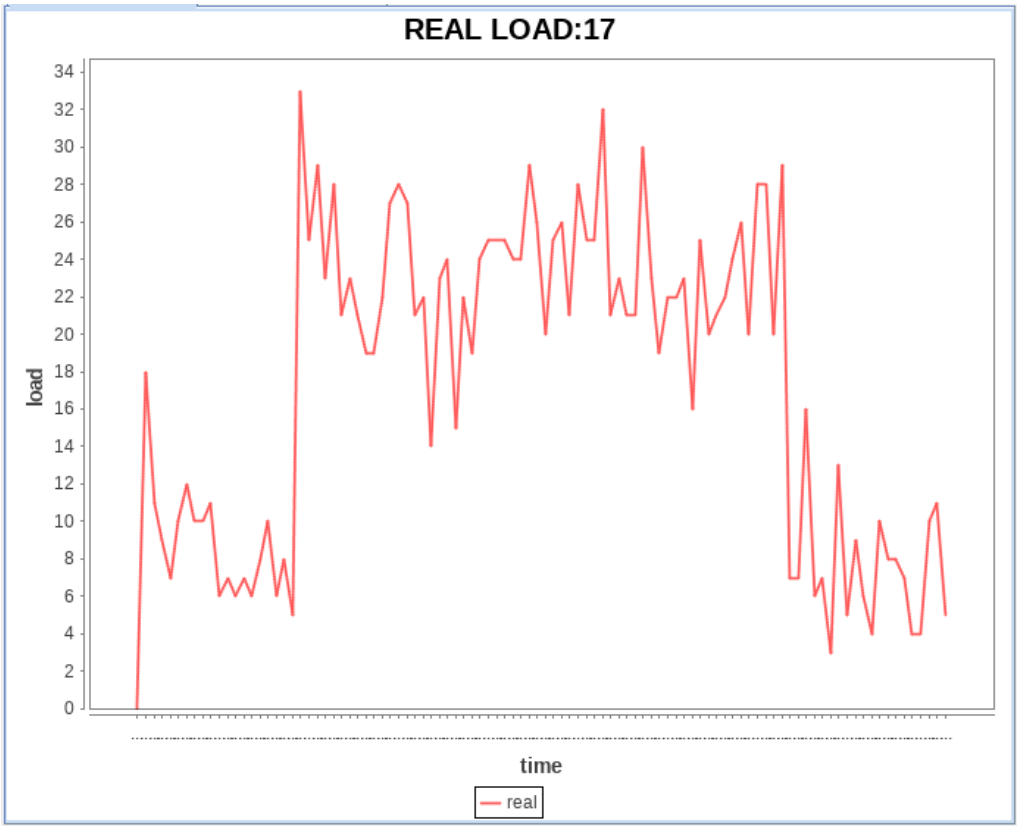
Load Work the tab, enter the following values:



Na nova tab, Load Frequency, insirá os parâmetros utilizado para fazer o teste conforme a figura a seguir:



The result can be analyzed by the next figure, this graph is native Bench4Q itself, which shows the load behavior over time. Although stochasticity of the load is modulated as programmed, that is characteristic of stochasticity Bench4Q, in order to maintain a more realistic behavior with clients accessing a stochasticity e-commerce.



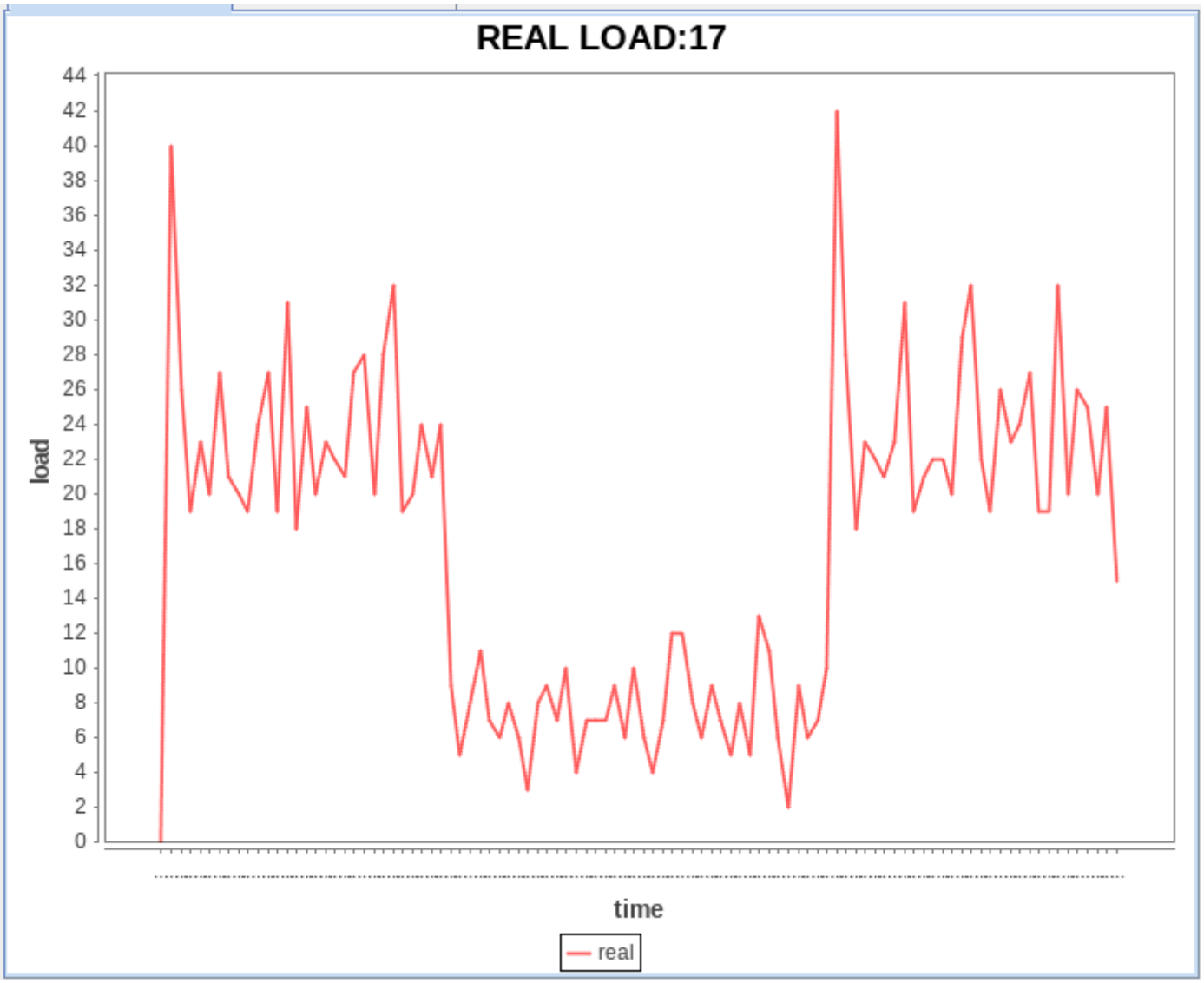
4.2 Other examples

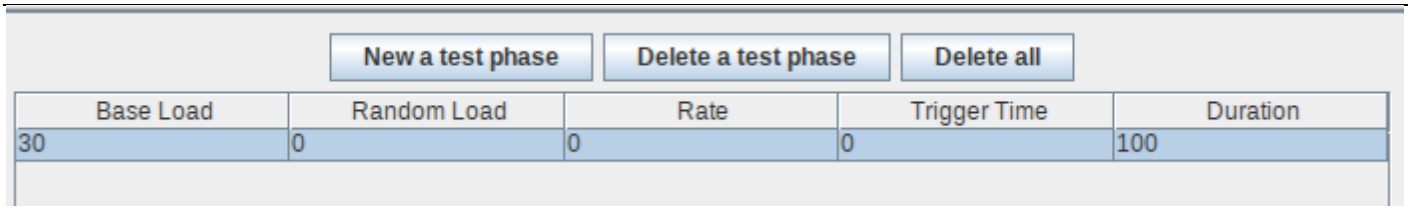
New a test phase | Delete a test phase | Delete all

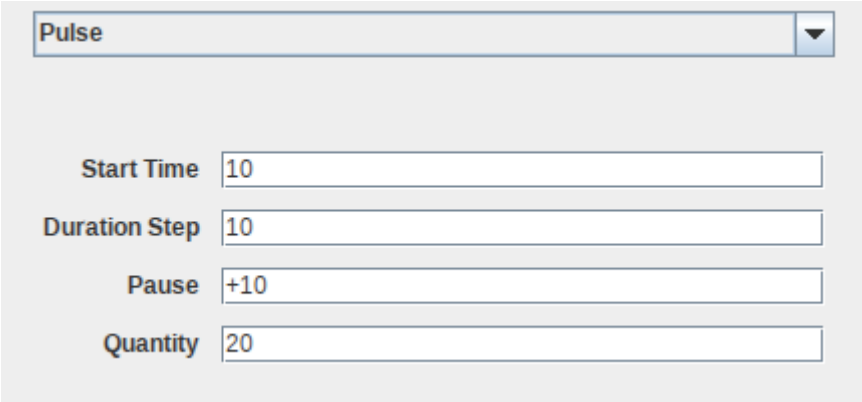
Base Load	Random Load	Rate	Trigger Time	Duration
30	0	0	0	100

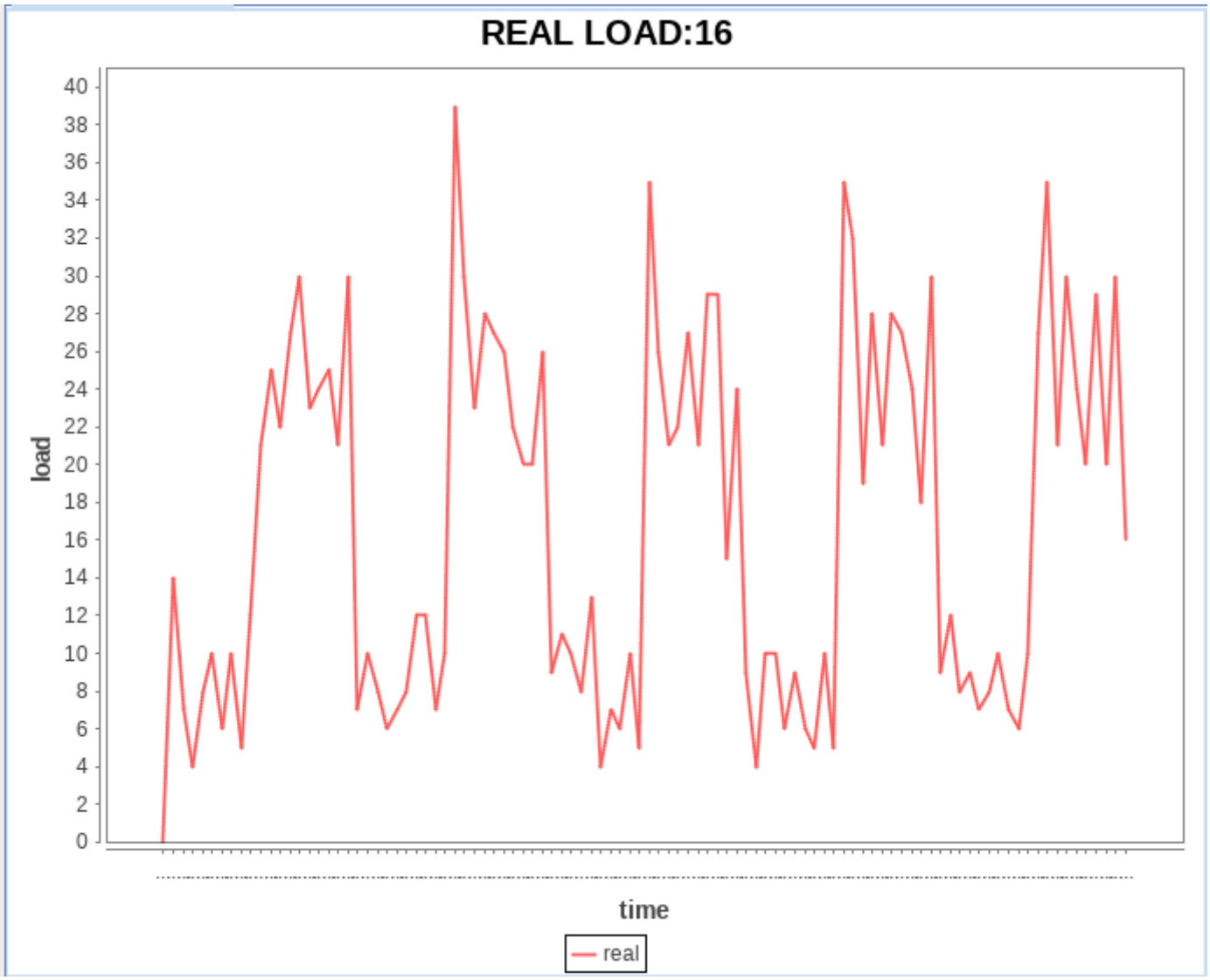
Pulse

Start Time: 0
Duration Step: 30
Pause: +40
Quantity: 20




New a test phase
Delete a test phase
Delete all
Base Load
Random Load
Rate
Trigger Time
Duration
30
0
0
0
100


Pulse
Start Time
10
Duration Step
10
Pause
+10
Quantity
20


REAL LOAD:16
load
time
real
0
2
4
6
8
10
12
14
16
18
20
22
24
26
28
30
32
34
36
38
40