# Extensão da geração de carga do Bench4Q para Benchmark de Desempenho em Regime Transiente.

***Flavio Luiz dos Santos de Souza***

**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. *EXEMPLAR DE DEFESA*

Área de Concentração: Ciências de Computação e Matemática Computacional

Orientador: Prof. Dr. Francisco José Monaco

**USP – São Carlos**
**Janeiro 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.
63 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. *EXAMINATION BOARD PRESENTATION COPY*

Concentration Area: Computer Science and Computational Mathematics

Advisor: Prof. Dr. Francisco José Monaco

**USP – São Carlos**
**January 2016**

*Este trabalho é dedicado às crianças adultas que,*
*quando pequenas, sonharam em se tornar cientistas.*
*Em especial, ao pesquisadores do Instituto de Ciências Matemáticas e de Computação (ICMC).*

# 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. 63 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. O referido *framework benchmark* 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 à perturbações e atender a requisitos de desempenho em regime transiente. No entanto, este *benchmark* não abrange os estados transiente do sistema. O presente trabalho tem por objetivo estender o *benchmark* Bench4Q acrescentando-lhe capacidades de excitar a resposta transiente do sistema mediante a perturbações da carga de trabalho. Para isso, o *software* foi acrescido de funcionalidade capaz de gerenciar a modulação da carga de trabalho. E 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. 63 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 framwork 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

# LISTA DE CÓDIGOS-FONTE

# LISTA DE TABELAS

# 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

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 uma importante e fundamental meio computacional.

O conceito 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 (**??**). **??**) exemplificam com o crescimento do Instagram, que construiu a sua base de usuários de 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 têm levado a uma quantidade considerável de publicações de artigos científicos nos últimos anos (**??**). Como apresentado nos trabalhos de **??**) e **??**), o número de pesquisas têm 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 tem devem ser alocados de forma autônoma, para que possam responder às influências externas, como a carga de trabalho. Observa-se, por outro lado que, os *data center* são muitas vezes subutilizados devido ao excesso de provisionamento, bem como as demandas de recursos que variam no tempo de acordo com os sistemas (**??**). **??**) afirmam que tempo e dinheiro são investido 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 ou escalável que pode ser utilizada para obter recursos sobdemanda; no entanto, um problema em aberto é decidir sobre a correta alocação de recursos ao implantar a nuvem (**??**). 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 muda imediatamente quando o pedal do acelerador é acionado assim como a temperatura em uma sala muda instantaneamente quando um aquecedor é ligado.

Este comportamento dinâmico começa a ser apreciado em grande sistemas computacionais, quanto 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 *e-commerce* conhecido como *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. Apesar de estar hospedado o projeto não com ciência de que o volume de requisições aumentaria, não foi possível manter a qualidade de QoS, tempo de respostas, aos clientes.

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 (**??**). 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 **??**), buscam identificar padrões de comportamento na carga de trabalho e otimizar a provisão dinâmica de recursos nas máquinas virtuais com base em algoritmos reativos. **??**) buscam detectar, por monitoramento, a padronização e a tendência da carga de trabalho com o objetivo de otimizar os recursos. **??**) também 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.**??**), utiliza de técnicas de estatística e avaliação dos recursos disponíveis para tomar a decisão de qual recurso deve ser

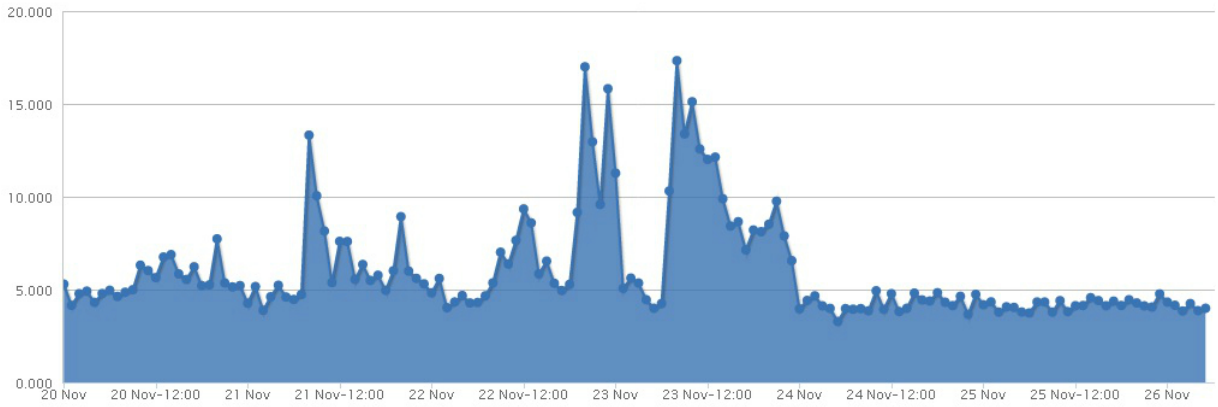



Figura 1 – Tempo de resposta do *Black Friday* Brasil 2012

Fonte: Adaptada de **??**).

alocado.

**??**) afirmam que atualmente a maioria das aplicações Web são concebidas com sistemas *mult-tiers* (multi-camadas) devido à flexibilidade de escalabilidade. O planejamento de capacidade é um passo importante para determinar a quantidade de recursos exigido para garantir determinada QoS. No entanto, em geral o planejamento de capacidade é basicamente uma decisão de longo prazo e quase estático, e os recursos são determinados pela taxa de utilização máxima da aplicação para evitar a pena excessiva de QoS.

**??**) afirmam que a partir de um planejamento de capacidade, 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 um sistema dinâmico, 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 (**??**).

Algumas vezes, é possível representar sistemas dinâmicos através de equações diferenciais 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 (**??**). 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, medição direta via instrumentação apropriada adequa-se nesse caso as necessidade; no entanto, é necessário a existência e disponibilidade do sistema ou protótipo, pois a avaliação é feita através de estímulos às entradas (*benchmark*) e leitura de suas saídas, possibilitando testes de caixa preta em que não é exigido conhecimento sobre o funcionamento interno do sistema (**??**), 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 analise. O desenvolvimento, contudo, de sistemas distribuídos de larga escala e complexidade altera essa realidade (**??????**).

No *Laboratório de Sistemas Distribuídos e Programação Concorrente (LaSDPC)*[1], onde desenvolve este projeto, trabalhos anteriores lidam com essa temática. O trabalho de **??**), 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, **??**) 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 **??**), em desenvolvimento também no LaSDPC, estuda e define uma metodologia de análise transiente dedicado a sistemas computacionais dinâmicos reais utilizando da especificação arquitetural proposto por **??**). 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 **??**) utiliza de um *benchmark* para a validação experimental de sua metodologia. No entanto, o *benchmark* escolhido por **??**), Bench4Q, não contempla das especificações apresentadas por **??**).

Segundo **??**) os *benchmarks* tradicionais não são suficientes para a análise desses novos serviços de elasticidade da computação em nuvem. O principal desafio dos novos *benchmarks* é

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

fazer com que as métricas ofereçam informações relevantes a esses diferentes serviços e com diferentes capacidades e garantias desses serviços. **??**) afirmam que, a maioria das aplicações Web são concebidas como sistemas *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 existam diversos *benchmarks* e ferramentas para o estudo, nenhuma delas estimulam a dinâmica transiente do sistema e permite uma avaliação em regime transiente, que se faz necessário para a pesquisa de **??**). A proposta deste trabalho é identificar um *benchmark* 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 **??**). 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 (**??**). A elasticidade é uma característica no contexto de 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 (**????**).

A elasticidade permite aos usuários 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 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 pode ser abordado utilizando-se diferentes abordagens (**??**).

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.

**??**) 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 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 (**??**). 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.

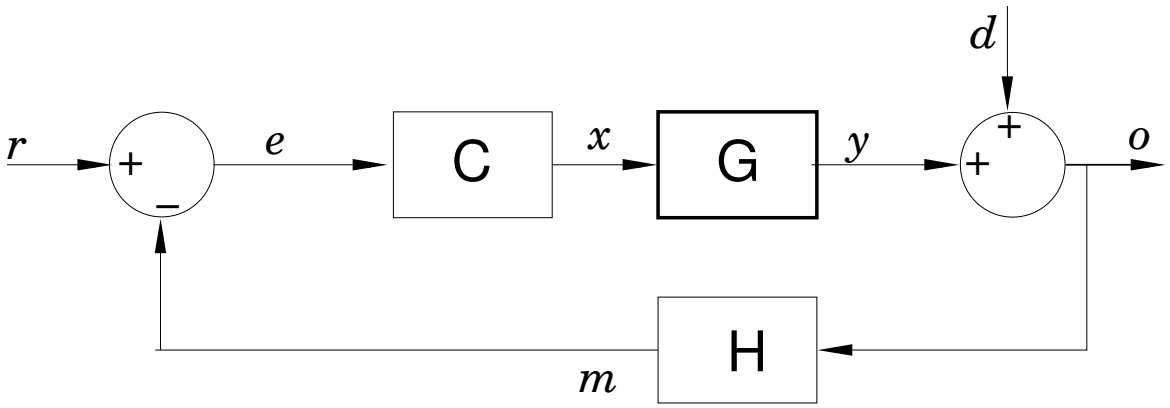



Figura 2 – Diagrama de blocos de um sistema de controle

Fonte: **??**).

**??**) foi ao encontro de soluções adaptativas para o gerenciamento de recursos e alta disponibilidade de aplicações 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. Formalmente um sub-dinamismo possui memória. 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 são funções 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 (**??**).

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 (sob condições normais de temperatura e pressão), 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.

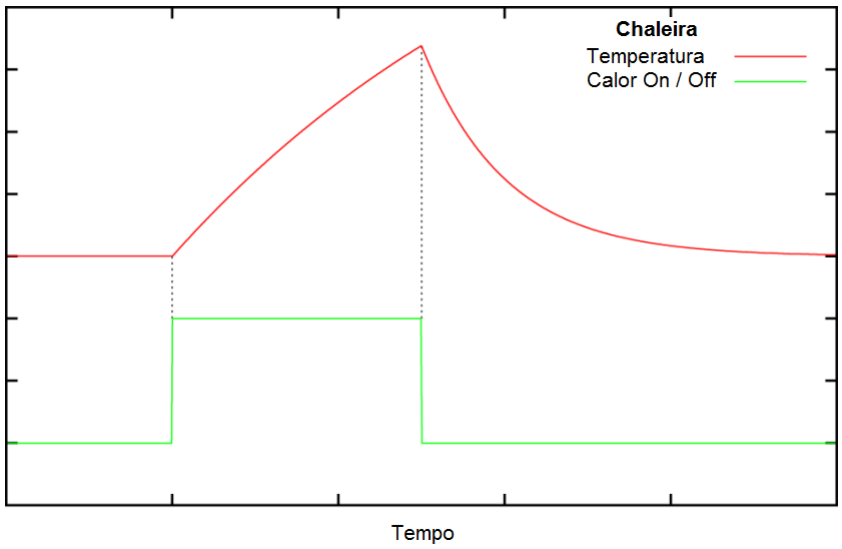



Figura 3 – Dinâmica e comportamento da chaleira aquecendo

Fonte: Adaptada de **??**).

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 á certa altura. Ao deslocarmos o peso tirando-o do repouso e soltarmos após o tencionamento 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.

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 (**??**).

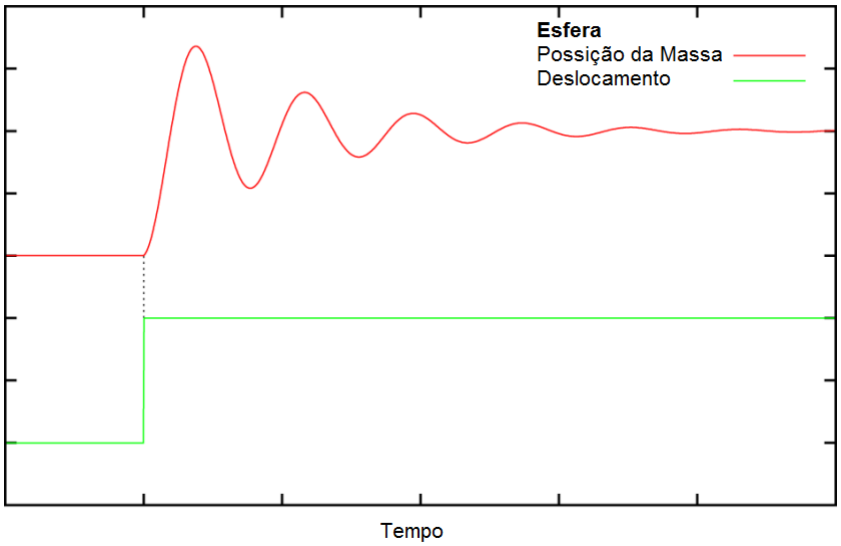



Figura 4 – Dinâmica e comportamento da mola

Fonte: Adaptada de **??**).

Embora a análise transiente ainda seja pouco explorada frente as principais abordagens de avaliação de desempenho, **??**) 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 computação 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 vários níveis. O efeito de pequenos atrasos e a propagação para outras camadas podem render comportamento dinâmico significativos (**??**).

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 (**??**).

Segundo **??**), 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 **??**) 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*. O estudo, inicia discutindo sobre as propriedades dinâmicas de sistemas

computacionais de grande escala em ambientes distribuídos e como essas características podem afetar o desempenho. 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.

**??**) apresentam um conjunto de requisito que especificam uma arquitetura conceitual, intitulada MEDC (*Monitor, Effector, Demanda and Capacity*) representado na Figura 5. 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*.

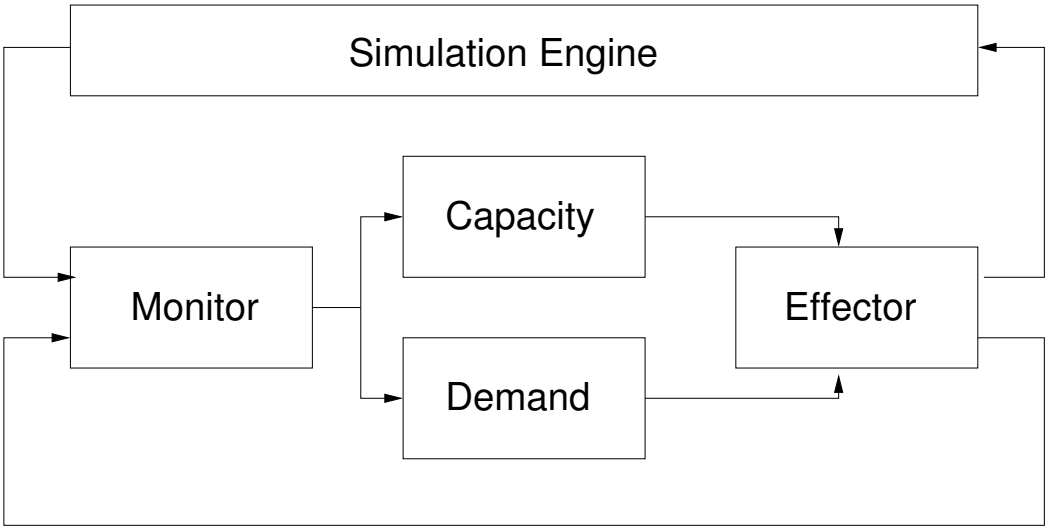
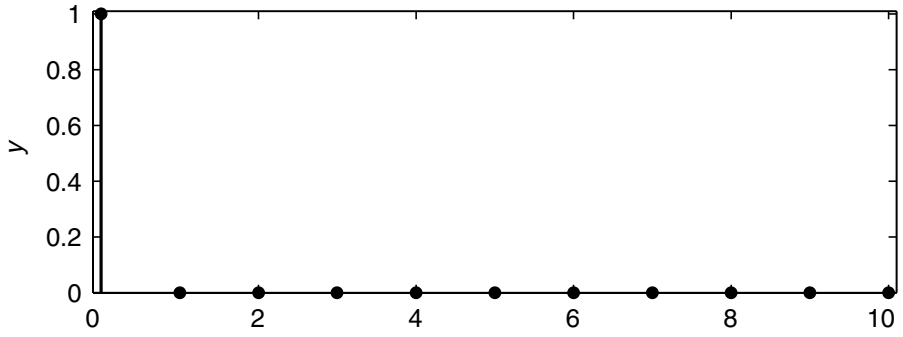



Figura 5 – Arquitetura conceitual MEDC

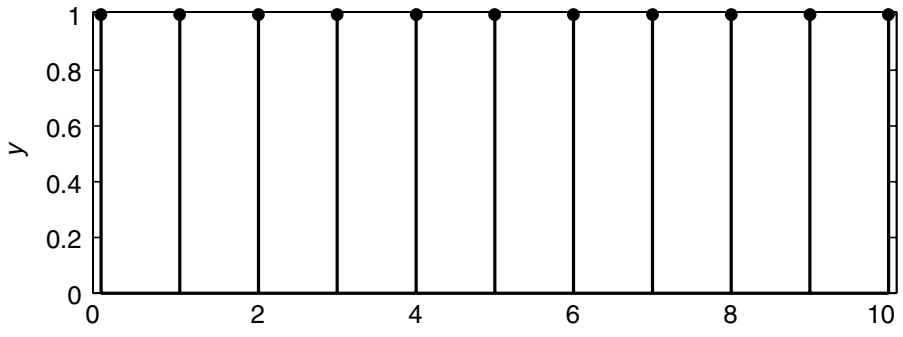
Fonte: **??**).

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

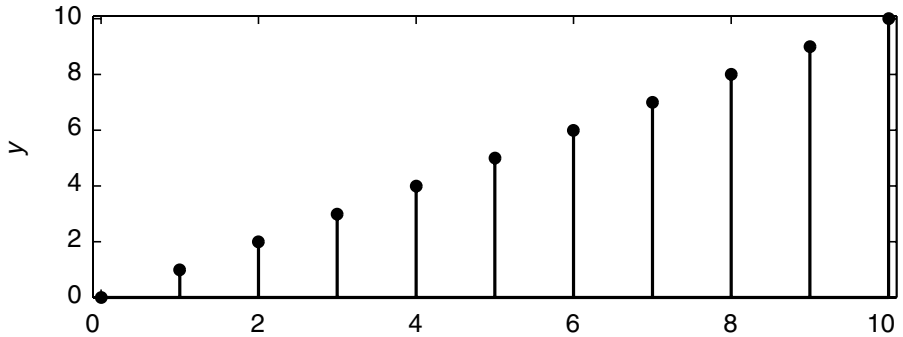
ou o liberação de novas entidades de carga de trabalho e de recursos. A ordem é efetivamente realizada pelo *Effector*, que recebe as solicitações dos controladores *Demand* e *Capacity* os manipulam antes de transmitir os eventos para o mecanismo de núcleo. É através do *Effector* que pode-se modelar, por exemplo, atrasos associados a alocação e desalocação, 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 exite uma ação associada de forma iterativa em cada etapa do ciclo de simulação (**??**).




(a) Impulse: $y(0) = 1$, $y(k) = 0$ for $k > 0$




(b) Step: $y(k) = 1$



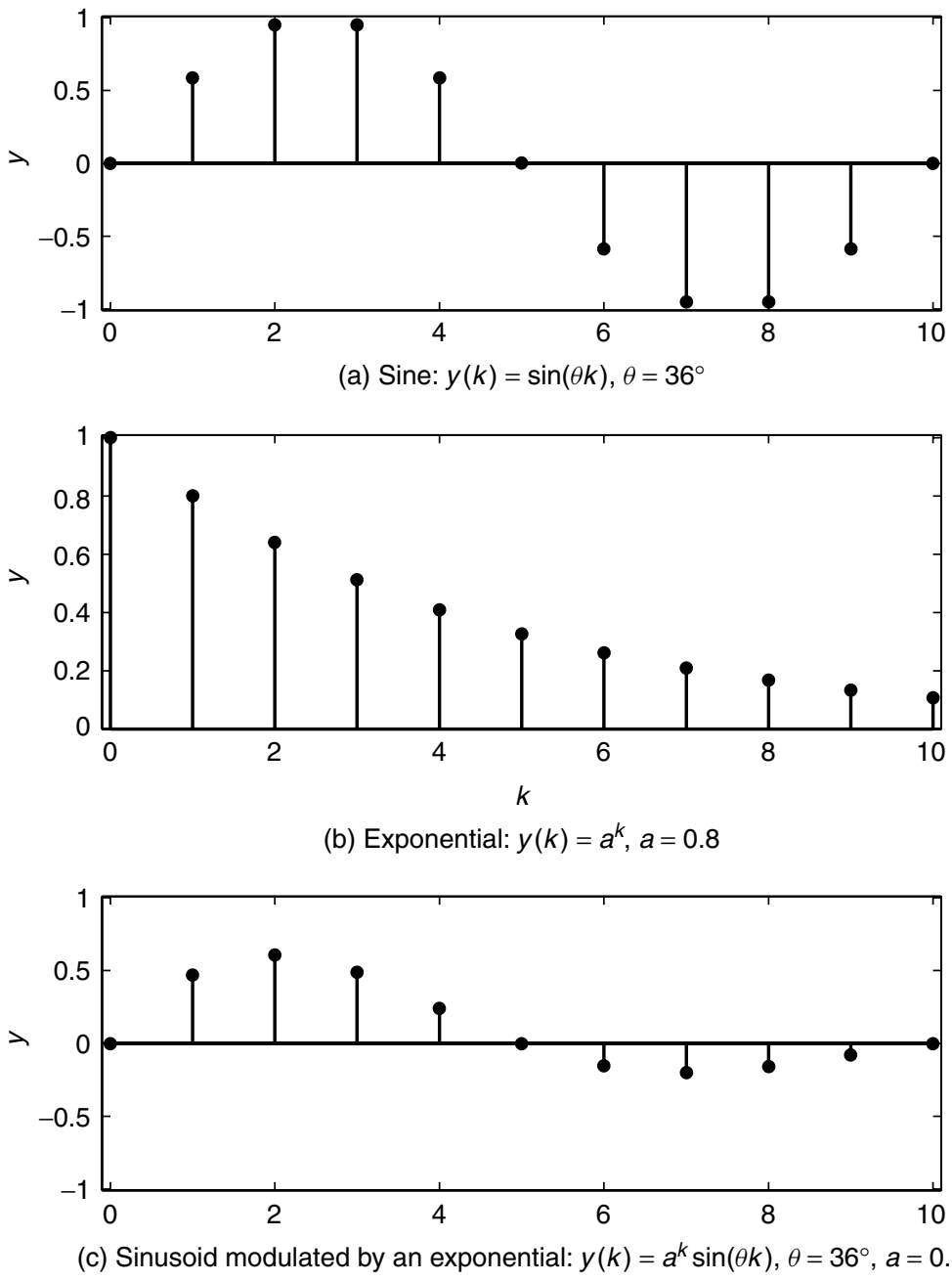

(c) Ramp: $y(k) = k$

Figura 6 – Sinais em tempo discreto comuns, parte 1

Fonte: **??**).

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 cargas de trabalho que possam provocar comportamentos propícios, onde as caraterísticas dinâmicas do sistema sejam claramente observadas **??**), 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 6 e 7.




(a) Sine: $y(k) = \sin(\theta k)$, $\theta = 36°$

(b) Exponential: $y(k) = a^k$, $a = 0.8$

(c) Sinusoid modulated by an exponential: $y(k) = a^k \sin(\theta k)$, $\theta = 36°$, $a = 0.8$

Figura 7 – Sinais em tempo discreto comuns, parte 2

Fonte: **??**).

**??**) pretende utilizar a especificação de arquitetura proposta por **??**) 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*, (**??????????**) :

**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 fornecer 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* (**??**).

**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. (**??**)

**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 (**??**).

**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 (**??**).

**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 (**??**). 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 (**??**).

**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 repetibilidade (**??**).

**Métrica** Uma métrica ser claramente compreensível. Segundo **??**) 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 (**??**).

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 (**??**).

O trabalho **??**), apresentam o conceito de sessão, que define uma sequencia de requisições de um único cliente. **??**), 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 (**??**), 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 (**??**).

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 8. 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 (**??**).

A ferramenta é composta por três partes: Console, Agente e SUT, conforme apresenta a Figura 8, 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.

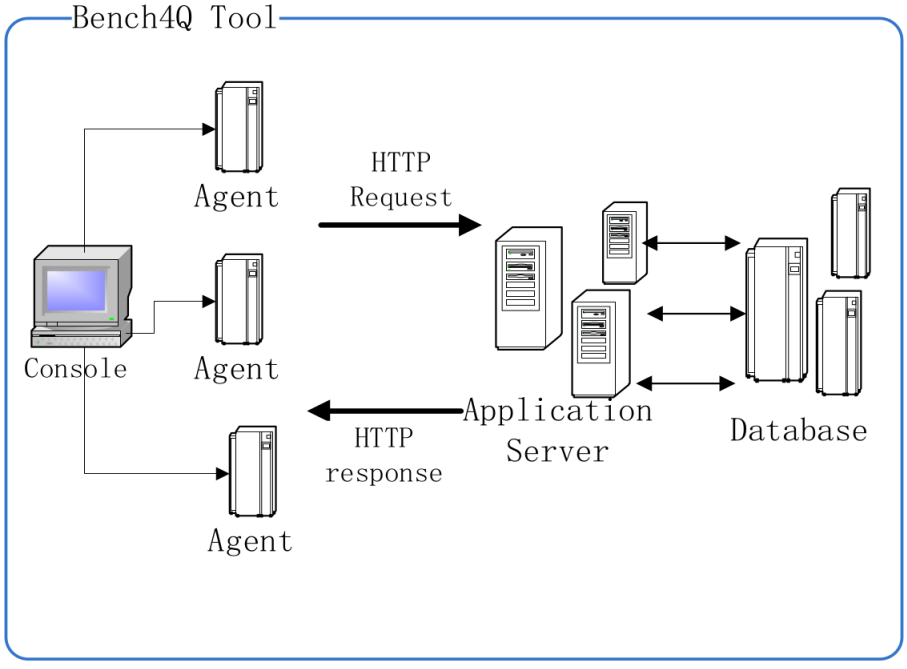



Figura 8 – Arquitetura Bench4Q

Fonte: **??**).

- **Console:** A Figura 9 apresenta a interface do console, onde configura-se o teste, coleta e exibição os resultados.

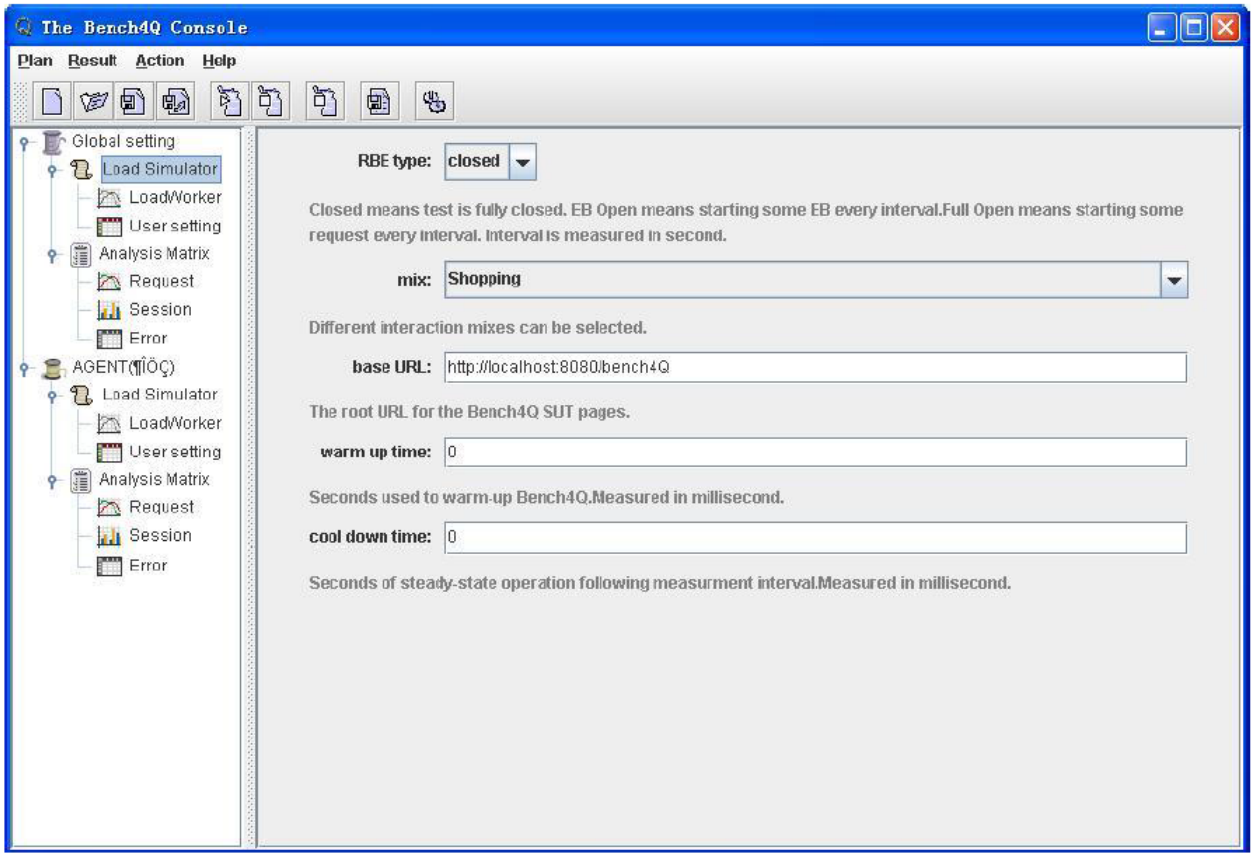



Figura 9 – Console Bench4Q

Fonte: **??**).

- **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 10, 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 (**??**). 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.

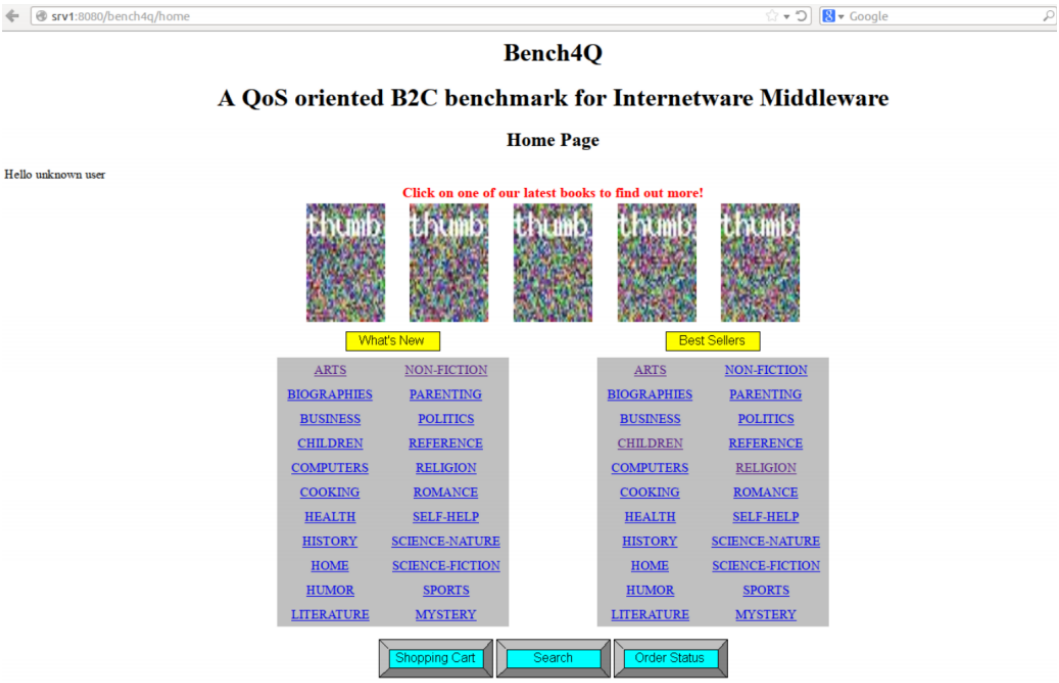



Figura 10 – SUT Bench4Q

Fonte: **??**).

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 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 (**??**). A Figura 11 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

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

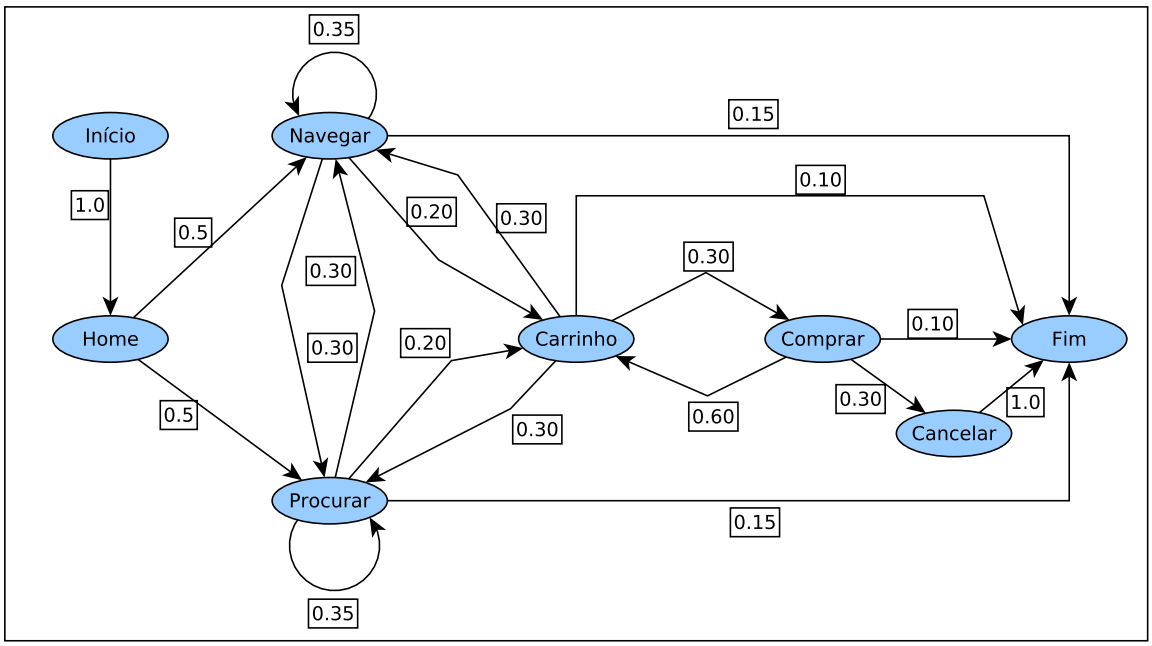



Figura 11 – CBMG - perfil de navegação dos *brwosers* do Bench4Q

Fonte: Adaptada de **??**).

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 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 (**??**). 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 extenção 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 **??**) 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 12, 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 error implicaria em maior probabilidade de compras efetivas.

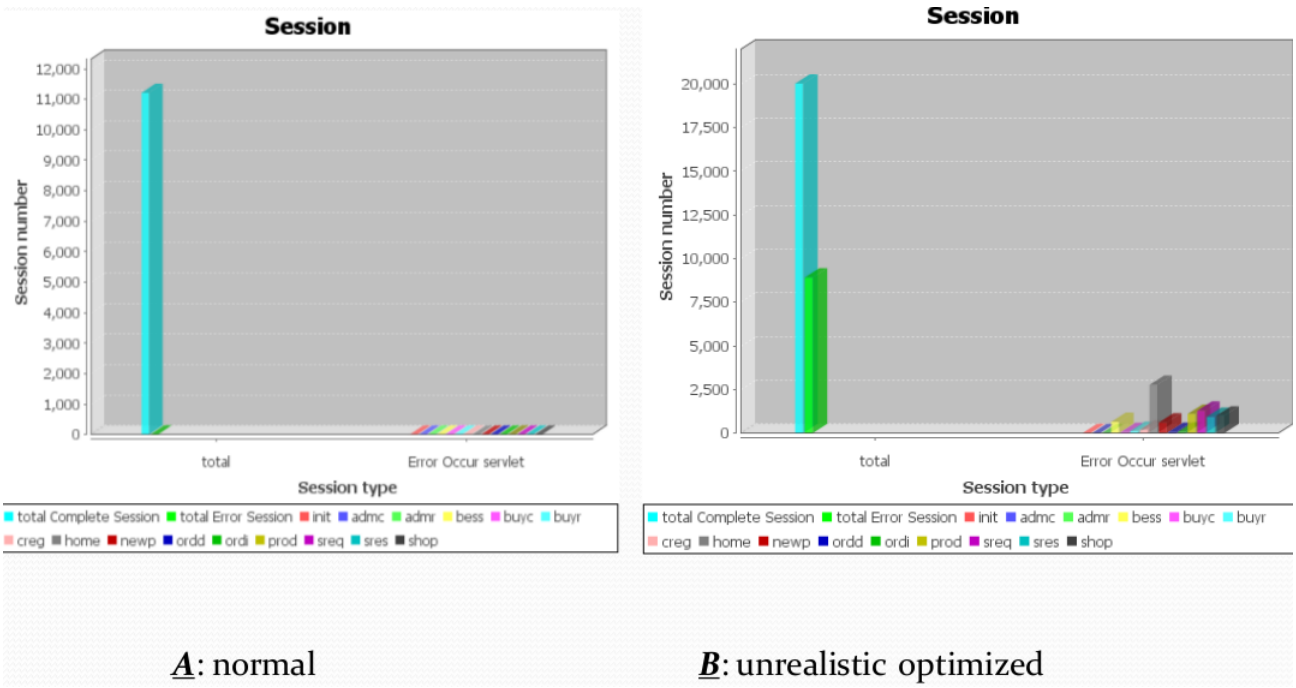



Figura 12 – Comparação da quantidade de requisições completadas com sucesso entre dois cenários: normal e otimizado não realisticamente.

Fonte: **??**).

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 (**????**). 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.

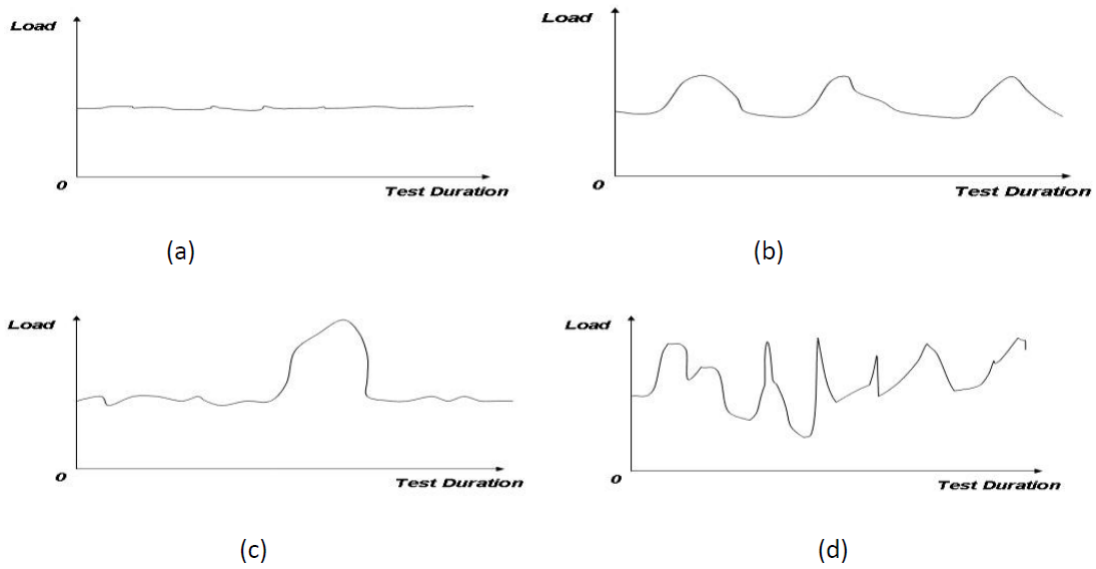
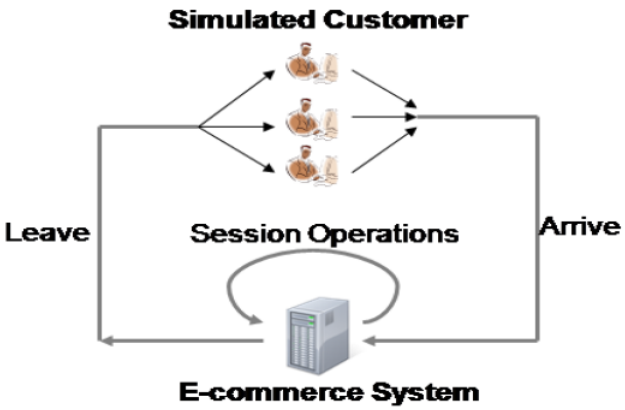



Figura 13 – Carga de trabalho gerada pelo Bench4Q

Fonte: **??**).

Existem dois modos de simulação de carga: o fechado e o aberto, conforme o serviço na Figura 14. No modo fechado, ilustrado na Figura 14 (a), um novo cliente só acessará depois do cliente antigo deixar o sistema. Já no modo aberto, ilustrado na Figura 14 (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 (**??**).




(a) Closed Mode


(b) Open Mode

Figura 14 – Tipo de sessões Bench4Q

Fonte: **??**).

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

CAPÍTULO

# 3

# METODOLOGIA

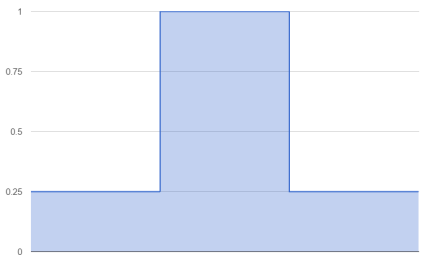
O objetivo deste trabalho é estender a carga de trabalho do *benchmark* Bench4Q para que seja possível estimular o sistema a apresentar a sua dinâmica, e assim possibilitando a analise transiente do sistema, incorporando-se o modulo *Demand* da arquitetura conceitual MEDC.

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. Sendo assim, com o preenchimento de um conjunto de parâmetros será possível a geração modulada da carga:

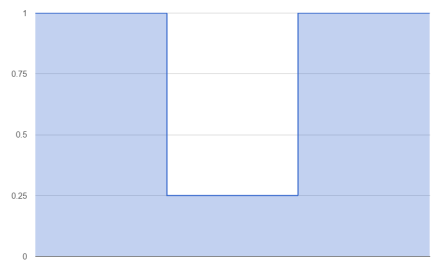
- **Tempo de planejamento de carga:** Um período de tempo em que a carga de trabalho é modulada, caracterizando a mudança do comportamento das requisições de maneira programada;
- **Tipo de modulação:** conforme apresentado no Capitulo 2, a modulação será apresentada conforme as funções ou sinais propostos por **??**);
- **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 cargas conforme os exemplos apresentados na Figura 15. Esses exemplos são derivados das funções apresentadas por **??**). Com esse tipo de variação da 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 utilizado em estudos de avaliação de desempenho não estacionária. A carga de trabalho Figura degrau da 15a, é a de maior interesse para o trabalho de

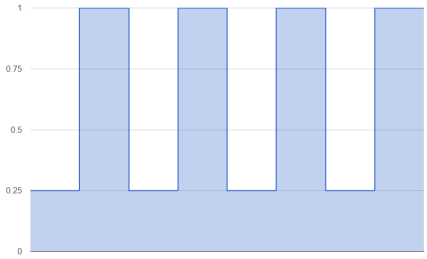
**??**). A carga se inicia estacionária e tem um crescimento e brusco, podendo se manter perene, ou retomar ao patamar inicial. Já carga representada pela Figura 15b tem comportamento oposto a Figura 15a. Por fim, a carga modulada conforme a Figura 15c, oscila da uma baixa intensidade à máxima intensidade. sempre com alterações bruscas, repentinas e com os mesmos intervalos de unidade de tempo.




(a) Degrau positivo




(b) Degrau negativo




(c) Onda quadrada

Figura 15 – Possibilidade de cargas moduláveis pela extensão

O Bench4Q simula de um *website* de um *e-commerce*, utilizando uma arquitetura *multi-tier*. Esta arquitetura particiona o processo de aplicação em níveis, onde cada camada fornece uma determinada funcionalidade. Uma vantagem de tal arquitetura é que pode proporcionar um elevado nível de escalabilidade e 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 arquitetura definida é esquematizado na Figura 16. Os parâmetros 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.

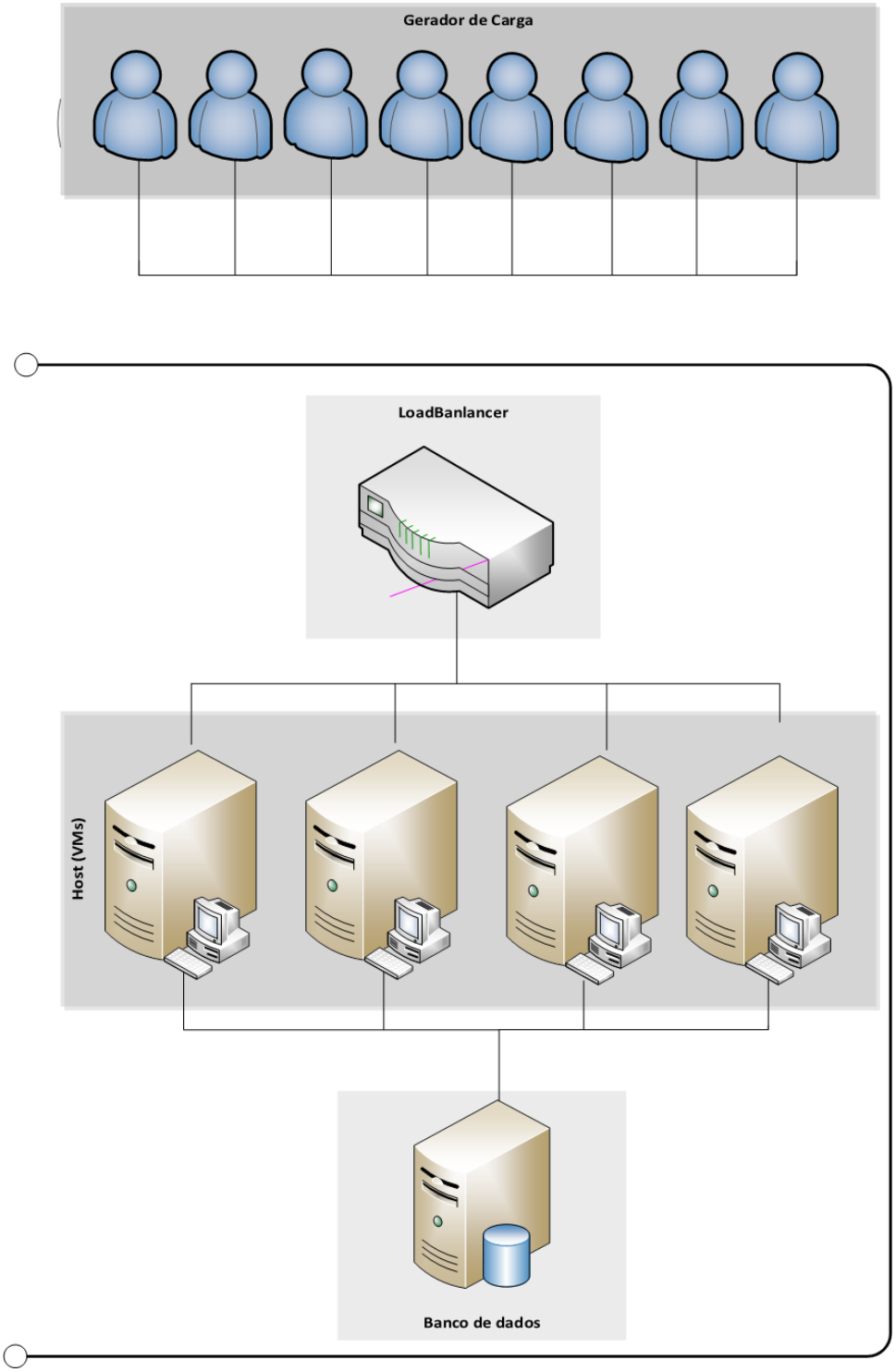



Figura 16 – Arquitetura do experimento

Fonte: Elaborada pelo autor.

O comportamento do sistema é descrito com a base em métricas. A métrica é uma função que transforma resultados medidos em uma forma facilmente compreendida (**??**). 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) (**??**). As métricas

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.

tradicionais, de analise estacionaria, não podem capturar os comportamentos transitórios do sistema em resposta a modulação da carga de trabalho.

**??**) apresentam as características e os comportamentos de uma métrica transiente, conforme ilustrado pela Figura 17, 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 performance)** - a diferença entre o *required vs performance*, e o desempenho em estado estacionário, após a redistribuição;

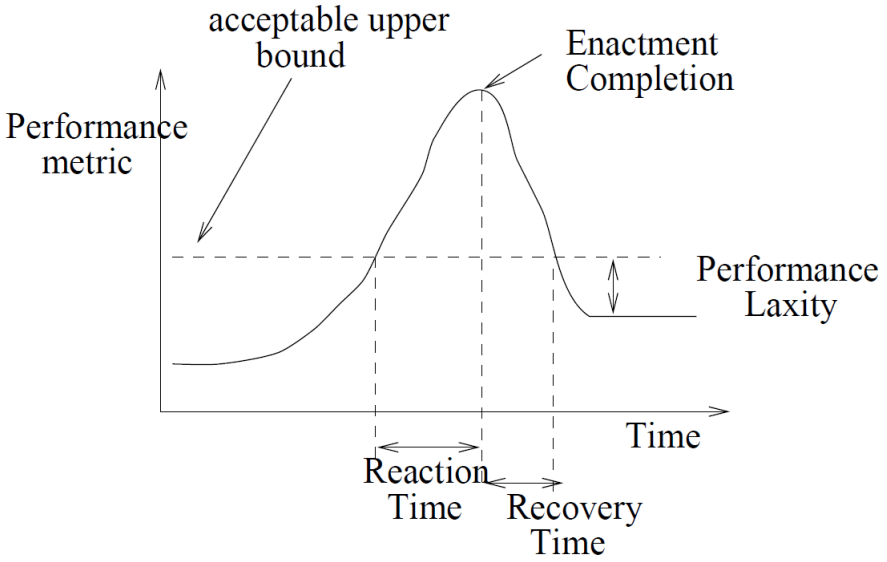



Figura 17 – Comportamento de métrica transiente

Fonte: **??**).

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 **??**). Existem diversos trabalhos dedicados a identificarem métricas transientes em vários contextos como estado por **??**), **??**) e **??**). **??**) 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 nuvem, mas é necessário medir diferentes métricas para sistemas escaláveis (ou seja, dinâmicos) onde os recursos vêm e vão. Ainda **??**) enfatiza que os novos *benchmarks* devem relatar métricas diferentes do que os *benchmarks* existentes: Em vez 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. Neste projeto é interessante salientar:

- **Conexões por segundo (*Load Balancer*):** Conforme sugerido por **??**), 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*): ??**), 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): ??**) 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 **??**), 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* | 20 EBs |
| *Nível 2* | 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. São feitas assim essa carga de trabalho estimulará o sistema a apresentar a sua dinâmica. Neste estudo, 3 replicações para cada 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, 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 descrevemos em detalhes a implementação aplicada ao *benchmark* Bench4Q e como ele pode ser estendido. Descrevemos também algumas das complexidades na produção de distribuições para a carga de trabalho. Essa ferramenta também está disponível sob uma licença de código aberto, para que outros possam usar e estender o *benchmark*, e contribuir com novos tipos de modulação de cargas de trabalho. O código fonte esta disponível em http://gitlab.lasdpc.icmc.usp.br/edwin/bench4q. A Figura 18 mostra o diagrama de classes envolvido na extensão do Bench4Q. 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*.

Apesar de permitir a geração de carga para o sistema SUT, o Bench4Q possui algumas limitações na sua versão original que dificultam a experimentação e analise de cenários de interesse ao trabalho de **??**) e **??**). As classes disponíveis no *benchmark* original não permitem a modulação de carga de trabalho. Essa limitação implica, por exemplo, na dificuldade de projetar um controlador para o gerenciamento de recursos, pois para esta atividade é necessário uma análise de resultados transientes mediante a modulação da carga de trabalho. A simulação de uma carga de trabalho em que há a alteração introduzida ao longo da simulação é o foco deste trabalho.

Conforme o diagrama de classes na Figura 18, é possível ter uma ideia do trabalho de extensão realizado no *benchmark*. Vale salientar que o Bench4Q é uma ferramenta completa e extensa, e diagramar todas as suas classes seria difícil e aumentaria consideravelmente a complexidade de entendimento. Sendo assim, aqui apresentamos somente as classes já existem no Bench4Q e que passaram por modificações para atender aos requisitos da proposta, juntamente com as novas classes que foram necessárias para o mesmo objetivo.

O Bench4Q fornece uma estrutura e componentes compartilhados para a comunicação entre os dois módulos da carga de trabalho: **Console** e **Agente**. Apesar de trabalharem em conjunto

e para um mesmo fim, os módulos (Console e Agente) são executados em máquinas distintas. A extensão é 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 da classe original Bench4QTreeModel, adicionando os parâmetros para a modulação: 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 e compreendê-los correspondentemente a modulação configurada na interface gráfica, gerando a carga programada durante a execução.

O trabalho de desenvolver a extensão compreendeu extensivas modificações no código-fonte original do Bench4Q e a criação de novas funcionalidades. Por conta de espaço e clareza não cabe uma descrição minuciosa de todas as alterações feitas. Resumem-se, apenas, algumas das principais intervenções realizadas.

## 4.1 Configuração da carga de trabalho

O Agente do Bench4Q é um cliente conectado ao modulo Console do *benchmark*, a ferramenta permite que diversos clientes em diferentes máquinas estejam conectados em um mesmo Console. O Cliente é um programa Java para gerar as operações que compõem a carga de trabalho. Cada *thread* executa uma série sequencial de operações, fazendo chamadas para o SUT. Para distribuir e controlar a submissão da carga de trabalho ao longo da simulação, uma estratégia utilizada é a modulação da mesma através 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 código 1 apresentado na integra, esta presente na 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 ponto central que resulta no comportamento final da modulação da carga de trabalho.

**Código-fonte 1:** Algoritmo calcula os tempos de iniciaçização 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();
```

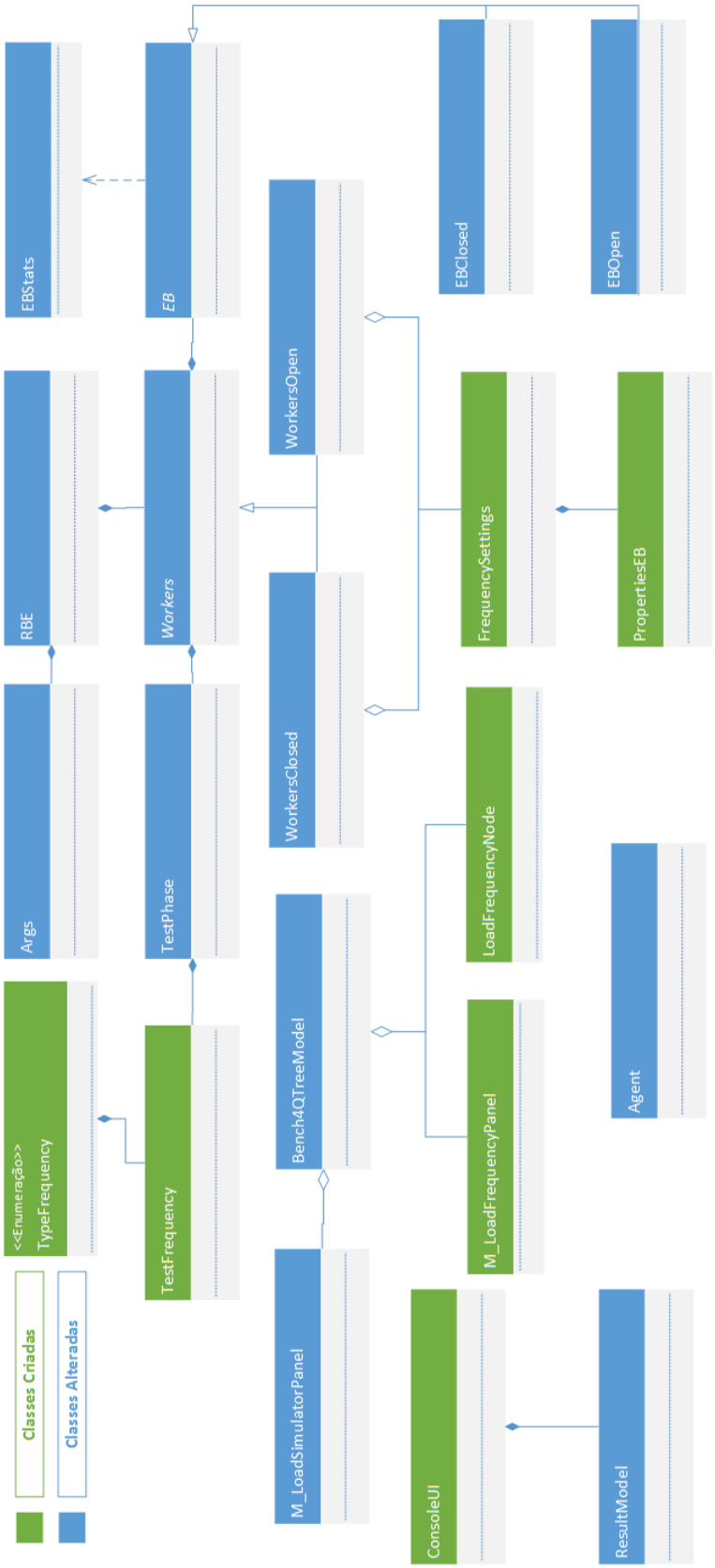



Figura 18 – Diagrama de classes da extensão do Bench4Q.

Fonte: Elaborada pelo autor.

```
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 verdade é que motivamos em tornar mais fácil a contribuição de novas variedades de carga de trabalho para outros desenvolvedores. Logo, as novas modulações a serem implementadas deve ser incluídas neste método, juntamente com o nome da nova 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 conexões, *Open*(Aberta) e *Close*(Fechada), consequentemente existem duas classes que tratam cada umas das conexões (*WorkersOpen* e *WorkersClose*) com o mesmo métodos, mas com programações diferentes. O código-fonte 2, apresentado na integra, ilustra o *core* da geração da carga referente a construção da modulação da carga já com as modificações da extensão, este método é o manipula as conexões geradas pelos clientes. Através dos parâmetros e dados calculados anteriormente é possível controlar as requisições afim de gerar as modulação deseja.

**Código-fonte 2:** 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 }
```

## 4.3 *Interface* gráfica

No console principal do Bench4Q, onde configura-se a execução do experimento, foi incluída uma nova opção *LoadFrequency* referente ao parâmetros da extensão da geração da carga modulada. Por esta opção, *LoadFrequency*, deve-se preencher os campos (*Start Time*, *Duration Step*, *Pause* e *Quantity*) que irão gerar a carga modulada conforme a programação. O código fonte 3 apresenta 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.

**Código-fonte 3:** 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    }
```

Este conjunto de classes as quais lidam, manipulam, gerenciam e modulam a carga de trabalho gerada pelo Bench4Q, utiliza de um 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 19 ilustra a interface gráfica por onde é possível modular a carga de trabalho do Bench4Q.

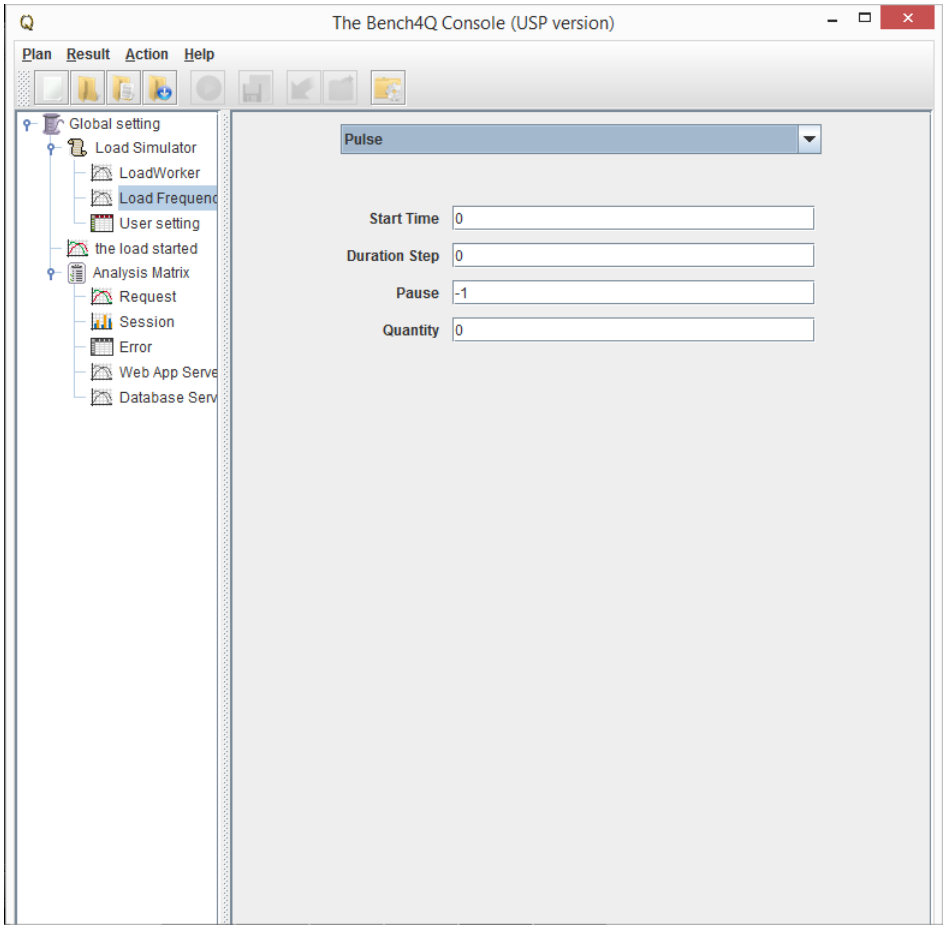



Figura 19 – Console de programação de carga de trabalho.

Fonte: Elaborada pelo autor.

## 4.4 Teste de modulação

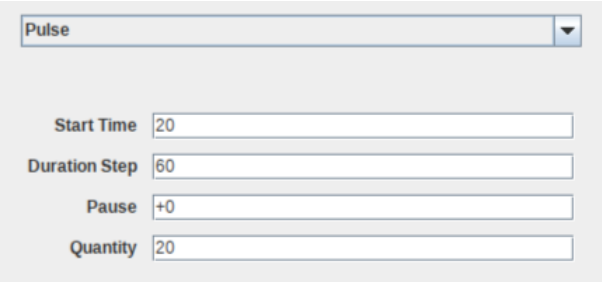
A carga de trabalho é imposta ao sistema por meio de requisições HTTP enviadas pelos EBs ao SUT que são executadas nos servidores de aplicação das máquinas virtuais instanciadas no *host*. Essas requisições exigem que as máquinas virtuais se ocupem pelo tempo necessário para processá-las, alterando o desempenho experimentado pelo sistema. Segundo **??**), 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.

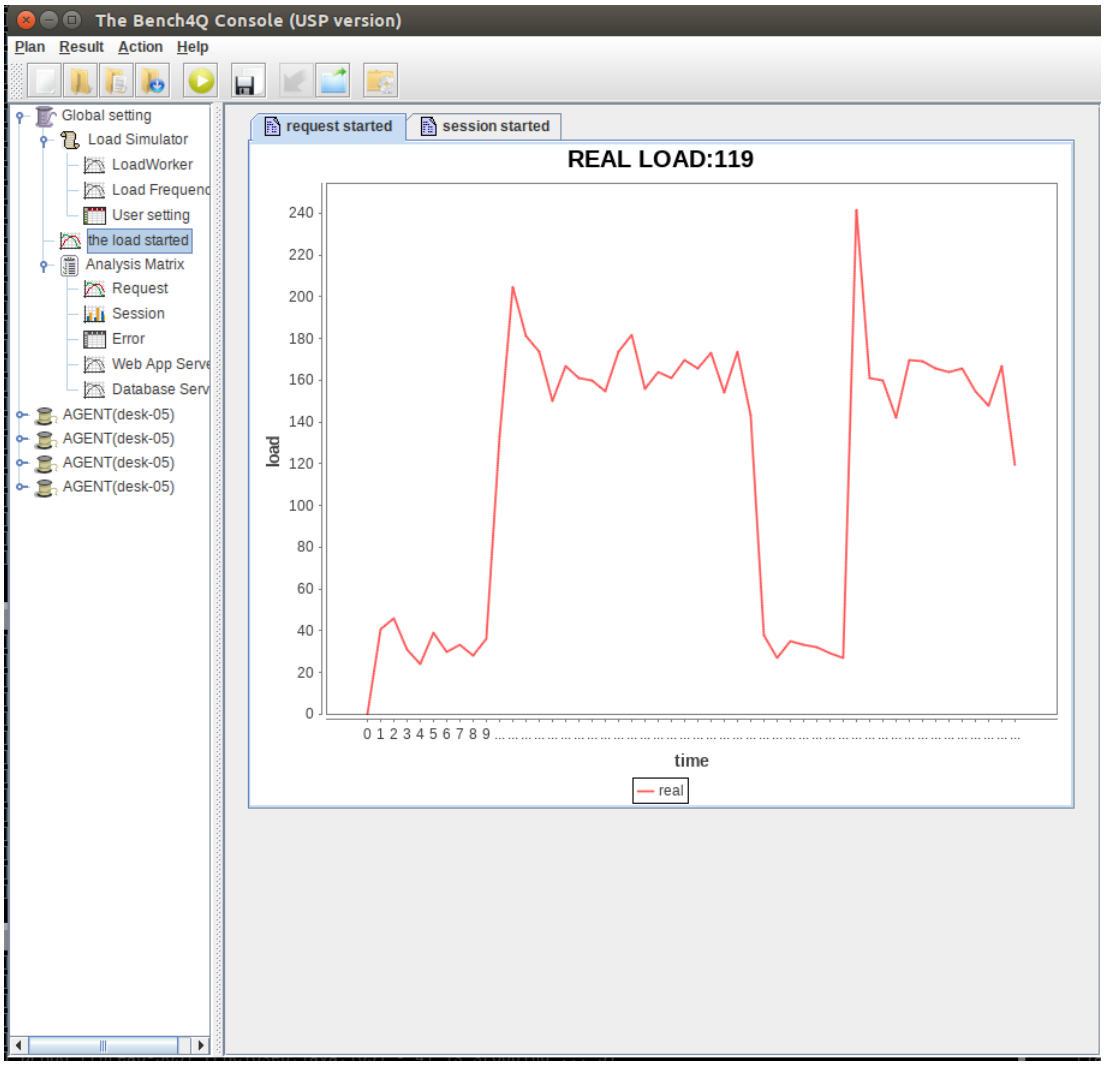
É possível informar previamente a execução os parâmetros da modulação. Por exemplo, ao escolher a opção degrau, é necessário informar quantos EBs geram o degrau, em que instante de tempo, e qual o tempo de duração e por fim qual a sua polaridade (com base em um pulso elétrico a positiva sairia de zero e chega a um, a negativa, sairia de um e chegaria a zero), é possível obter resultados conforme a Figura 20b.

A Figura 20 ilustra uma carga teste modulada já pela extensão. A figura 20a apresenta os parâmetros utilizado para fazer o teste. A carga modulada atuará a partir do 10º 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 para modular o comportamento da carga. Este comportamento pode ser apreciado no item 20b da mesma figura 20. Vale salientar que o gráfico gerado e apresentado na figura 20 de item 20b, é uma característica nativa ao *benchmark*.

O Bench4Q, possui uma documentação sobre a ferramenta. Devido a extensão do *benchmark* foi elaborada uma documentação seguindo os padrões da última versão original e esta pode ser conferida no apêndice A 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 do mesmo.




(a) Teste de configuração da carga a ser modulada




(b) Carga gerada com base na configuração teste

Figura 20 – Teste de modulação da carga

Fonte: Elaborada pelo autor.

CAPÍTULO

# 5

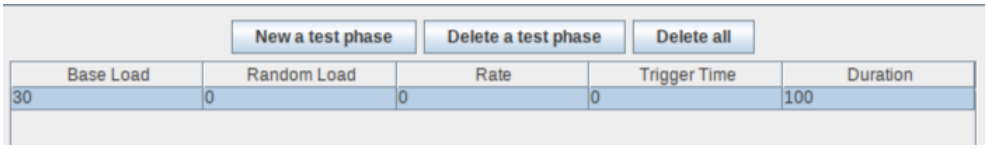
# RESULTADOS

As análises apresentadas nessa seção resumem os resultados dos apenas executado 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:

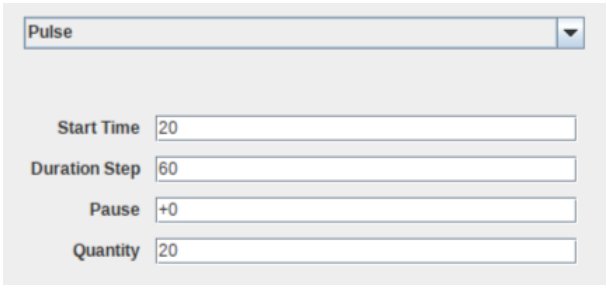
- Configuração da carga no Bench4Q;
- Configuração para modular a carga;
- Resultado da carga gerada.

A Figura 21 apresenta a primeira exemplificação gerada com o Bench4Q já com a extensão desenvolvida neste trabalho. A figura 21a 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 21b 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. Decorrido 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 influencia devido ao seu valor 0. O resultado pode ser analisado através da Figura 21c. 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*.

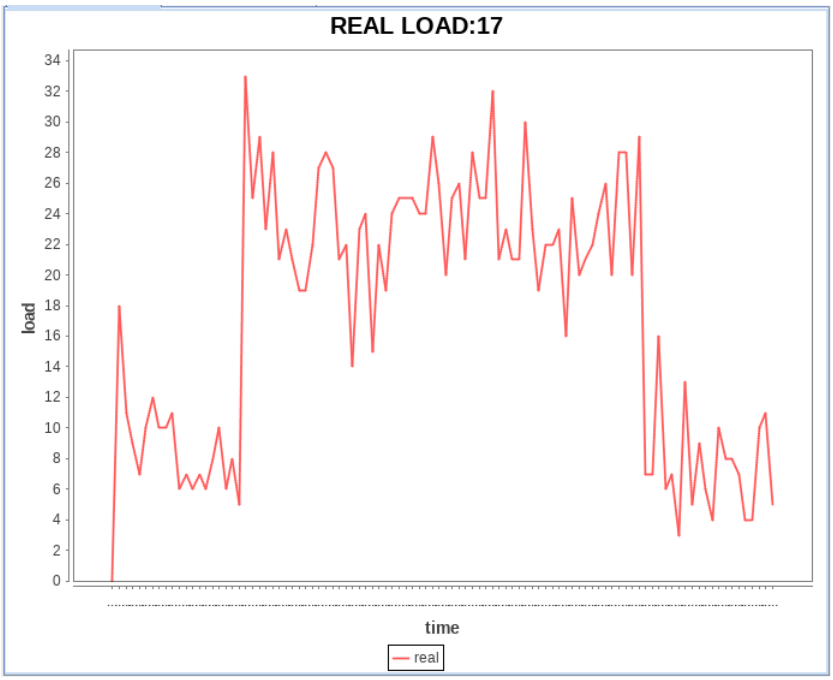
A Figura 22 apresenta os resultados dos parâmetros para o objetivo do Degrau Negativo. Os parâmetros de *Base Load* e *Duration* são os mesmos do experimento anterior: 30 EBs e 100




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




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

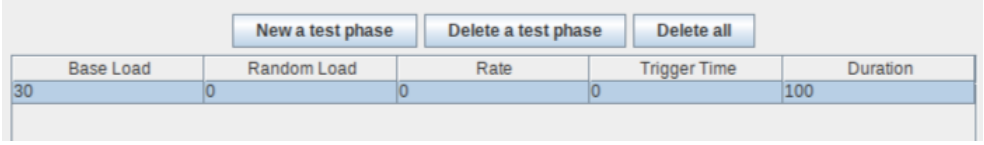



(c) Carga gerada com base nas configuração

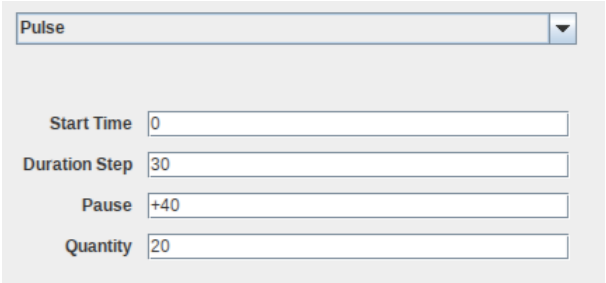
Figura 21 – Carga gerada com base na configuração: Degrau Positivo

Fonte: Elaborada pelo autor.

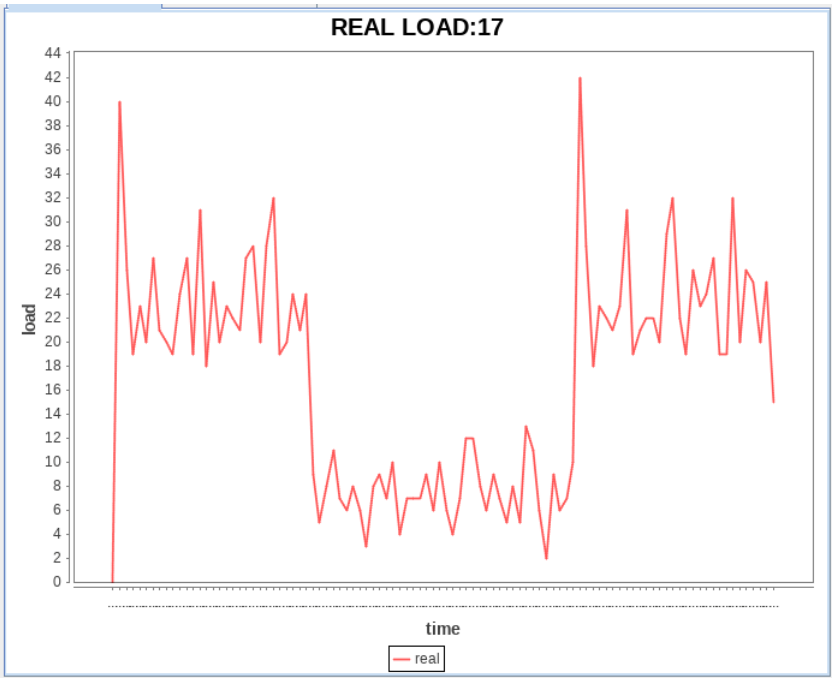
segundos de execução respectivamente, conforme apresentado na Figura 22a. Já na 23b que demonstra os parâmetros utilizados para modular a carga, o *Start Time* recebe o valor 0, assim a carga modulado iniciar com potência máxima utilizando os 30 EBs sendo 20 EBs setado no *Quantity* para reservá-los 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 22c.




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




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

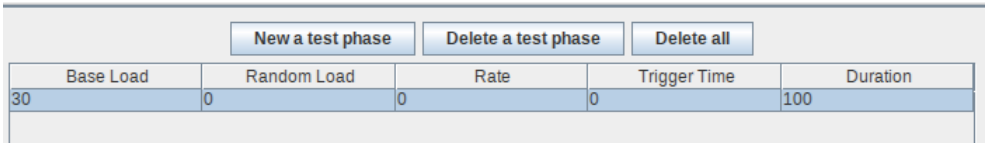



(c) Carga gerada com base nas configuração

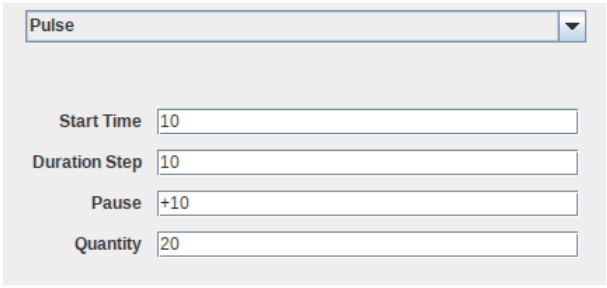
Figura 22 – Carga gerada com base na configuração: Degrau Negativo

Fonte: Elaborada pelo autor.

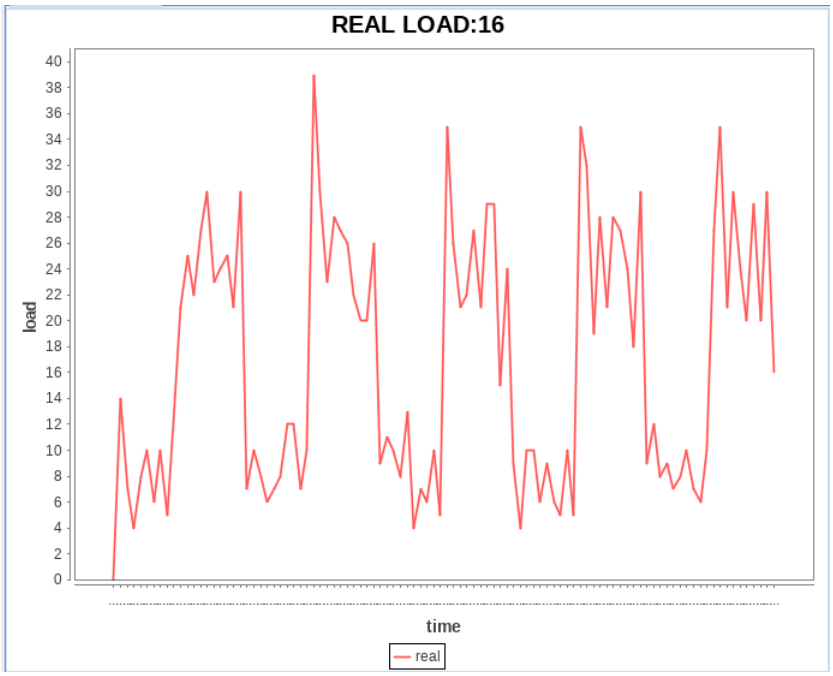
A Figura 23 apresenta o resultado da modulação de uma Onda Quadrada. Os parâmetros iniciais do Bench4Q referente a Figura 23a são os mesmos valores dos outros dois exemplos anteriores. Para gerar uma carga modulada com comportamento oscilatório como a de uma onda quadrada, os parâmetros 23b são definidos 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 configuração

Figura 23 – Carga gerada com base na configuração: Onda Quadrada

Fonte: Elaborada pelo autor.

## 5.1 Contribuição

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 **??**). Seguindo o planejamento de experimento apresentado no Capítulo 3 foi possível coletar dos dados que geram os gráficos a seguir. Vale lembrar, que foram utilizada dois níveis de cargas de trabalho, 20 e 60 EBs com um unico degrau positivo. Em ambos os casos, foi inicialmente utilizado 40% da carga até o instante de 30 segundos de experimentação, quando ocorre um crescimento brusco na carga utilizando os 60% restante da carga, e gerando um degrau positivo. A carga mantém-se

máxima, em 100%, durante 40 segundos (decorridos 70 segundo 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.

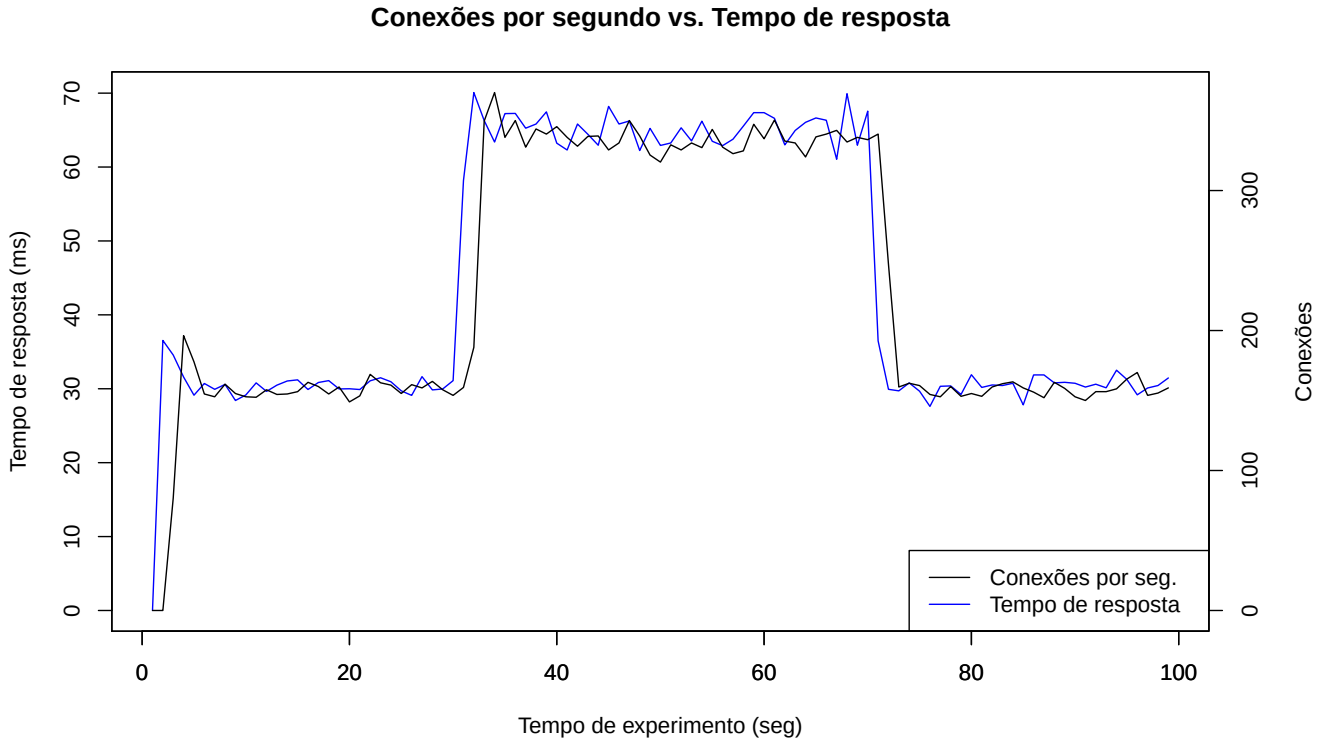



Figura 24 – Conexões por segundo vs. Tempo de resposta

Fonte: Dados da pesquisa.

É possível observar a Figura 24, 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 segundo 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 ocorre baixando de desempenho da métrica que se mantem até o final da execução. Na Figura 24 é possível perceber, em ambas as elevações, que o Tempo de Resposta reage primeiro em comparação ao número de conexões. Isso ocorre devido aos níveis da arquitetura da solução. Esse atraso é referente a dinâmica inerente ao sistema *mult-tiers*. Entretanto, na queda do volume de carga, no intervalo de 70 a 75 segundos, o Tempo de Resposta reage primeiro a queda de carga, quando comparado ao número de conexões por segundo. O tempo de resposta não se

apresenta como uma boa métrica de comparação.

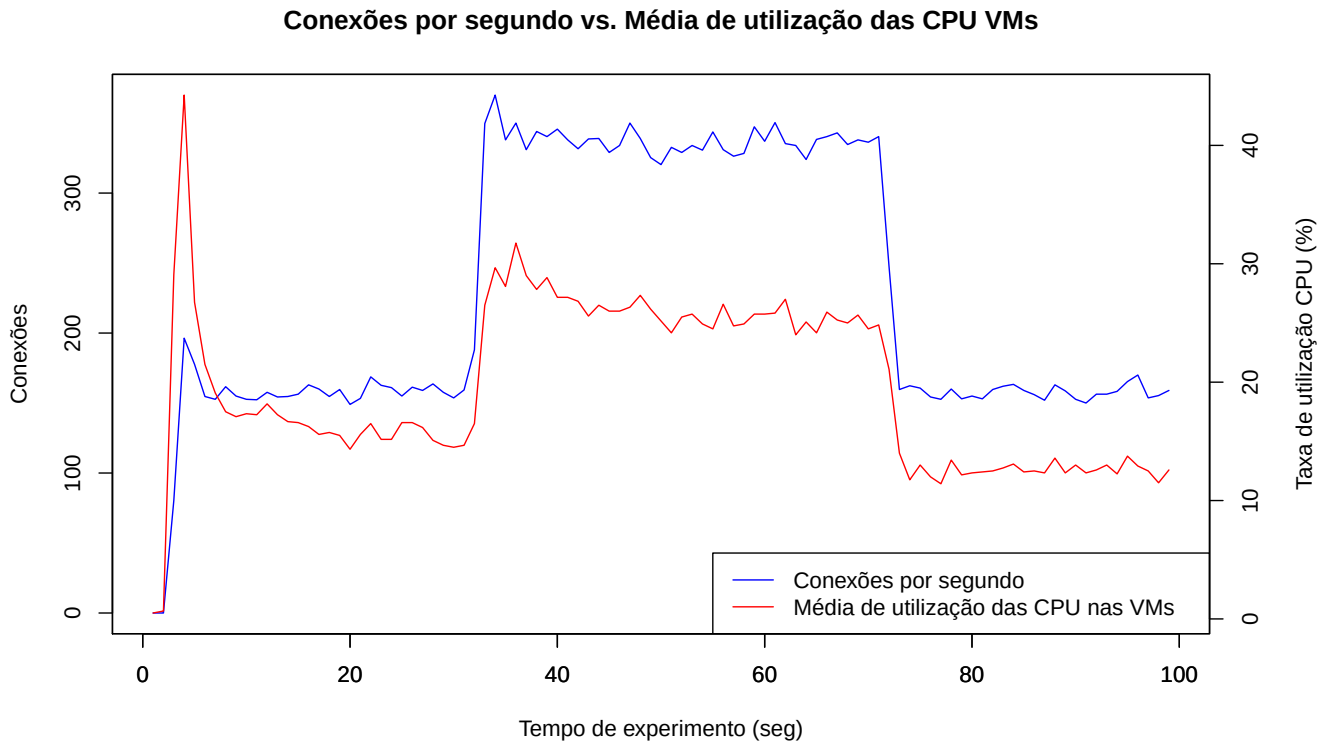



Figura 25 – Conexões por segundo vs. Média de utilização de CPU nas VMs

Fonte: Dados da pesquisa.

A Figura 25 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 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. Decorrido 40 segundos de carga máxima a utilização inicia de maneira mais enfática, atingindo os 30%. Com o decorrer do período de máximo da carga, a taxa de CPU de maneira tênue a atingir os 23% aproximadamente. O que chama a atenção na Figura 25, são os trechos de mudança no volume 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 sob mesma linha do tempo. Por fim, o trecho de 70 a 75 segundos, o cenário se inverte quando há uma redução na carga de trabalho. A taxa de utilização da CPU reage primeiro a mudança quando comparada com o número de conexões por segundo.

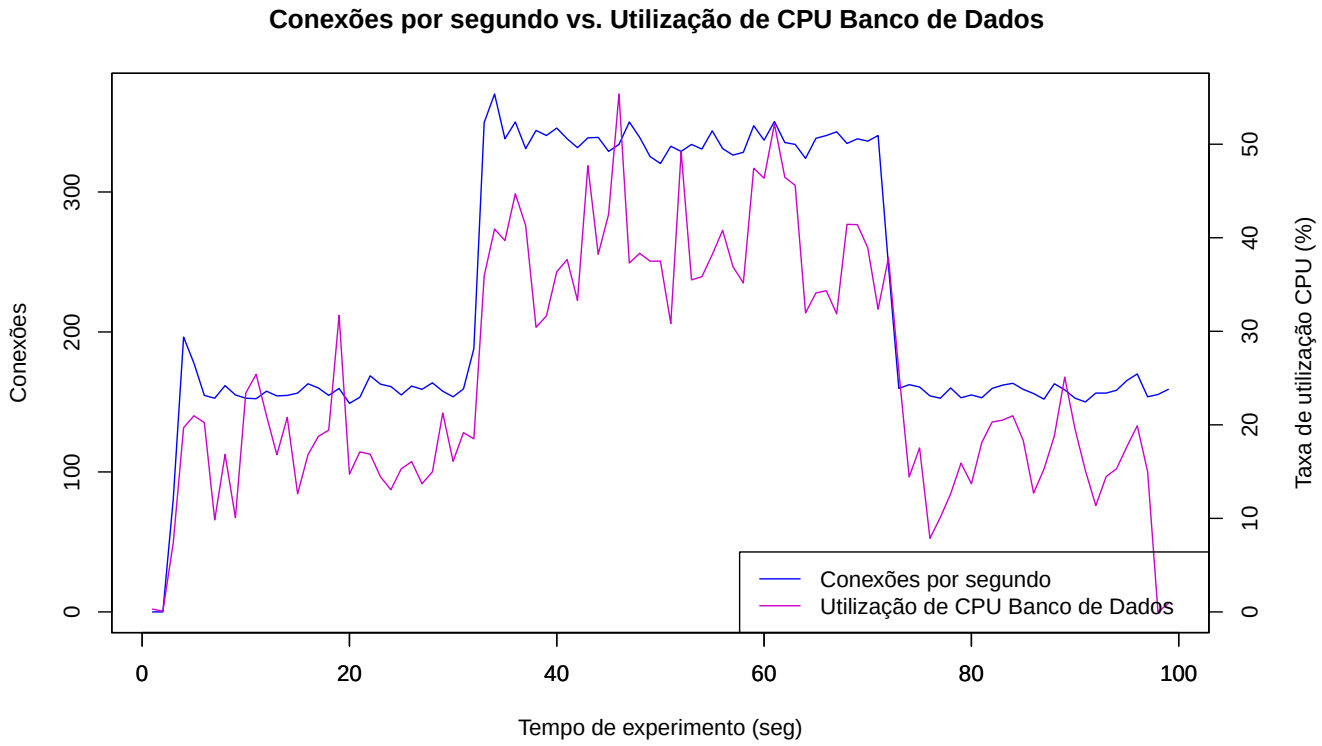



Figura 26 – Conexões por segundo vs. Utilização de CPU no banco de dados

Fonte: Dados da pesquisa.

Referente a Figura 26, onde se compara o número de conexões por segundo com a Taxa de utilização da CPU no banco de dados, podemos observar uma estocasticidade na taxa de utilização da CPU no banco de dados que está relacionada ao ser o gargalo do sistema. É possível observar a dinâmica de um sistema *multi-tiers*. Do instante 0 até a metade do experimento, o comportamento foi semelhante ao apresentado pela Figura 25, 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. Isso se deve ao fato de o banco de dados ter operações em execução mesmo quando foram finalizadas, as consultas por exemplo o encerramento de transações e *commits*.

A Figura 27 apresenta as duas métricas considerando o ambiente utilizado do experimento. Nela é possível observar novamente a falha gerada pela dinâmica intrínseca quando se lida com sistema *mult-tiers*. O tempo de resposta apresentado é medido no cliente pelo Bench4Q, sendo a camada superior da arquitetura. Já a taxa de utilização da CPU é coletada no banco de dados, estando na camada inferior da arquitetura. Na primeira metade do gráfico, de 0 a 50 segundos,

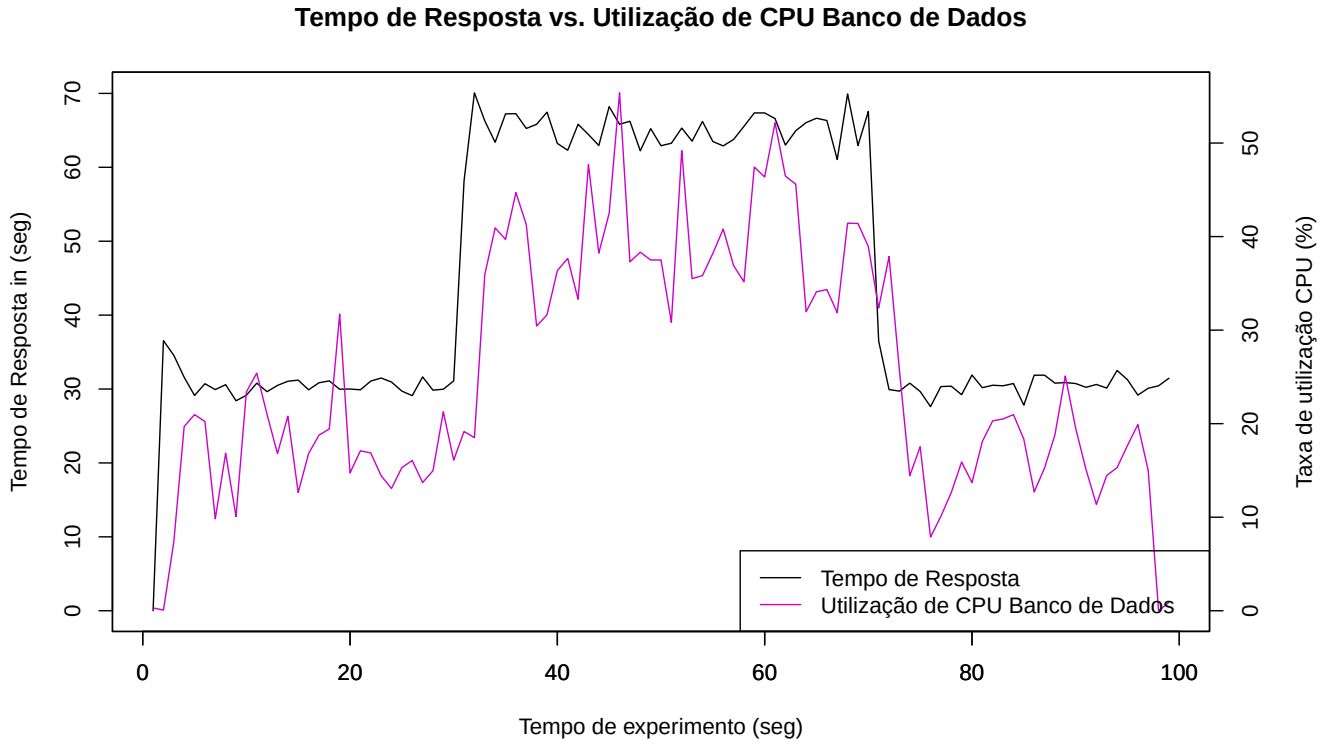



Figura 27 – Tempo de Resposta vs. Utilização de CPU no banco de dados

Fonte: Dados da pesquisa.

é possível observar o atraso das requisições mediante a transação entre todas as camadas da solução. Já na segunda metade, entre 50 segundos até o final do experimento, podemos observar o atraso novamente, porém este tem um impacto negativo para a métrica de Tempo de Resposta. No trecho de aproximadamente 75 segundos, quando há uma queda na carga de trabalho, o Tempo de Resposta reage primeiro que o consumo de CPU do banco de dados, mascarando o real desempenho da aplicação.

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 **??**) e **??**) 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 **??**). 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 **??**), 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 A;
- 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 **??**), e em paralelo a outra iniciativa de **??**) 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 esta 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 (**??**) 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

APÊNDICE

A

# 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 – USP Extension

*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 - USP** | 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

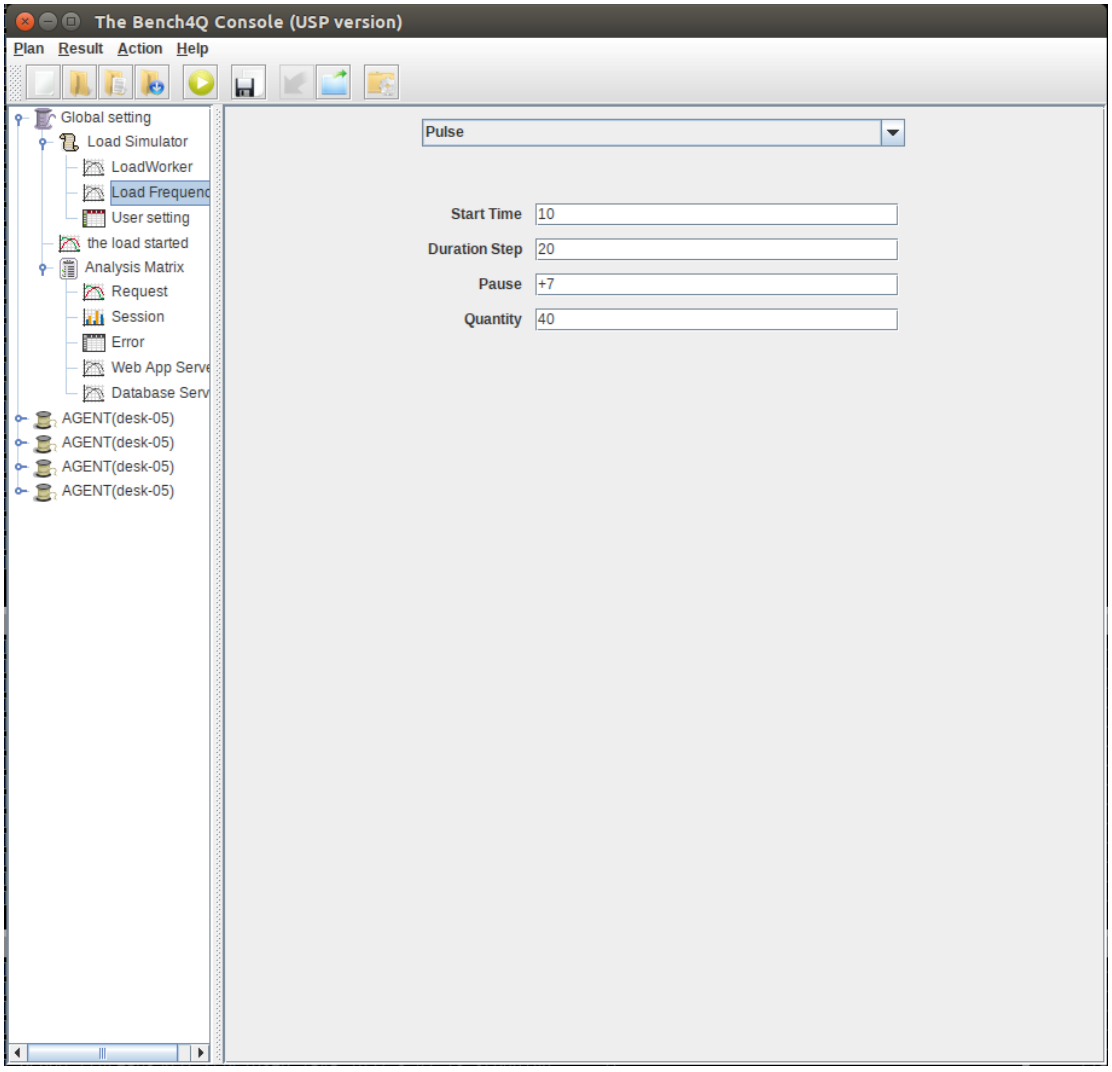
Bench4Q Tool is designed for Bench4Q benchmarking. Bench4Q Tool offers a convenient way to configure the test and analysis the test result. Now, it is possible Workload Modeling

## 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.

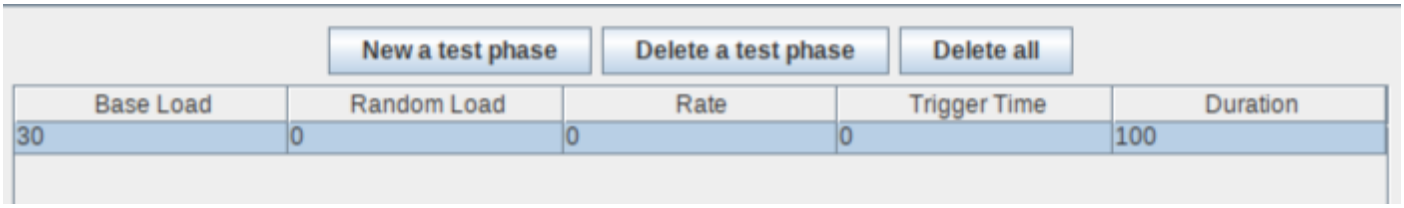
- **Start Time:** A period of time that the workload is modulated, characterizing the behavior of the change requests programmed manner;
- **Step Duration:** as shown in Chapter 2, the modulation will be displayed on Degray way;
- **Pause:** Period of interruption / pause after the load planning time;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

# 4 Getting Started

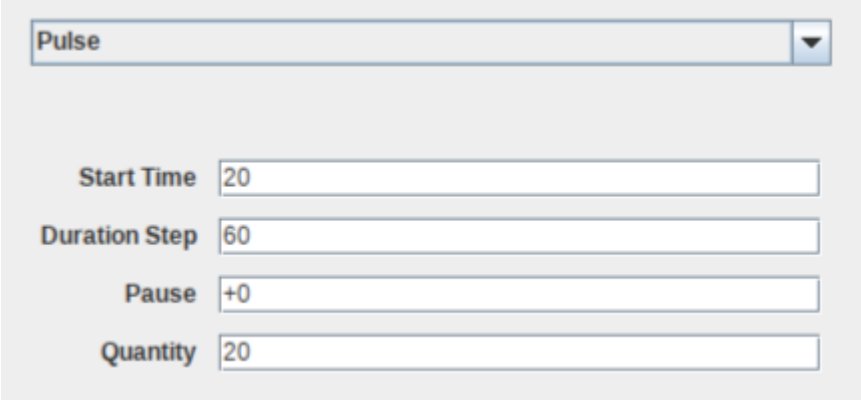
In order to run a Bench4Q test, you will need to get some basic information of the Bench4Q Tool.

## 4.1 Execution workload model

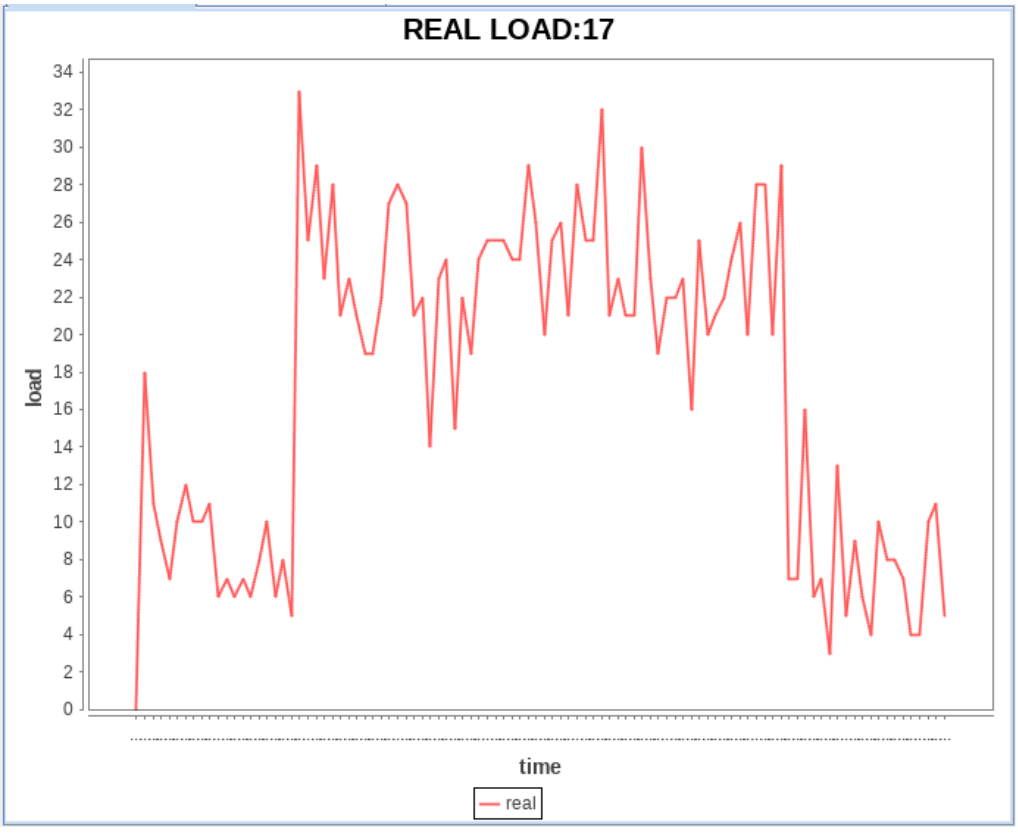
Load Work the tab, enter the following values:



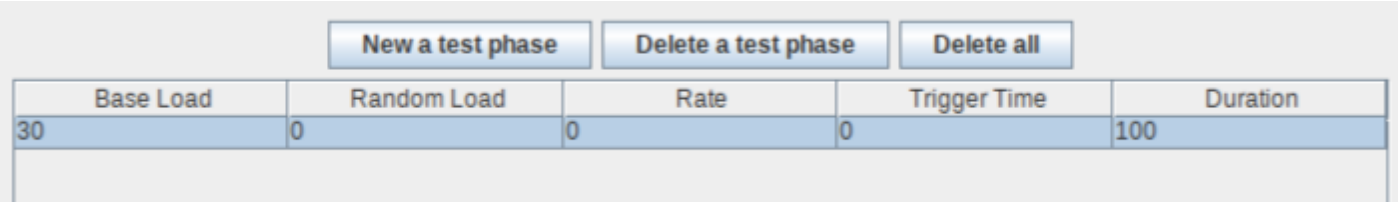

Na nova tab, Load Frequency, insirá os parâmetros utilizado para fazer o teste conforme a figura a seguir:

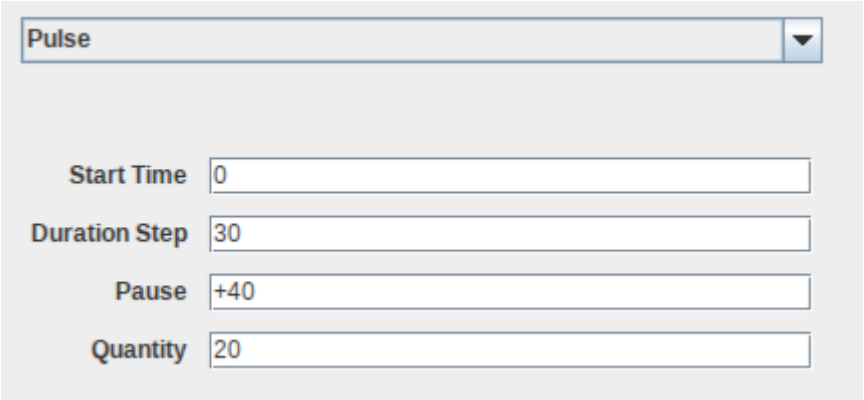


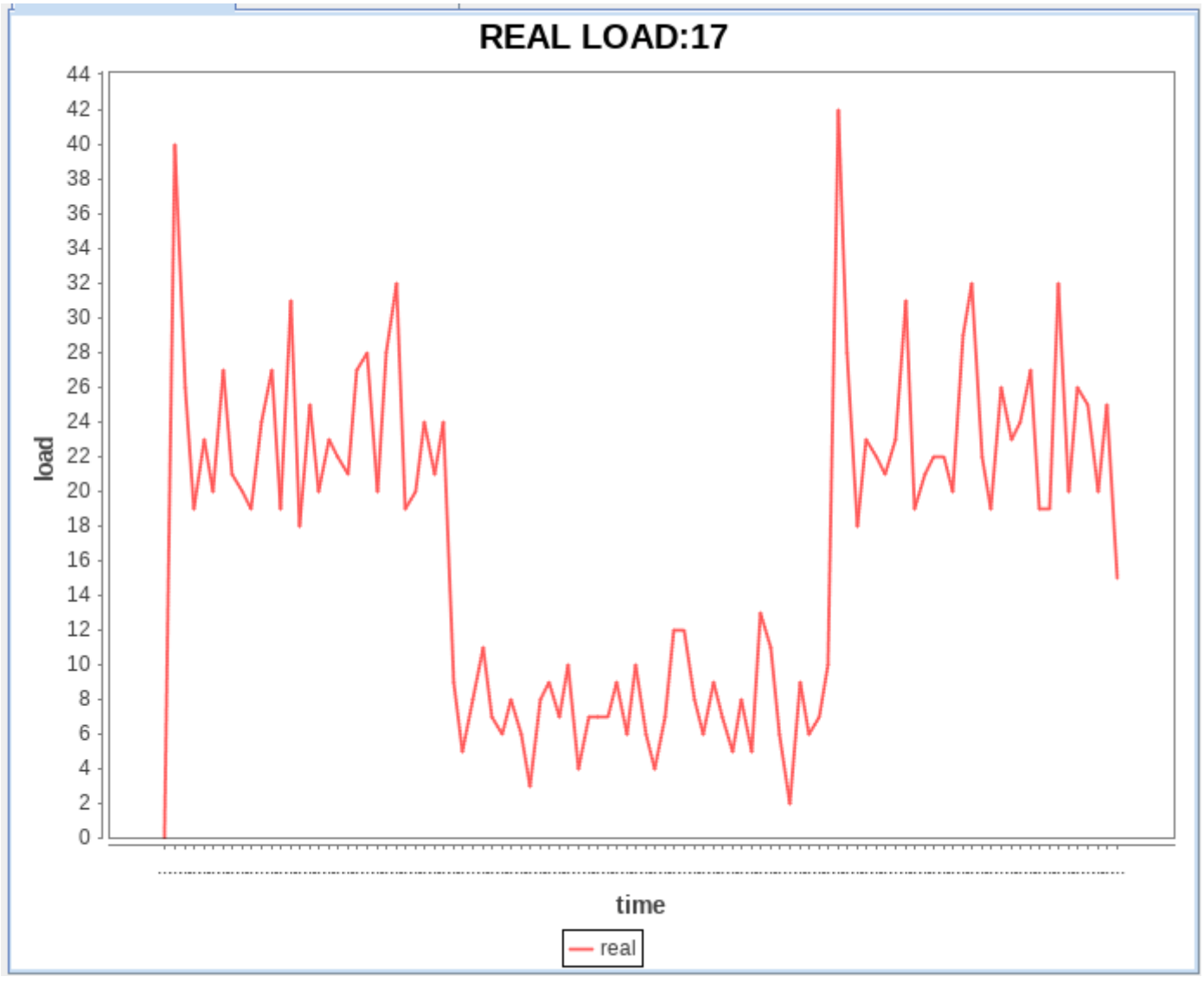

The result can be analyzed by the next figure, this graph is native Bench4Q itself, which shows the load behavior over time. Although stochasticity of the load is modulated as programmed, that is characteristic of stochasticity Bench4Q, in order to maintain a more realistic behavior with clients accessing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

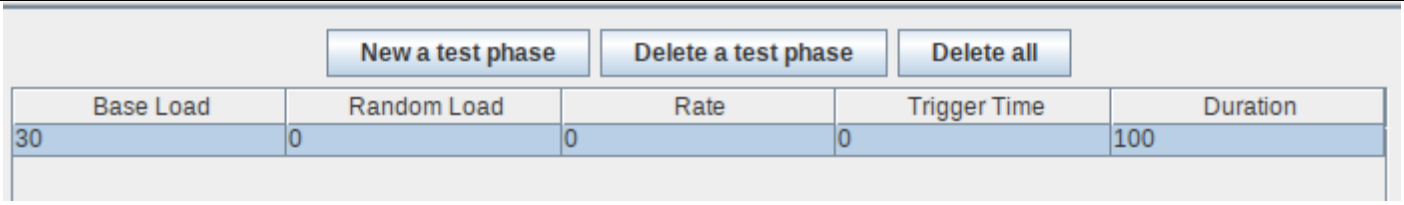


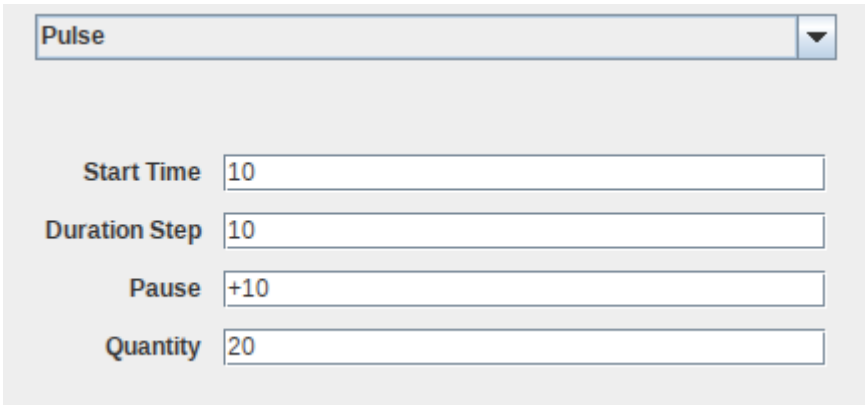
## 4.2 Other examples

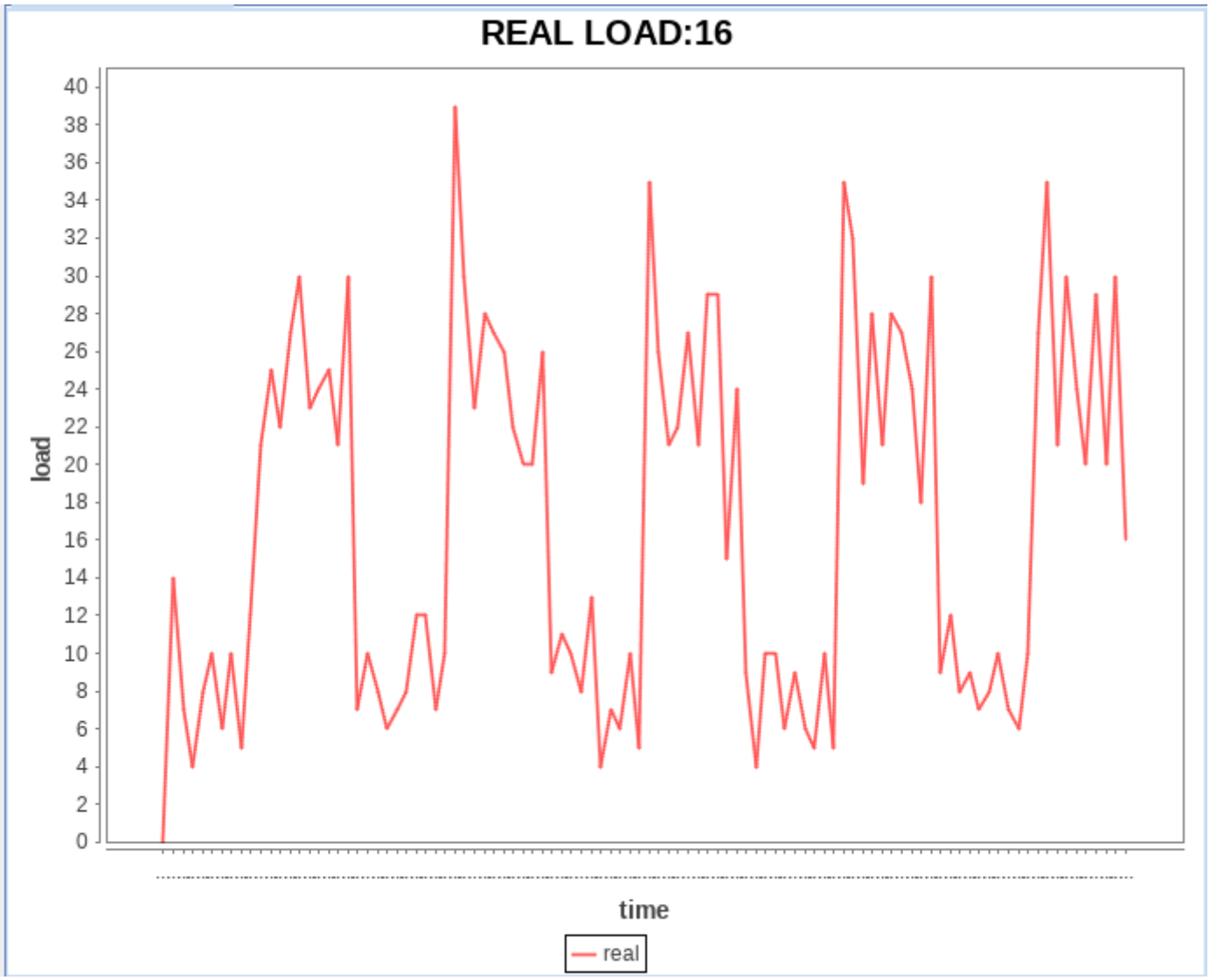



New a test phase
Delete a test phase
Delete all
Base Load
Random Load
Rate
Trigger Time
Duration
30
0
0
0
100


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


REAL LOAD:16
load
time
real