Geração de carga de trabalho transiente para aplicações de *e-commerce* multicamadas

Lourenço Alves Pereira Júnior[1,2], Flávio Luiz dos Santos de Souza[2,3], Edwin Luis Choquehuanca Mamani[2], Francisco José Monaco[2]

[1]Instituto Federal de São Paulo — IFSP

ljr@ifsp.edu.br

[2]Universidade de São Paulo — USP

{ljr,flaviols,edwin,monaco}@icmc.usp.br

[3]Universidade de Franca — UNIFRAN

flavio.souza@unifran.edu.br

Abstract. *Considering the great adoption of cloud computing as operational platform by e-commerce retailers and its dynamical environment setup, workload on this systems is basically time-varying. Caused by user bahavioral patterns, as well as flash crowds and virals, the system trespass differents operational regions leading to transients. In a scenario where transients is commonplace, the proper study of this phenomenon is important. Although, in spite of many benchmarks whose purpose is load systems with synthetic work, it is not usual to find an interface to express how workload must change on the fly during an experimental execution, as a matter of concern. Thus, this papers presents the implementation of a time-varying workload generator as an extension of the Bench4Q tool. The results point towards the importance of applying such kind of performance tests by allowing the apprisal of system's transients when using IBM DB2 and PostgreSQL, revealing the dynamic behavior of the performance metric response time.*

Resumo. *Dado o ambiente dinâmico proporcionado por ambientes de computação e sua grande adoção como plataforma operacional de lojas virtuais (*e-commerce*), a carga de trabalho característica desses sistemas sofre flutuações ao logo de sua execução. Ocasionada por padrões de acesso dos usuários, promoções relâmpago e virais de redes sociais, a carga de trabalho imposta ao sistema o expõe a diferentes patamares de utilização. A mudança entre esses patamares faz com que regimes operacionais transientes sejam comuns, o que torna seu estudo importante. No entanto, apesar de existirem muitas ferramentas de* benchmark*, poucas tem a preocupação de disponibilizar uma interface em que seja possível expressar as variações que devem ocorrer durante os experimentos. Assim, este trabalho apresenta a implementação de um gerador de carga de trabalho variante no tempo. Os resultados evidenciam a importância de se aplicar testes de desempenho com uma ferramenta como essa, proporcionando a apreciação de diferentes padrões comportamentais ao utilizar os banco de dados IBM DB2 e PostgreSQL, revelando o comportamento dinâmico da métrica tempo de resposta.*

1. Introdução

A caracterização e a geração de carga de trabalho para sistemas computacionais é uma área de pesquisa consolidada, abrangendo vários tipos de aplicações como, por exemplo, cargas para servidores web, redes sociais, serviços de distribuição de vídeo, dispositivos móveis, computação em nuvem e *datacenter* [Calzarossa et al. 2016]. Observa-se que cada vez mais aplicações Web são migradas para ambientes de computação em nuvem, cuja motivação se dá pela elasticidade, que permite o correto dimensionamento da capacidade computacional em função da demanda [Qu et al. 2016]. As flutuações na demanda de serviços são cada vez mais frequentes (oriundas de padrões de utilização de serviços, campanhas virais, promoções relâmpago etc.), e novas abordagens são experimentadas em ambientes reais de empresas de grande porte, como é o caso da Netflix [Yuan et al. 2013]. Nesse cenário, destaca-se a natureza variante das cargas de trabalhos das aplicações atuais, de forma que começa a ser um requisito para avaliação de desempenho propostas de mecanismos que permitam expressar alterações na carga de trabalho em execução.

A abordagem tradicional para avaliação de desempenho consta da representação de sistemas com modelos regressivos ou baseados em Teoria de Filas que capturam o comportamento do sistema em regime estacionário [Jain 1990, Menasce et al. 2004]. De forma análoga, a caracterização da carga de trabalho é feita pela obtenção de traços de execuções reais, sua análise e respectivo ajuste da funções de probabilidade estatística que melhor adéqua (*fit*) aos dados. Essa caracterização também é estacionária, e sua aplicação é realizada por meio de um gerador de números pseudo-aleatório [Feitelson 2015]. Nesses casos, a prática consiste em descartar os dados de regime transiente e aproveitar somente aqueles estacionários. No entanto, no domínio deste trabalho, as aplicações estão suscetíveis a cargas variantes no tempo e, por consequência, estados operacionais transientes são comuns e podem conter informações importantes.

Iniciado em meados da década de 1990, um novo paradigma de implementação de controladores de recursos de sistemas computacionais baseado em Teoria de Controle vem sendo aplicado até os dias atuais nos mais diversos domínios de aplicação, como é o caso de sistemas Web de grande porte [Hellerstein et al. 2004b]. Esse paradigma tem a vantagem de ser possível representar o transiente dos sistemas computacionais, trazendo todo um arcabouço numérico que possibilita a utilização de métricas de desempenho antes inéditas em ciência da computação. Por exemplo, tempo de assentamento (*settling time*), sobre-passagem (*overshoot*) e ganho. No entanto, observa-se uma abordagem pontual (ou *ad-hoc*) para solução de problemas [Huang et al. 2014]. Ressalta-se a necessidade da execução de experimentos específicos para a correta modelagem de sistemas para aplicação de Teoria de Controle.

Como proposto por [Mamani et al. 2015], pode-se derivar um arcabouço genérico para avaliação de desempenho de sistemas computacionais, de modo que seja possível estabelecer um conjunto de requisitos para sua efetiva implementação [Pereira et al. 2015a], bem como tratar adequadamente os dados experimentais [da Luz et al. 2016]. Assim, pode-se utilizar de um conjunto de ferramentas para experimentação e análise genéricas o suficiente para serem aplicadas em diversas aplicações. Ressalta-se principalmente o caso em que o desempenho seja em função da carga de trabalho [Pereira 2016]. É importante mencionar que, para este tipo de avaliação, o sistema é modelado como um sistema dinâmico o que permite um novo olhar sobre a geração de carga traba-

lho [Pereira et al. 2017]. O modelo conceitual MEDC (*Monitor, Effector, Demand, and Capacity*) [Pereira et al. 2015a, Pereira et al. 2015b] especifica responsabilidades que permitem a realização de um modelo dinâmico do sistema. Consistindo do monitoramento periódico de métricas de desempenho que são passadas para as responsabilidades *demand* e *capacity* a fim de que seja possível implementar flutuações na demanda do serviço em modelagem e aplicar políticas de ajuste de capacidade em tempo de execução. *Effector* aplica de fato as alterações no sistema, podendo inserir elementos que deturpam a dinâmica do sistema (como adição de falhas).

Dada a importância desse tema, evidenciado pela característica de elasticidade da computação em nuvem e a escassez de ferramentas que permitem gerar transientes de forma sintética em serviços computacionais, neste trabalho procuramos entender como gerar tais variações no regime operacional. Este artigo, portanto, tem o objetivo *1) propor uma interface que permita a especificação do comportamento de uma carga de trabalho que varia durante o experimento e 2) demonstrar sua utilização por meio de um estudo de caso com os SGBDs PostgreSQL e IBM DB2*. Os resultados foram relevantes, expondo que o transiente pode levar o sistema condições extremas de curta duração. Observamos que transiente é absorvido pelo sistema e segue uma certa "inércia". Seu negligenciamento pode levar à especificação de políticas de *auto-scaling* que desconsideram o comportamento do sistema, o que implicaria em aumento de custos para a aplicação.

A seguir, na Seção 2 sao discutidos trabalhos relacionados. Na Seção 3 é a presentado o OnlineBench4Q e as alterações necessárias para que o gerador de carga de trabalho variante no tempo. Na Seção 4 é detalhado como a geração de carga de trabalho é feita. A Seção5 apresenta um estudo de caso com IBM DB2 e PostgreSQL. Por fim, na Seção 6 conclui-se este estudo.

2. Trabalhos relacionados

Os serviços de carga automatizada em ambientes de nuvem são muito comuns como pode-se citar o Loader.io [Loader.io 2017] e NewRelic [NewRelic 2017]. Eles não implementam funcionalidades de permitem expressar cargas transientes com características periódicas e formatos específicos (senoidal, degrau, PRBS etc.) que auxiliam no processo de identificação do sistema — NewRelic e Loader.io apresentam uma demanda crescente progressiva (rampa). De mesma forma, trabalhos científicos para geração de carga de trabalho distribuída e em nuvem ainda não contemplam nativamente a alteração das características da carga de trabalho em tempo de execução [Ferreira et al. 2016]. O presente trabalho tem a preocupação de fornecer uma interface flexível na qual o usuário seja capaz de especificar como as variações na carga serão executadas em ambiente de *e-commerce*.

Kraken, uma ferramenta de teste de carga para *datacenters* do Facebook [Veeraraghavan et al. 2016], realiza testes realísticos ao permitir que uma fração da carga real do sistema em produção seja direcionada a um *cluster* e assim verificar qual a carga máxima suportada pelo sistema. Os resultados evidenciam que a capacidade individual de uma máquina não corresponde proporcionalmente a capacidade de todas as máquinas no *cluster*. Destacaram que os resultados dos testes serviram como *feedback* para a equipe de desenvolvimento promover melhorias. O trabalho do Facebook corrobora para o presente artigo mostrando a relevância do tema, uma vez que descrevem a presença de transientes no ambiente real deles e as alterações realizadas não impactam

instantaneamente no desempenho observado. O contexto e a motivação para nossa proposta estão alinhados com o que foi descrito nos experimentos realizados com o Kraken. A saber, um cenário com comportamento emergente (o desempenho do todo é resultado da combinação das partes) e transiente apreciável. Nosso trabalho difere em relação à carga de trabalho (nossa sintética e a deles a carga real do sistema), à escala (nossa célula computacional é composta por apenas 24 máquinas físicas) e à aplicação (deles rede social e nosso uma aplicação de *e-commerce*).

Trabalhos como [Babu et al. 2009] e [Zheng et al. 2009] exploram a execução de experimentos em ambientes reais em que a carga é oriunda dos próprios clientes. Isso é um ponto positivo, no entanto, a abordagem é concentrada em produzir resultados que respondam a questões do tipo "e-se" (*what-if*). O contexto de nossa proposta é que experimentos *off-line* podem ser realizados a fim de se derivar um modelo capaz de representar o desempenho sistema em diferentes cenários e com diferentes cargas. O princípio é que, após a obtenção de um modelo de Função de Transferência, simulações são executadas e seu resultado é acurado o suficiente para que não seja necessária a execução de casos *what-if* no sistema real, evitando custos para a obtenção das respostas. Uma limitação desta proposta é que o desempenho pode ser predito em uma região de operação especificada. Detalhes desta abordagem podem ser observados em [Pereira et al. 2017] e [Yang and Liu 2012]. Nossa proposta também serve como ferramental para a expressão dos experimentos específicos executados de forma *off-line*.

Ao focar em soluções específicas para aplicação de *e-commerce*, destaca-se o TPC-W, um *benchmark* para avaliação de sistemas de *e-commerce* [Menasce 2002], que utilizada o grafo CBMG (*Custormer Behavioral Model Graph*) [Menascé et al. 1999] para especificar as transições que um cliente realiza no sistema em teste. O Bench4Q [Zhang et al. 2011] é uma extensão do TPC-W que adiciona métricas baseadas em sessões de usuários para garantir Qualidade de Serviço (QoS). O presente trabalho faz uso do Bench4Q como base de implementação de um *benchmark* capaz de gerar carga variante no tempo. Tanto o TPC-W quanto o Bench4Q não oferecem suporte adequado para análise de desempenho transiente.

Como foi evidenciado, o presente trabalho diferencia-se dos demais por dois motivos: 1) o contexto de sua aplicação, avaliação de desempenho não estacionário e 2) sua flexibilidade de configuração permitindo especificação de cargas de trabalho clássicas para identificação de sistemas (degrau, rampa, senoidal, trem de pulsos e PRBS), bem como especificadas pelo usuário. A implementação segue o modelo de requisitos MEDC aplicada ao Bench4Q, dando origem ao OnlineBench4Q destalhado a seguir.

3. Implementação do gerador de carga para o OnlineBench4Q

Com o proposito de estimular o sistema a apresentar a sua dinâmica e analisar o desempenho transiente do sistema, o presente trabalho estende o gerador de carga de trabalho do *benchmark* Bench4Q. O Bench4Q é uma extensão do TPC-W [Menasce 2002], e tem por objetivo o *tuning* de aplicações *e-commerce* com requisitos de QoS. 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 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

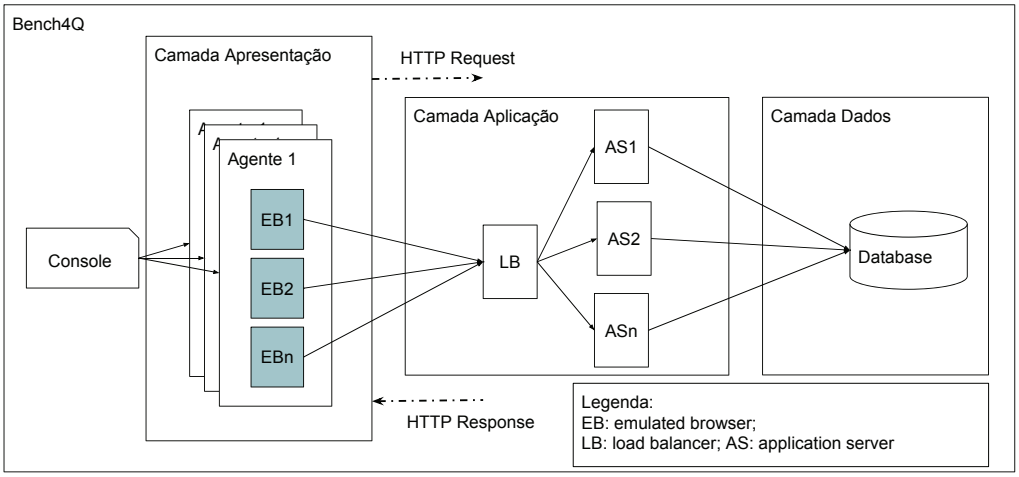
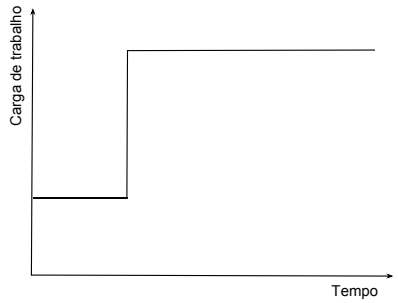


Figura 1. Arquitetura do Bench4Q. Console é uma interface de configuração de experimentos. A camada de apresentação é formada por agentes que possuem *browsers* emulados que fazem e recebem requisições ao/do sistema em teste (SUT). A camada de aplicação é composta por um balanceador de carga que repassa as requisições aos servidores de aplicação. A camada de dados é instanciada por um SGBD. O SUT é composto pelas camadas Aplicação e Dados.

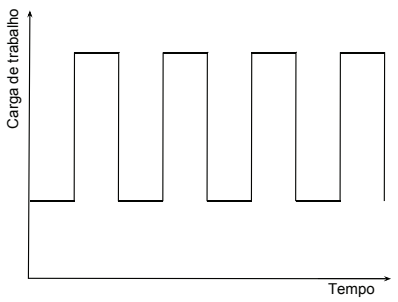
qual é possível ajustar separadamente as configurações para cada agente. Esses agentes geram carga (requisições HTTP) para o servidor de aplicação onde está hospedado o *e-commerce*, como ilustrado na Figura 1. Os resultados da avaliação de carga aplicada ao *e-commerce* são coletados pelo Console, que apresenta alguns gráficos, os quais facilitam a interpretação da avaliação mediante as diretrizes do TPC-W [Zhang et al. 2011].

A ferramenta é composta por três partes: Console, Agente e SUT (Figura 1), e também disponibiliza interfaces para o monitoramento de recursos para o servidor de aplicação e para o banco de dados; este monitoramento inclui CPU, memória, rede e tempo de resposta. O `console` é a entidade que recebe as configurações do experimento, permitindo que seja especificada a quantidade de `agentes` a serem instanciados, bem como a quantidade de *browsers* emulados (EB — *emulated browsers*) em cada agente. Os EBs comportam-se como clientes para o sistema em estudo (SUT — *System Under Test*). No entanto, o Bench4Q não é capaz de modular a carga de trabalho gerada a fim de permitir observação, análise e estudo do comportamento dinâmico do sistema. A proposta da extensão do Bench4Q é manter o padrão de usabilidade e possibilitar a modulação da carga de trabalho atrás de uma interface gráfica. Sendo assim, com o preenchimento de um conjunto de parâmetros será possível a geração modulada da carga:

- **Tempo de planejamento de carga:** Um período de tempo em que a carga de trabalho é modulada, caracterizando a mudança do comportamento das requisições de maneira programada;
- **Tipo de modulação:** conforme as funções ou sinais propostos por [Hellerstein et al. 2004b];
- **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.



(a) Formato Degrau. Pode ser positivo ou negativo.



(b) Trem de pulsos. Pode alterar a duração e a amplitude dos pulsos.

Figura 2. Modulação das cargas. O formato degrau permite realizar uma alteração brusca e duradoura. O trem de pulsos permite a criação de oscilações na carga de trabalho. A rampa pode ser derivada do trem de pulsos ao alterar progressivamente as amplitudes e manter fixa a duração das mudanças.

Através dessa interface é possível parametrizar a modulação da cargas conforme os exemplos apresentados na Figura 2. Esses exemplos são derivados das funções apresentadas por [Hellerstein et al. 2004b]. Com esses tipos de variações em tempo de execução, é possível estimular o sistema de maneira a expor sua dinâmica. É importante criar cargas de trabalho que possam ser utilizadas em estudos de avaliação de desempenho não estacionária e emular alterações no regime operacional da aplicação.

A solução proposta é simples e parte da especificação de uma unidade mínima de perturbação denominada pulso. Com ela é possível especificar o **início** em que a perturbação ocorrerá, sua **duração**, quanto tempo em **pausa** e a **amplitude** da perturbação. Deve-se considerar que a execução e um experimento iniciará em uma patamar estacionário de carga e após o tempo estipulado em *início* a programação das perturbações acontecerão. Assim, a especificação de uma perturbação do tipo degrau (Figura 2(a)), carga em que há uma perturbação que dura até o final do experimento, é feita escolhendo um instante de tempo para seu início, sua duração em conformidade com o tempo de execução do experimento, pausa como nula (valor zero) e amplitude a intensidade de carga de trabalho a ser adicionada ao sistema. A unidade mínima serve como bloco (*building block*) para a especificações de outras cargas de trabalho como: pulso unitário, trem de pulsos, PRBS etc. O PRBS (*pseudo-random binary signal*) é uma sequência de pulsos com variações na duração e tem como característica comportar-se como uma carga de trabalho neutra e ao mesmo tempo capaz de expor a dinâmica do sistema em estudo, maiores detalhes disponíveis em [Ljung 1999].

Ao ler a configuração do experimento, o `console` calcula a quantidade máxima de EBs a serem utilizados no experimento — denominada `tEBs`. Na sequência, instancia os `agentes` e cria uma quantidade de EBs balanceada em cada agente. Os EBs possuem dois estados básicos: ativos e inativos. Após a instânciação inicial, um subconjunto de `tEBs` é configurado no estado de ativo e o restante como inativo. Após o período inicial, alterna-se aqueles em estado inativo para ativo, cuja consequência é aumentar a carga. De modo análogo, pode-se diminuir a quantidade de EBs ativos na intenção de impor uma carga menos severa ao sistema. A Figura 3 ilustra a criação dos dois grupos de modo que os EBs ativos e inativos transitam de um grupo para outro.

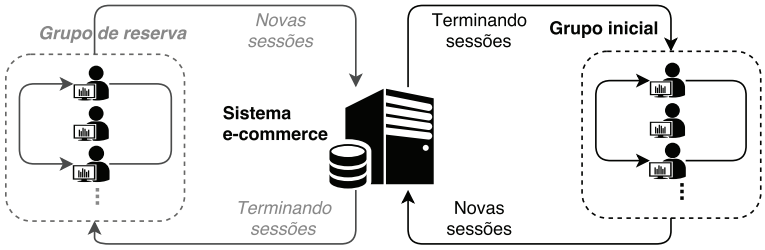


Figura 3. O mecanismo de modulação de carga prevê a criação de dois grupos: um contendo clientes ativos e outro com inativos. Durante a execução do experimento os clientes são alternados entre os grupos o que gera aumento/diminuição na carga de trabalho imposta ao sistema.

A validação desse modelo de geração de carga foi feito através da implementação de um protótipo denominado OnlineBench4Q. O termo *online* refere-se ao fato de ser possível alterar a estacionariedade da carga de trabalho em tempo de execução. Como base para sua implementação foi utilizado o modelo conceitual MEDC [Pereira et al. 2015a]. A alteração descrita neste artigo corresponde à responsabilidade *demand* do MEDC. A implementação completa conta com elasticidade de recursos computacionais (*capacity*), gerenciamento eficiente dos estados das máquinas virtuais (*effector*) e monitoramento integrado para o ambiente controlado de testes. O detalhamento da implementação encontra-se em [Mamani 2016, Souza 2016].

4. Geração da carga de trabalho

Como uma extensão do Bench4Q, o OnlineBench4Q adiciona as responsabilidades especificadas pelo MEDC e aproveita as funcionalidades da versão original. Dessa forma, iniciamos com a descrição das propriedades da carga de trabalho sintética gerada pelo Bench4Q que corresponde àquelas utilizadas em nossos experimentos de validação.

Conforme descrito em [Zhang et al. 2011], o Bench4Q utiliza configurações para descrever como a carga de trabalho impactará no SUT. O primeiro parâmetro diz respeito aos clientes que geram a caga, podendo ser fechada, com um número fixo de clientes que realizam requisições e recebem respostas até o final do experimento, e aberta, com a criação de um cliente que realiza um número k de requisições e depois deixa o sistema. Nossos experimentos baseiam-se na configuração aberta, com o intuito de representar o comportamento típico de um site de *e-commerce* que recebe clientes ao longo de sua operação.

Outro parâmetro para a carga de trabalho diz respeito à quantidade de operações de navegação e compras realizadas pelos clientes. As operações de navegação tem um custo menor para o sistema, pois resultam em consultas, cujos resultados podem ser armazenados temporariamente (*cache*) e não requerem tratamento especial para lidar com concorrência, haja vista que o tipo de sistema de gerenciamento de banco de dados mantém as propriedades ACID (atomicidade, consistência, isolamento e durabiliadde). Há três configurações padrão: `browsing` (95% navegação e 5% compra), `shopping` (80% e 20%), `ordering` (50% e 50%). Nossa configuração foi feita como `browsing`. Os valores que determinam a convergência para navegação ou compra são especificados através

Tabela 1. Configuração do hardware utilizado nos experimentos.

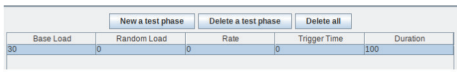
Nome	CPU (GHz)	Mem. (GB)	HD (Mb/s)	SO	#
Load Balancer	INTEL i5-3330-(3.0) x4	7,8	87,74	ClearOS 6.6	1
DB server (pool)	AMD Q6600-(2.4) x4	7,8	71,21	Ubuntu 14.04.2	8
DB server (master)	FX-8320-(3.5) x8	23	285,58	Ubuntu 14.04.3	1
Clientes	AMD Q6600-(2.4) x4	7,8	76,96	Ubuntu 14.04.2	9
Hipervisor	X5660-(2.8) x12	11	251,98	Ubuntu 15.04	1
Servidor de VMs	X5660-(2.8) x12	11	251,98	Ubuntu 15.04	4
VMs	QEMU Virtual x1	1	178,60	Ubuntu 12.04.5	-

do grafo CBMG que especifica como as transições entre as páginas ocorrerão durante as sessões dos clientes. O CBMG utilizado foi aquele especificado por padrão no Bench4Q.

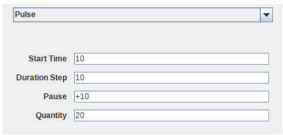
O tamanho das sessões e o intervalo entre as requisições foram configurados como uma distribuição exponencial, sem alterações aos valores padrão do Bench4Q. A taxa de tolerância a atrasos é descrita por uma distribuição normal. Ao iniciar a execução de um experimento, requisições começarão ser realizadas no ponto de acesso do SUT e assim gerarão carga ao sistema. Haverá um período de tempo no início e no final de cada experimento denotado como aquecimento (*warm-up*) e resfriamento (*cool down*), no entanto, a carga imposta ao sistema é tipicamente estacionária. A configuração da capacidade pode ser realizada de forma empírica, realizando experimentos na implementação real e verificando uma métrica de desempenho, por exemplo a taxa de utilização do sistema (CPU) em função da quantidade de requisições por segundo.

A prova de conceito ocorreu por experimentos que geraram degrau, pulso positivo e negativo e trem de pulsos. Por limitações de espaço, listamos aqui a validação por meio de trem de pulso.A descrição completa pode ser encontrada em [Souza 2016]. O experimentos realizados consideraram uma carga inicial de 30 EBs (`tEBs`) com uma perturbação do tipo trem de pulsos em que a duração dos pulsos corresponde a 10 s, e a quantidade de EBs que serão ativados/desativos é de 20 EBs. O ambiente físico foi configurado conforme Tabela 1, interligadas por três enlaces Ethernet de 1 Gbps. O Balanceador de Carga possui duas interfaces, uma para a rede cliente e outra para o *cluster* do *hypervisor* e servidores de VMs; as máquinas de BD são isoladas e podem ser acessadas a partir dos servidores de aplicação. Foram configuradas oito máquinas para agentes, um balanceador de carga, quatro máquinas virtuais (VMs) como servidores de aplicação e um banco de dados. A Figura 4 apresenta os resultados obtidos. As imagens apresentam o conteúdo de telas do OnlineBench4Q geradas a partir do `console`. A configuração de uma carga de trabalho do tipo pulso (Figura 4(a)) e que a quantidade de requisições foi conforme especificado durante a execução do experimento (Figura 4(b)). Desse modo, conclui-se que a ferramenta executou a carga de trabalho conforme especificada.

Os resultados da validação apenas demonstram como é feita a configuração do sistema, mas não revelam sua potencial utilidade como ferramenta de avaliação de desempenho de modo a levar o sistema a condições em que seja possível apreciar seu comportamento dinâmico. Na próxima Seção tratamos desta questão e mostramos a comparação do desempenho do sistema ao alterar o banco de dados de PostgreSQL para IBM DB2.

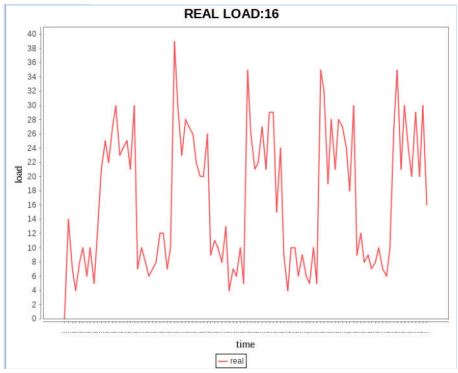


Configuração da carga no Bench4Q, para uma onda quadrada



Configuração para modular a carga como uma onda quadrada

(a) Interface gráfica do `console` para a configuração da carga de trabalho.



(b) Gráfico apresentado no `console` contendo os resultados da execução do experimento. Eixo das coordenadas o tempo e nas abssissas a quantidade de requisições por segundo.

Figura 4. Resultados da validação. O `console` apresenta uma interface gráfica para configuração da carga e após a realização de um experimento um gráfico pode ser gerado contendo os resultados obtidos.

5. Comparação de PostgreSQL e DB2

A função de persistência dos dados é importante para sistemas de *e-commerce* e serve como meio para armazenamento de transações. Esses sistemas são tradicionalmente implementados em uma arquitetura de múltiplas camadas, sendo a escolha mais simples a de três camadas. A camada de dados, que implementa a persistência das informações, representa o principal gargalo do sistema, pois seu desempenho é geralmente orientado ao de entrada e saída (*i/o-bound*). Como principal ponto de estrangulamento, seu comportamento dinâmico influencia toda a aplicação. Nesse sentido, os experimentos descritos nesta Seção têm por objetivo identificar como o comportamento transiente de dois sistemas gerenciadores de banco de dados (SGBD) impactam no desempenho percebido pelo usuário. Os SGBDs apresentam comportamentos distintos. A comparação entre os tempos de resposta deixou evidente que o desempenho do PostgreSQL foi menor e mais previsível do que o DB2. O DB2 teve um desempenho melhor do que o PostgreSQL e, devido a mecanismos adaptativos que otimizam sua operação, seu desempenho apresentou uma comportamento dinâmico significante. As configurações do sistema são:

- Quatro `agentes`;
  - Cada agente gerecia 20 EBs inicias passando para 40 EBs (carga alta);
  - As requisições utilizadas pelos EBs são do tipo *browsing*;
  - Configurações padrão do Bench4Q para geração das requisições.
- Quatro VMs foram utilizadas na camada de aplicação;
  - O SUT fornecido pelo *benchmark* foi projetado para suportar o DB2, porém foi adicionado suporte ao PostgreSQL.
- A métrica utilizada na avaliação é o tempo de resposta em milissegundos mesurada nos clientes;
- A máquina que hospedou os banco de dados é **DB Server (master)**, conforme descrito na Tabela 1;
- Cada experimento foi replicado quatro vezes, e, nos gráficos, o intervalo de confiança de 95% é representado pela sombra em torno da média.

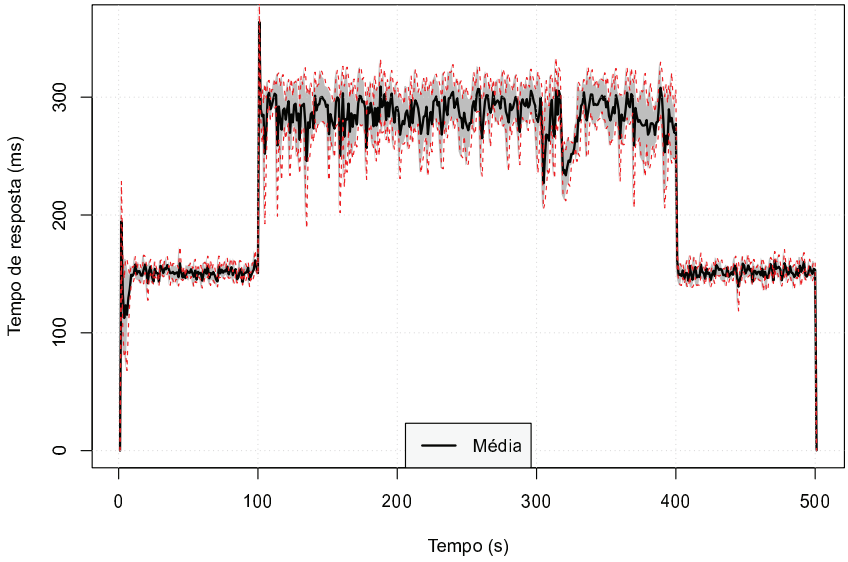


Figura 5. Tempo de resposta das requisições nos clientes utilizando o banco de dados PostgreSQL. Área cinza corresponde ao intervalo de confiança de 95%.

5.1. PostgreSQL

PostgreSQL é um conhecido banco de dados de código aberto frequentemente utilizado na computação em nuvem, e para fins de alcançar o melhor desempenho possível do banco de dados, foi utilizada a ferramenta *pgtune* [1] que permite redefinir os valores dos parâmetros definidos inicialmente para um contexto de alto desempenho. A Figura 5 apresenta o tempo de resposta de 500 s de experimento. A estratégia foi executar o *benchmark* inicialmente com a metade da quantidade de EBs disponíveis, após o segundo 100, o módulo *Demand* do *Load balancer* indica ao *Effector* que a carga deve ser dobrada. O mecanismo consiste em ativar o restante dos EBs que se encontravam em estado inativo. O resultado da mudança é uma perturbação que eleva o tempo de resposta do sistema de 150 milissegundos para um valor estacionário aproximado de 286 milissegundos.

O tempo de resposta manteve-se estacionado em 286 milissegundos aproximadamente durante o pulso. Com a execução do experimento foi verificado que a versão do PostgreSQL utilizada não apresenta um comportamento adaptativo para mudanças abruptas. No segundo 400, há a diminuição da carga para o patamar inicial do experimento. Observa-se que o desempenho é proporcional à quantidade de carga (requisições) nos dois patamares observados. Percebeu-se a ocorrência de um pico que degrada o desempenho logo após alterações bruscas, observar os instantes iniciais e por volta dos segundos 101 e 110. Muito embora, de magnitude considerável (aproximadamente 30% do patamar estacionário final), seu impacto é instantâneo, não ocasionado em maiores degradações.

5.2. IBM DB2

O DB2 é um *software* de caráter comercial, mas com uma ampla influência na área acadêmica. Vários trabalhos foram desenvolvidos visando um funcionamento adaptativo, principalmente abordagens relacionadas com a gestão automática de memória [Hellerstein et al. 2004a, Mateen et al. 2009, Storm et al. 2006].

[1] `http://pgfoundry.org/projects/pgtune`

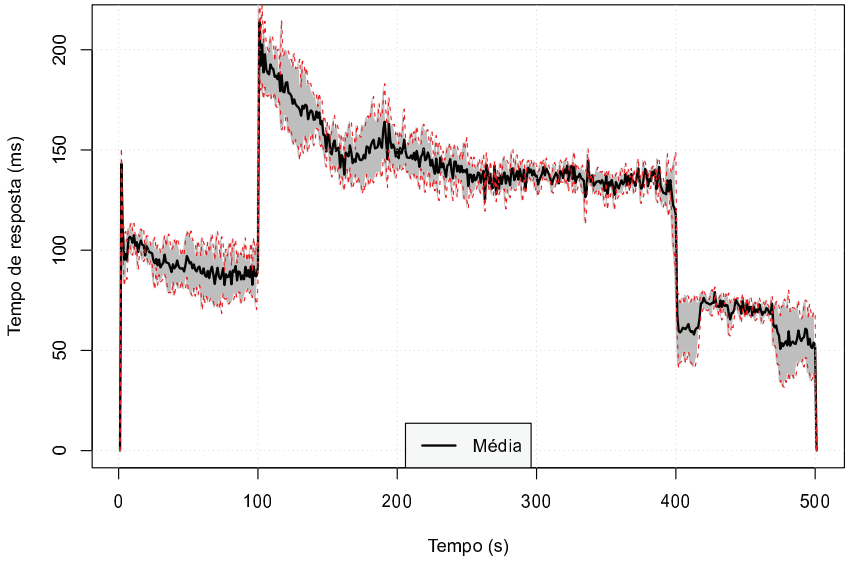


Figura 6. Tempo de resposta das requisições nos clientes utilizando o banco de dados DB2. Área cinza corresponde ao intervalo de confiança de 95%.

A Figura 6 mostra o comportamento do tempo de resposta obtido do experimento da Seção 5.1, alterando-se o SGBD para DB2. Os resultados deixam claro que há um período no qual o desempenho encontra-se em estado transiente. No início, esse período corresponde a aproximadamente 60 s. Após a perturbação, o transiente dura por volta de 140 s, dividido em duas fases: 1) uma que dura por volta de 60 s na qual o sistema experiencia uma perca de desempenho expressiva e 2) outra de 80 s em que os mecanismos internos otimizam o sistema como um todo para lidar com a carga imposta.

Na perspectiva do monitor de recursos do sistema operacional, a memória virtual ocupada pelo banco de dados aumentou de 3 GB para 11 GB após a mudança na carga de trabalho. Como a carga utilizada é do tipo *browsing*, há fortes indícios de que há *cache* de dados na memória principal. Diferentemente do PostgreSQL, não foi realizado nenhum *tuning* no banco de dados, pois a documentação explicita que o próprio *software* realiza todo o *tuning* necessário em tempo real e conforme a demanda recebida.

A carga de trabalho imposta nos intervalos (1,100) e (401, 500) são as mesmas. Assim, uma ressalva importante é que, após o degrau terminar no segundo 400, o tempo de resposta foi menor do que aquele obtido no início do experimento, mesmo que sobre condições operacionais equivalentes. Como o DB2 foi forçado a otimizar seu desempenho ao excursionar em uma região de operação mais severa, ao retornar ao patamar inicial, seu desempenho pôde ser melhor. Em outros experimentos preliminares, observamos que o DB2 mantém a característica de alterar seu desempenho durante o tempo.

Estes experimentos evidenciam a utilidade de um *benchmark* para avaliação do regime não-estacionário. Ressaltando a utilidade de um *benchmark* capaz de expor os comportamentos dinâmico de sistemas computacionais. Estes cenários não são tão particulares como aparentam, e podem ser observados comumente em ambientes de computação em nuvem, em que existem uma ampla variedade de provedores de *software* e *hardware* com restrições de desempenho diferentes.

6. Conclusões

Atualmente a maioria dos sistemas computacionais de grande porte são compostos por multicamadas, arquitetura que permite escalabilidade e amplamente adotada em aplicações web. Para estas, o planejamento de capacidade é um método que permite especificar a quantidade de recursos necessária para oferecer um determinado nível de QoS. No entanto, esse planejamento é basicamente uma decisão de longo prazo e com características estáticas. Geralmente, os recursos são determinados por métricas e parâmetros sem contar com a mudança nos patamares operacionais planejados. Desta forma, os recursos podem sofrer uma sobrecarga em situações em que há perturbação na carga de trabalho e, dependendo do caso, por levar a descumprimento dos níveis de acordo do serviço. Para este artigo, a carga de trabalho possui a característica de grande variação, fato que exige uma avaliação de desempenho focada a regime transiente.

Detalhamos os problemas e requisitos para a implementação do módulo capaz de promover flutuações na carga de trabalho do OnlineBench4Q. Foi proposto um mecanismos que permite a criação de dois grupos de clientes (EBs), para os estados ativos e inativos, em que parte desses clientes migram de um grupo para o outro, aumentando ou diminuindo a carga. Essa movimentação causa um efeito do tipo pulso na carga e serve como unidade mínima de perturbação para a construção dos mais diversos tipos de cargas de trabalho. Sua utilidade foi mostrada em um estudo de caso em que foi possível verificar o comportamento dinâmico diferente entre os banco de dados PostgreSQL e IBM DB2. Se neste artigo detalhamos o mecanismo para geração de carga variante no tempo, em [Pereira et al. 2017] realizamos a análise de resposta em frequência do OnlineBench4Q em uma configuração de porte maior. Os conceitos aqui apresentados podem ser generalizados e podem ser explorados em outros contextos, como em [Rodrigues et al. 2015].

Como trabalhos futuros, pretende-se elaborar outras cargas de trabalho como onda senoidal, especificada pelo usuário e também em função da experiência dos clientes, implementando os conceitos de abandono (*bouncing*) e convergência. No sentido da reprodutibilidade dos experimentos, pode-se criar um mecanismo que exporta as cargas de trabalho para reuso em outras ferramentas. Outro ponto a ser explorado é a avaliação do custo extra para tratar dos transientes, considerando sua efemeridade na perturbação.

Agradecimentos

Agradecemos CAPES, FAPESP, CNPq, LaSDPC/USP e IFSP pelo apoio financeiro.

Referências

Babu, S., Borisov, N., Duan, S., Herodotou, H., and Thummala, V. (2009). Automated experiment-driven management of (database) systems. In *Proceedings of the 12th Conference on Hot Topics in Operating Systems*, HotOS'09. USENIX.

Calzarossa, M. C., Massari, L., and Tessera, D. (2016). Workload characterization: A survey revisited. *ACM Comput. Surv.*, 48(3):48:1–48:43.

da Luz, H. J. F., Júnior, L. A. P., dos Santos de Souza, F. L., and Monaco, F. J. (2016). Modelagem analítica de sobrecarga transiente em sistemas computacionais por meio de parâmetros dinâmicos obtidos empiricamente. In *XIV Workshop de Computação em Clouds e Aplicações*, Salvador, BA. Sociedade Brasileira de Computação.

Feitelson, D. G. (2015). *Workload Modeling for Computer Systems Performance Evaluation*. Cambridge U. Press.

Ferreira, C. H. G., Nunes, L. H., Pereira, L. A., Nakamura, L. H. V., Estrella, J. C., and Reiff-Marganiec, S. (2016). Peesos-cloud: A workload-aware architecture for performance evaluation in service-oriented systems. In *IEEE World Congress on Services*.

Hellerstein, J., Storm, A., Surendra, M., Lightstone, S., Parekh, S., and Garcia-Arellano, C. (2004a). Incorporating cost of control into the design of a load balancing controller. *Proceedings. RTAS 2004. 10th IEEE Real-Time and Embedded Technology and Applications Symposium, 2004.*, pages 376–385.

Hellerstein, J. L., Diao, Y., Parekh, S., and Tilbury, D. M. (2004b). *Feedback Control of Computing Systems*. John Wiley & Sons, Inc., Hoboken, NJ, USA, 1 edition.

Huang, D., He, B., and Miao, C. (2014). A survey of resource management in multi-tier web applications. *IEEE Communications Surveys Tutorials*, 16(3):1574–1590.

Jain, R. (1990). *The art of computer systems performance analysis: techniques for experimental design, measurement, simulation, and modeling*. John Wiley & Sons.

Ljung, L. (1999). *System Identification: Theory for the User*. Pearson Education.

Loader.io (2017). Application load testing tools for api endpoints. `https://loader.io/`. Accessado: 31/03/2017.

Mamani, E. L. C. (2016). *Metodologia de benchmark para avaliação de desempenho não-estacionária: um estudo de caso baseado em aplicações de computação em nuvem*. PhD thesis, PPG-CCMC ICMC USP.

Mamani, E. L. C., Pereira, L. A., Santana, M. J., Santana, R. H. C., Nobile, P. N., and Monaco, F. J. (2015). Transient performance evaluation of cloud computing applications and dynamic resource control in large-scale distributed systems. In *High Performance Computing Simulation (HPCS), 2015 International Conference on*, pages 246–253.

Mateen, A., Raza, B., Hussain, T., and Awais, M. (2009). Autonomicity in Universal Database DB2. In *2009 Eighth IEEE/ACIS International Conference on Computer and Information Science*, pages 445–450. IEEE.

Menasce, D. (2002). TPC-W: a benchmark for e-commerce. *IEEE Internet Computing*.

Menascé, D. A., Almeida, V. A. F., Fonseca, R., and Mendes, M. A. (1999). A methodology for workload characterization of e-commerce sites. In *Proceedings of the 1st ACM Conference on Electronic Commerce*, EC '99, New York, NY, USA. ACM.

Menasce, D. A., Dowdy, L. W., and Almeida, V. A. F. (2004). *Performance by Design: Computer Capacity Planning By Example*. Prentice Hall, Upper S River, NJ, USA.

NewRelic (2017). Newrelic: Application performance management and monitoring. `https://newrelic.com/`. Accessado: 31/03/2017.

Pereira, L. A. (2016). *Uma abordagem baseada em resposta em frequência para modelagem e avaliação de desempenho não estacionária em sistemas computacionais*. PhD thesis, PPG-CCMC - ICMC-USP.

Pereira, L. A., Mamani, E. L. C., Santana, M. J., Santana, R. H. C., Nobile, P. N., and Monaco, F. J. (2015a). Non-stationary simulation of computer systems and dynamic

performance evaluation: A concern-based approach and case study on cloud computing. In *Computer Architecture and High Performance Computing (SBAC-PAD), 2015 27th International Symposium on*, pages 130–137.

Pereira, L. A., Mamani, E. L. C., Santana, R. H. C., Santana, M. J., and Monaco, F. J. (2017). Análise de resposta em frequência para modelagem e geração de carga de trabalho em aplicações de *e-commerce*. In *XXXV Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (SBRC)*.

Pereira, Jr., L. A., Mamani, E. L. C., Santana, M. J., Santana, R. H. C., Monaco, F. J., and Nobile, P. N. (2015b). Extending discrete-event simulation frameworks for non-stationary performance evaluation: Requirements and case study. In *Proceedings of the 2015 Winter Simulation Conference*, WSC '15, Piscataway, NJ, USA. IEEE Press.

Qu, C., Calheiros, R. N., and Buyya, R. (2016). Auto-scaling web applications in clouds: A taxonomy and survey. *arXiv Computing Research Repository (CoRR)*, abs/1609.09224.

Rodrigues, R. A., Pereira, L. A., Santana, R., Santana, M., and Monaco, F. J. (2015). Benchmark para análise comportamental do sistema de memória virtual do linux. In *XIV Workshop em Desempenho de Sistemas Computacionais e de Comunicação (WPerformance)*, Recife, PE. Sociedade Brasileira de Computação.

Souza, F. L. S. (2016). Extensão da geração de carga do Bench4Q para benchmark de desempenho em regime transiente. Master's thesis, PPG-CCMC ICMC USP.

Storm, A. J., Garcia-Arellano, C., Lightstone, S. S., Diao, Y., and Surendra, M. (2006). Adaptive self-tuning memory in DB2. *Proceedings of the 32nd international conference on Very large data bases*, pages 1081–1092.

Veeraraghavan, K., Meza, J., Chou, D., Kim, W., Margulis, S., Michelson, S., Nishtala, R., Obenshain, D., Perelman, D., and Song, Y. J. (2016). Kraken: Leveraging live traffic tests to identify and resolve resource utilization bottlenecks in large scale web services. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pages 635–651, GA. USENIX Association.

Yang, F. and Liu, J. (2012). Simulation-based transfer function modeling for transient analysis of general queueing systems. *European Journal of Operational Research*.

Yuan, D., Joshi, N., Jacobson, D., and Oberai, P. (2013). Scryer: Netflix's predictive auto scaling engine - part 2. `https://goo.gl/6tlHdb`. Accessado: 01/12/2016.

Zhang, W., Wang, S., Wang, W., and Zhong, H. (2011). Bench4q: A qos-oriented e-commerce benchmark. In *Computer Software and Applications Conference (COMPSAC), 2011 IEEE 35th Annual*, pages 38–47.

Zheng, W., Bianchini, R., Janakiraman, G. J., Santos, J. R., and Turner, Y. (2009). Justrunit: Experiment-based management of virtualized data centers. In *Proceedings of the 2009 Conference on USENIX Annual Technical Conference*, USENIX'09. USENIX.