Uma abordagem baseada em resposta em frequência para modelagem e avaliação de desempenho não estacionária em sistemas computacionais

Lourenço Alves Pereira Júnior

SERVIÇO DE PÓS-GRADUAÇÃO DO ICMC-USP

Data de Depósito:

Assinatura: _____

Lourenço Alves Pereira Júnior

Uma abordagem baseada em resposta em frequência para modelagem e avaliação de desempenho não estacionária em sistemas computacionais

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

Área de Concentração: Ciências de Computação e Matemática Computacional

Orientador: Prof. Dr. Marcos José Santana

USP – São Carlos
Dezembro 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)

Júnior, Lourenço Alves Pereira
J95a Uma abordagem baseada em resposta em frequência para modelagem e avaliação de desempenho não estacionária em sistemas computacionais / Lourenço Alves Pereira Júnior; orientador Marcos José Santana. -- São Carlos, 2016.
173 p.

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

1. Avaliação de Desempenho. 2. Caracterização de carga de trabalho. 3. Análise no Domínio da Frequência. 4. Função de Transferência. I. Santana, Marcos José, orient. II. Título.

Lourenço Alves Pereira Júnior

Frequency-domain response analysis of computing systems

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

Concentration Area: Computer Science and Computational Mathematics

Advisor: Prof. Dr. Marcos José Santana

USP – São Carlos
December 2016

Dedico este trabalho a minha família.

AGRADECIMENTOS

Ao Professor Franciso José Monaco pela confiança em mim depositada para o desenvolvimento deste trabalho e por sua orientação que me permitiu realizá-lo.

Ao Professor Marcos José Santana pela confiança em mim depositada e pela paciência e ajuda em sua orientação.

Aos amigos e colegas de laboratório pelos bons momentos vivenciados, pela colaboração e apoio que sempre tive.

À minha esposa, às minhas filhas, à minha mãe e à minha irmã pelo suporte e ajuda que sempre me deram.

Ao IFSP pelo período de afastamento a mim concedido, e à CAPES, ao CNPq, ao IFSP e ao NApSoL pelo apoio financeiro concedido. Ao LaSDPC pela infraestrutura que permitiu a realização dos experimentos deste trabalho.

“A ausência da evidência não significa evidência da ausência.”

-Carl Sagan

RESUMO

PEREIRA JR., L. A.. **Uma abordagem baseada em resposta em frequência para modelagem e avaliação de desempenho não estacionária em sistemas computacionais**. 2016. 173 f. Tese (Doutorado em Ciências – Ciências de Computação e Matemática Computacional) – Instituto de Ciências Matemáticas e de Computação (ICMC/USP), São Carlos – SP.

Este trabalho detalha a motivação para a avaliação de desempenho não estacionária de sistemas computacionais e introduz uma abordagem para a modelagem dinâmica de desempenho baseada na análise de resposta em frequência. A modelagem dinâmica é uma abordagem essencial que se desenvolveu como recurso importante na engenharia e algumas ciências naturais por muitas décadas e conta com uma coleção de ferramentas matemáticas para descrever o comportamento dinâmico de sistemas. Seja por tradição ou pela dificuldade de aplicação o uso de modelos dinâmicos em avaliação de desempenho de sistemas computacionais é recente e consideravelmente menos explorada que em outros domínios. A contribuição proposta por esta pesquisa, é a formulação de um arcabouço para avaliação de desempenho não estacionário de sistemas computacionais. O propósito desse arcabouço é produzir um modelo analítico dinâmico experimentalmente construído, que represente a dinâmica que governa o desempenho do sistema. A abordagem é composta por: modelo conceitual para formulação de métricas de desempenho transientes; um método empírico para a obtenção do modelo dinâmico; uma metodologia de análise baseada em resposta de frequência. Usos práticos são ilustrados através de estudos de caso em que os resultados demonstram a aplicabilidade da abordagem.

Palavras-chave: Avaliação de Desempenho, Caracterização de Carga de Trabalho, Análise no Domínio da Frequência.

ABSTRACT

PEREIRA JR., L. A.. **Uma abordagem baseada em resposta em frequência para modelagem e avaliação de desempenho não estacionária em sistemas computacionais**. 2016. 173 f. Tese (Doutorado em Ciências – Ciências de Computação e Matemática Computacional) – Instituto de Ciências Matemáticas e de Computação (ICMC/USP), São Carlos – SP.

This research work details the motivation for non-stationary performance evaluation in computer systems and introduces an approach for dynamic performance modeling based on frequency-response analysis. Dynamic modeling is an essential approach which developed as an important resource in engineering and natural sciences for many decades and counts on a collection of mathematical tools for describing the dynamic behavior of systems. Whether for tradition or difficulty in applying its methods, the use of dynamic modeling in computer systems performance evaluation is recent and less exploited than in other domains. The contribution proposed by the present research is the formulation of a non-stationary performance evaluation framework. The purpose of this framework is to produce an empiric analytical model which represents the dynamics governing the system performance. The approach comprises: a conceptual model for the formulation of dynamic performance metrics; a method to obtain the dynamic model experimentally; a practical methodology based on frequency-domain analysis. Piratical uses are illustrated by study cases in which results show the applicability of the approach.

Key-words: Performance Evaluation, Workload characterization, Frequency-domain analysis.

LISTA DE ILUSTRAÇÕES

LISTA DE ALGORITMOS

LISTA DE CÓDIGOS-FONTE

LISTA DE TABELAS

SUMÁRIO

CAPÍTULO

1

INTRODUÇÃO

Em sistemas computacionais, a avaliação de desempenho é comumente empregada para responder perguntas tais como quão bom é o serviço entregue sob dada carga de trabalho ou qual o limite da capacidade do sistema em operar com níveis mínimos de qualidade. Uma estimativa para tal resposta pode ser obtida experimentalmente impondo-se ao sistema uma entrada estável e medindo-se a variável de desempenho de interesse, seja esta vazão, tempo de resposta, taxa de faltas ou outro parâmetro. Um modo simples de realizar isso é efetuar a medida da variável de desempenho sobre o sistema propriamente dito ou sobre um modelo de simulação.

Embora prática, a obtenção dos resultados através dessa forma é custosa em termos de tempo, dado que o sistema deve ser estimulado com uma nova entrada cada vez que a saída correspondente deve ser predita. Usualmente, experimentos devem ser replicados certo número de vezes para obtenção de confiança estatística.

Com exceção de realizar a medida diretamente sobre o sistema, para se obter uma resposta imediata do tipo "e se", pode ser útil ter em mãos um modelo analítico f da capacidade do sistema que corresponda à noção representada na Figura 1,

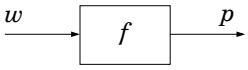


Figura 1 – Modelo conceitual de capacidade.

a qual expressa uma relação entre desempenho p (*performance*) entregue e carga de trabalho w (*workload*) imposta ao sistema. Em engenharia de desempenho de sistemas computacionais, abordagens largamente utilizadas para tal modelo incluem Teoria de Filas e outras ferramentas estatísticas baseadas em modelos Markovianos. Alguns desses métodos são revistos no Capítulo 2. Devido à complexidade dos mecanismos internos e processos concorrentes, é frequentemente difícil aplicar um método tipo caixa-branca (dedutivo) para construir modelos analíticos de sistemas de grande escala. Quando esse é o caso, uma alternativa pode ser um

modelo obtido empiricamente derivado de medidas de entrada-saída. O procedimento conhecido como caixa-preta (indutivo) consiste em realizar medidas sobre o sistema e ajustar um modelo analítico que corresponda à relação observada. Uma variação desse método consiste em realizar as medidas sobre um modelo simulado e recebe o nome de metamodelagem. De qualquer modo, obtém-se um modelo empírico que não necessariamente contém parâmetros associáveis à teoria, e cujos resultados podem não extrapolar além da faixa experimental. Modelos de caixa-preta são, porém, úteis para análise em muitas situações práticas.

Para obtenção do modelo empírico é suficiente expor o sistema a uma entrada e registrar a saída produzida. Nessa análise, usualmente descarta-se o intervalo em que o sistema está em inicialização e limita-se a observação ao período depois que a saída tenha se estabilizado, com sistema na condição operacional especificada. Isso porque, anteriormente, *buffers* são preenchidos e recursos inicializados, de modo que estão presentes condições transitórias. Depois da inicialização, a saída do sistema chega ao regime permanente, e as medidas de desempenho são registradas. O período de inicialização a ser descartado é usualmente denominado *warm-up* (aquecimento).

Geralmente, técnicas de caixa-preta aplicadas à avaliação de desempenho de sistemas computacionais têm como objetivo determinar a capacidade do sistema proporcionalmente a uma carga constante, resultando em um modelo de desempenho estático. Tal modelo é útil para predizer a quantidade de recursos necessária para suportar a demanda operacional sob diferentes condições de carga. Frequentemente é importante verificar como o desempenho se altera temporalmente em resposta a perturbações na carga de trabalho ou nos recursos. Deste ponto de vista, o modelo analítico ilustrado na Figura 1 pode ser detalhado como na Figura 2.

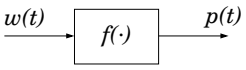


Figura 2 – Modelo de capacidade estacionária.

A especialização se faz útil porque essa abordagem é tão predominante em avaliação de desempenho de sistemas computacionais que a estimação de capacidade *estacionária* é frequentemente assumida sem a distinção explícita da complementar estimação de capacidade *não-estacionária*, isto é, avaliação de desempenho durante o regime transiente. A razão para esse fato é devida à pouca importância da dinâmica de sistemas computacionais tradicionais em muitos problemas de engenharia. Sob essa perspectiva, ordinariamente, plataformas computacionais respondem suficientemente rápido a mudanças operacionais, de modo que o desempenho é afetado por perturbações na carga em uma escala de tempo de menor ordem de magnitude que a da resolução de tempo do processo em questão. Assim é viável assumir que, para efeito prático, a carga de trabalho afeta o desempenho instantaneamente. Portanto, o desempenho pode ser

descrito como uma função da carga:

$$p(t) = f(w(t)). \tag{1.1}$$

Uma segunda razão da predominância da avaliação de desempenho estacionária é porque mesmo quando a inércia do gerenciamento de recursos e atrasos são apreciáveis, se o sistema é projetado para operar sob condições estritamente estacionárias, os transientes podem ser considerados fora da faixa de operação e, portanto, descartados da análise. Sistemas de processamento em lote entram nessa categoria.

Essas hipóteses não necessariamente valem para sistemas de modo geral. Especialmente em ambientes computacionais de larga escala e aplicações multicamadas, o efeito combinando dos pequenos atrasos e sua propagação pelos subsistemas interconectados podem resultar em comportamento dinâmico significativo, observável como efeitos inerciais perceptíveis na resolução de tempo da análise. Esse é o caso de sistemas *soft real-time* que devem operar sob carga altamente variável. Quando essa circunstância se apresenta, os impactos das variações da carga de trabalho no desempenho podem ser expressivos, de modo que transientes podem ser perceptíveis. Isso significa que uma perturbação brusca no número de requisições por segundo que chegam no sistema computacional pode fazer com que o tempo de resposta mude gradualmente de um nível estacionário inicial até chegar ao nível estacionário final. Durante o intervalo transitório, o desempenho pode exibir diferentes comportamentos dependendo da dinâmica do sistema: pode variar abruptamente, ou lenta e continuamente, ou mesmo apresentar oscilações. Essa propriedade do comportamento do sistema é denominada *dinâmica*. Formalmente, um sistema dinâmico é aquele em que a saída em qualquer instante de tempo depende não apenas da entrada imediata, mas também de seu estado interno. Fisicamente, esse estado interno representa a "energia" armazenada no sistema como resultado das entradas anteriores. Por exemplo, a velocidade de um veículo em movimento é função não apenas da potência entregue ao motor presentemente (entrada), mas também da velocidade atual (estado interno, armazenado como energia cinética) de modo que para saber a velocidade do veículo, ambos, a entrada e a condição inicial são necessárias.

Com respeito ao objetivo da modelagem de desempenho estacionário, anteriormente representada na Figura 2, na análise não-estacionária procura-se por um modelo dinâmico conforme referido na Figura 3.

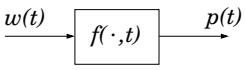


Figura 3 – Modelo de capacidade não-estacionária.

Aqui a variável independente t em $f(\cdot,t)$ significa que $f(w(t),t)$ pode mudar com o tempo, mesmo que a entrada $w(t)$ seja constante, isto é, o desempenho pode evoluir temporalmente mesmo sob carga estacionária como resultado da dinâmica do sistema.

A natureza dos estados internos que determinam a dinâmica pode ser compreendida por uma analogia com o processo físico de equilíbrio térmico. Nesse exemplo, um corpo aquecido a uma temperatura inicial $T(t) = T_0$ perde calor para o ambiente que encontra-se a uma temperatura constante $T_a < T_0$. Mesmo sem recorrer à termodinâmica, é simples perceber que a transferência de calor do objeto mais quente para o meio mais frio ocorre em virtude da diferença de temperatura; conforme o corpo esfria, o potencial para expulsar o calor decresce fazendo com que "a velocidade de resfriamento", isto é, a taxa de mudança de temperatura, decresça também. Isso descreve uma relação diferencial em que a temperatura instantânea do corpo é a solução de uma equação diferencial

$$\frac{dT(t)}{dt} \propto T_a - T_0. \tag{1.2}$$

Explorando, ainda, a analogia, seja o exemplo esquematizado na Figura 4.

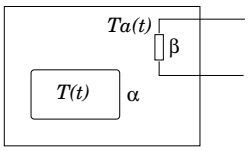


Figura 4 – Analogia com equilíbrio térmico.

O diagrama mostra um objeto sendo aquecido por uma resistência elétrica. Neste exemplo, o sistema em estudo é o objeto e a entrada é a temperatura $T_a(t)$ produzida pelo aquecedor resistivo. A saída é a temperatura $T(t)$ do objeto. A meta é determinar o modelo dinâmico do sistema $f(\cdot,t)$ tal que $T(t) = f(T_a(t),t)$.

Baseado no raciocínio anterior, pode-se escrever a equação diferencial

$$\frac{dT(t)}{dt} = \alpha[\beta T_a(t) - T(t)], \tag{1.3}$$

em que α é o coeficiente de transferência de calor, uma propriedade física do material que mede quão rápido o calor é transferido devido ao potencial de temperatura (por exemplo, um valor alto para metais, baixo para isolantes térmicos) — inclui as propriedades do gás dentro da câmara de aquecimento e o material do objeto. O termo β é a eficiência térmica do aquecedor, uma propriedade física do dispositivo que mede qual a temperatura presente na câmara devido à temperatura produzida pela resistência (considerando-se que um pouco da temperatura se perde por diversos efeitos físicos, de modo que a temperatura da resistência é maior que a temperatura média do interior da câmera, considerada homogênea). No mundo real, α e β podem variar com o tempo e com a temperatura tal que se pode escrever

$$\frac{dT(t)}{dt} = \alpha(T(t),t)\left[\beta(T(t),t)T_a(t) - T(t)\right], \tag{1.4}$$

que é uma equação no geral não-linear, para qual não há solução genérica — e de fato pode não ser solúvel em muitos casos. Em vez disso, se considerar (plausivelmente) que a dependência térmica dos parâmetros é desprezível na faixa de operação, pode-se escrever

$$\frac{dT(t)}{dt} = \alpha(t)\big[\beta(t)T_a(t) - T(t)\big], \tag{1.5}$$

cuja solução, calculável analiticamente por métodos padrão de resolução de equações diferenciais ordinárias, é

$$T(t) = Ce^{-\int \alpha(t)dt} + e^{-\int \alpha(t)dt} \int e^{\int \alpha(t)dt} \beta(t)T_a(t)dt. \tag{1.6}$$

A constante C é determinada a partir das condição inicial $T(0) = T_0$.

Simplificando-se ainda mais, se os coeficientes são invariantes no tempo tal que $\alpha(t) = K_1$ e $\beta(t) = K_2$, a solução da equação 1.6 torna-se

$$T(t) = Ce^{-K_1 t} + e^{-K_1 t} \int e^{K_1 t} K_2 T_a(t)dt \tag{1.7}$$

De volta à analogia, essa é uma função no tempo que diz como a temperatura do objeto (desempenho) desenvolve-se temporalmente conforme a temperatura do aquecedor (carga do trabalho) muda dinamicamente. Por exemplo, se o objeto está a uma temperatura inicial $T(0) = T_0$ quando o aquecedor é desligado $T_a(t) = 0$ para $t \geq 0$ (supressão da carga), a equação 1.7 se reduz à forma da equação 1.8, a qual diz que a temperatura do objeto (desempenho) decai exponencialmente enquanto a temperatura armazenada (carga residente) é consumida à exaustão:

$$T(t) = T_0 e^{-K_1 t} \tag{1.8}$$

Da mesma forma, se depois de um longo período de entrada constante, que corresponda a uma temperatura do objeto $T(0) = T_0$, a temperatura cresce abruptamente (rajada de carga) e mantem-se no valor mais alto $Ta(t) = T_A$, para $t \geq 0$ a equação 1.7 toma a forma 1.9

$$T(t) = T_0 e^{-K_1 t} + K_2 T_A (1 - e^{-K_1 t}) \tag{1.9}$$

Novamente pode-se ver que, na ausência de entrada, i.e. $T_a(t) = 0$, a saída decai exponencialmente. Contudo, pode-se também ver que, no cenário descrito, a saída aumenta assintoticamente até o valor final de estabilização T_A.

O propósito dessa analogia é comparar o escopo da modelagem estacionária e não estacionária, bem como suas diferentes metodologias e requisitos. O exemplo descrito foi desenvolvido na abordagem de caixa-branca, utilizando princípios elementares da Física. Para

aplicar uma abordagem de caixa-preta, o método de avaliação de desempenho estacionário corresponderia a medir os valores de $T_a(t)$ e $T(t)$ para obter relação entre as variáveis.

A comparação apenas entre os valores finais das duas variáveis para diversos valores de $T_a(t \geq 0) = T_A$, revelaria um modelo estático como o da Figura 2, ao passo que a comparação das sequências temporais completas da evolução das duas váriáveis resultaria em um modelo como o da Figura 3.

Note-se que o valor resultante obtido pela análise estacionária é precisamente o calculado pelo modelo dinâmico quando os transitórios terminam, fazendo-se t tender ao infinito:

$$\begin{aligned} T &= cT_A \\ c &= \lim_{t \to \infty} \frac{T(t)}{T_a(t)} = K_2 \\ \therefore T &= K_2 T_A \end{aligned} \tag{1.10}$$

Vê-se assim que o modelo estacionário é uma forma mais restrita do modelo dinâmico, que revela apenas os valores após a estabilização. Se o sistema está constantemente em regime transiente devido a carga não estacionária, a previsão de desempenho obtida pelo modelo estacionário não corresponde à saída real. Por exemplo, a contabilização dos recursos utilizados durante a operação pode divergir nos dois cálculos.

Uma contribuição central pretendida pela linha de investigação científica desenvolvida no grupo de pesquisa, em cujo escopo o presente trabalho é inserido, é lançar luz acerca das condições em que a hipótese de dinâmica desprezível de sistemas computacionais não é adequada, bem como as circunstâncias em que ser capaz de predizer o comportamento transiente é útil na avaliação de desempenho. Esse é o caso, por exemplo, de sistemas distribuídos de larga escala e aplicações complexas multicamadas em que o efeito combinado dos atrasos no caminho entrada-saída dos diversos componentes que compõem o sistema pode resultar em dinâmica significante.

Propriedades dinâmicas de desempenho de sistemas computacionais podem resultar do hardware ou do software. No nível físico, inércia resulta de tempos de respostas de operação de I/O, canais de comunicação e eletrônica dos periféricos; no nível lógico, a dinâmica pode resultar do preenchimento de buffers, contenção em acesso concorrente a recursos compartilhados, inicialização de recursos e latência de comunicação entre componentes da arquitetura. Também ações de auto adaptação baseadas em valores agregados, tais como tempo médio de resposta, utilização de CPU etc., introduzem dinâmica, uma vez que médias são, matematicamente, "fontes de inércia".

Esses efeitos ocorrendo através do sistema podem ter como resultado que o caminho entrada-saída seja um processo não instantâneo e não desprezível na escala de tempo de interesse. Isso significa que uma mudança súbita na demanda — por exemplo um rápido aumento no

número de requisições ou a falha no recurso — pode causar um efeito retardado no desempenho tal que a consequência da perturbação na condição operacional não seja visível imediatamente. Em vez disso, pode se manifestar como um simples atraso ou, dependendo da dinâmica, evoluir temporalmente alterando o desempenho gradualmente. Uma dinâmica de primeira ordem (Capítulo 3), por exemplo, pode fazer o desempenho variar suavemente e monotonicamente até atingir o estacionário final. Uma dinâmica de ordem mais alta, por sua vez, pode fazer o desempenho exibir oscilações amortecidas até a estabilização.

Contrastando com a identificação de sistema estacionário, enquanto um modelo de capacidade desse tipo prediz o desempenho em função da carga imediatamente imposta, o modelo analítico de capacidade não estacionária prediz o desempenho como função da carga e do tempo. Essa súbita distinção é essencialmente importante para a discussão que se segue. Ressalta que o escopo da avaliação de desempenho estacionária é de informar como o sistema lida com a *carga*, enquanto a avaliação de desempenho não estacionária estende essa cobertura para informar como o sistema lida com a *variação de carga*.

Conhecer o modelo de desempenho dinâmico do sistema computacional pode revelar, por exemplo, que a potência computacional suficiente para atender a uma demanda moderada constante e garantir dada qualidade de serviço, pode ser insuficiente mediante a ocorrência de variações de carga transitórias. Uma analogia é o motor de um veículo que pode necessitar mais potência para acelerar o veículo a uma velocidade moderada a partir do repouso, do que para mantê-lo em alta velocidade constante.

A análise transiente pode também revelar que, dependendo das propriedades dinâmicas, um sistema com recursos suficientes para atender a uma carga estacionária pode ser sobrecarregado por uma variação abrupta de carga ainda que de pequena amplitude. Uma comparação simples esclarece o argumento. A Figura 5 ilustra um sistema mecânico onde uma mola mantém suspensa uma massa m presa a uma de suas extremidades. Como sabido, a força exercida por uma mola é proporcional e no sentido contrário à distensão. Logo, sob a ação da gravidade, na condição de equilíbrio, a mola é esticada da sua posição relaxada até determinado alongamento. Nesse ponto, a força exercida pela mola se anula perfeitamente com a força peso de forma que a extremidade livre permanece imóvel. Considere-se esse ponto a referência $x = 0$ para a posição vertical da massa.

Nessa analogia o sistema de interesse é a mola, e se está interessado na sua capacidade de suportar uma determinada massa m, sem que essa chegue ao nível do solo. O exemplo sugere o paralelo com um sistema computacional para o qual se deseja determinar a máxima carga de trabalho admissível (massa) sem que determinado recurso (mola) se torne insuficiente para atender o SLA[1] — por exemplo, provisões de buffers, número de processadores etc. — pela degradação excessiva de desempenho (distenção) — por exemplo, tempo de resposta, taxa de utilização, consumo energético etc.

[1] Service-level agreement: acordo de nível de serviço.

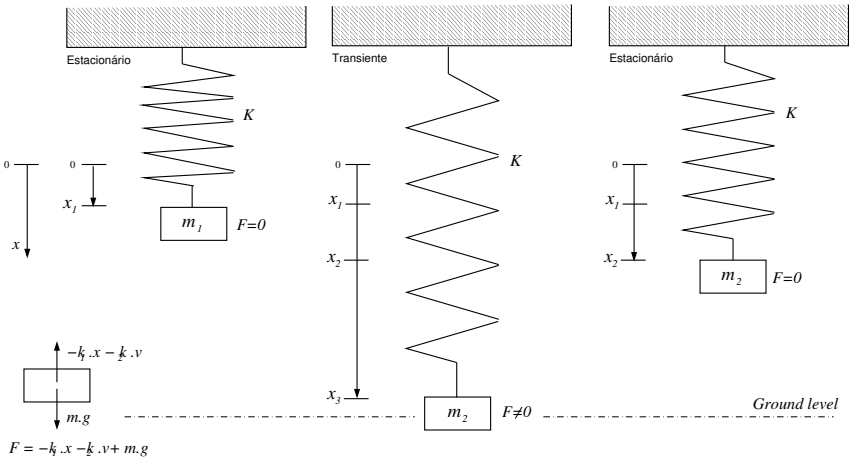


Figura 5 – Analogia massa-mola.

A avaliação de desempenho no regime estacionário permite constatar que, com uma massa m_1 a distensão máxima da mola é x_1 (à esquerda), e que $m_2 > m_1$ é ainda uma carga suportável pelo sistema, posto que a distensão $x_2 > x_1$ não viola o SLA (à direita). Não é difícil aumentar incrementalmente a massa até encontrar um valor m_3 que distenda a mola até o nível do solo, representando, assim, um limite para a capacidade do sistema. A partir daí, é possível projetar uma política de admissão para rejeitar uma carga excessiva (massa maior que m_3) ou replanejar a capacidade do servidor (substituindo a mola).

Esse é o resultado da análise estacionária. Observe-se, por outro lado, o que ocorre se, a partir da condição inicial onde a mola suporta a massa m_1, a carga subir bruscamente para o valor m_2. Ambos os valores m_1 e m_2 são suportados pelo sistema sem violar o SLA quando se considera o desempenho final, findos os transitórios. Porém durante o período de estabilização, no regime transiente, é intuitivo prever que a mola será distendida para além de x_2 por ação da brusca perturbação. No caso de uma mola ideal, é esperado que o sistema oscile; no caso real, espera-se que a mola perca gradualmente energia até que finalmente estacione em x_2. Note-se portanto que a provisão de recursos (constante da mola) determinada como suficiente pela avaliação de desempenho estacionária não é válida na condição transiente.

Um sistema computacional pode entrar em oscilação devido a caminhos de realimentação introduzidos por mecanismos adaptativos projetados de modo que alguma medida de desempenho seja usada para regular a demanda — por exemplo, um filtro de admissão ou gerenciador de recursos. Se a dinâmica de sistema é inadvertidamente ignorada, mudanças de desempenho podem ser detectadas tardiamente e uma reação desproporcional pode ser tomada quando a perturbação já tiver terminado, causando uma inversão do fenômeno e assim por diante. Em casos severos, o comportamento oscilatório pode possuir frequências de ressonâncias que podem saturar a capacidade do sistema ou causar propagação das oscilações através do sistema. Em casos extremos uma única perturbação pode levar o sistema a instabilidade[2] catastrófica.

[2] É bem conhecido o caso da ponte do rio Tacoma, que na década de 40 colapsou após entrar em ressonância

Esses exemplos e considerações ilustram a utilidade de conhecer as propriedades dinâmicas de desempenho de sistemas computacionais. Contudo, enquanto a modelagem não estacionária não é um conceito desconhecido e, de fato tem sido uma ferramenta essencial em diversos segmentos da engenharia, é entretanto, relativamente menos explorada no domínio da avaliação de desempenho de sistemas computacionais. A despeito de trabalhos pioneiros e contribuições registradas na literatura (Capítulo 2), de modo geral, a modelagem dinâmica não é uma abordagem principal em avaliação de desempenho de sistemas computacionais. E embora desenvolvida como fundamento da Teoria de Controle e Análise de Sinais, as quais desempenham papel importante em problemas envolvendo engenharia elétrica e mecânica, bem como ramos da química, biologia, epidemologia, em grande parte, a modelagem dinâmica para análise não estacionária é frequentemente uma disciplina exótica para o arcabouço tradicional de sistemas de computação, mais especificamente, para avaliação de desempenho.

Nesse ponto, destaca-se uma das contribuições propostas pelo presente estudo, o de auxiliar a compreensão dos potenciais usos de análise não estacionária na avaliação de desempenho de sistemas computacionais, especificamente, como ferramenta de modelagem e análise. Este intuito é consistente com a hipótese de pesquisa do trabalho de investigação desenvolvido no LaSDPC[3], e com outros trabalhos de pesquisa para os quais o resultado do presente estudo contribuiu durante seu desenvolvimento.

Uma segunda contribuição pretendida pela pesquisa é abordar o fato de que, relativamente à modelagem estática (não estacionária), a modelagem dinâmica (transiente) é consideravelmente mais difícil. Primeiramente porque modelos dinâmicos são inerentemente mais sofisticados do ponto de vista analítico. Matematicamente suas expressões correspondem a equações diferenciais — no tempo contínuo — ou equações de diferença — no tempo discreto — nem todas das quais tem solução analítica. Seu tratamento envolve aproximações, especialmente baseadas em hipóteses de linearidade e invariância no tempo. O ferramental padrão para a solução de equações diferenciais inclui transformação de domínio por meio de operador de Laplace, ou seu equivalente discreto, a transformada Z.

Em segundo lugar, também porque, a despeito de técnicas existentes para abordar sistemas físicos, naturais ou artificiais, o seu uso em sistemas computacionais, em virtude da relativa recentidade, não possui estabelecidas especializações dos métodos gerais para as particularidades dos sistemas computacionais. De fato, sistemas computacionais possuem características que tornam sua modelagem dinâmica particularmente difícil. Uma das razões é que se tratam de sistemas notoriamente não lineares: a complexidade da interação de muitos subssistemas e natureza lógica de seus algoritmos não correspondem a comportamento linear. Desempenho de sistemas computacionais satura-se e frequentemente apresenta características assimétricas com respeito a variações positivas e negativas (por exemplo, tempo de alocação e dealocação de recursos).

mecânica com ventos relativamente suaves, mas com frequência coincidente com a frequência natural da estrutura.

[3] Laboratório de Sistemas Distribuídos e Programação Concorrente

Também apresentam parâmetros variantes do tempo (por exemplo velocidade de I/O influenciada por cache). Ainda seus processos são de natureza estocástica, o que acrescenta complexidade adicional. Essas dificuldades contribuem para a pouca utilização das técnicas aqui descritas em avaliação de desempenho. Nesse sentido, uma adição deste trabalho é o desenvolvimento e formulação de uma abordagem prática para avaliação de desempenho não estacionária de sistemas computacionais.

Enquanto alguns trabalhos na literatura da área (Capítulo 2) tem recentemente reportado o uso de modelagem dinâmica em alguns problemas de computação, na sua maior parte o uso dessas técnicas é explorada como ferramenta matemática auxiliar para solução de problemas específicos, geralmente, aplicações de mecanismos realimentados (*feedback*) para alocação de recursos. Frequentemente tais usos são mencionados como artifícios *ad hoc*, em lugar de metodologias generalizadas, e usualmente requerendo expertise em Teoria de Controle e disciplinas relacionadas além do escopo do *background* usual da avaliação de desempenho.

Este trabalho, diferentemente, introduz modelagem dinâmica como metodologia para avaliação de sistemas computacionais. Para esse fim, a pesquisa necessitou enfrentar diversos desafios específicos da modelagem empírica de sistemas computacionais e desenvolver metodologias e ferramentas de instrumentação para seu uso prático.

Uma característica importante da abordagem é a modelagem e análise no domínio da frequência aplicada à avaliação de desempenho. Basicamente a modelagem no domínio da frequência significa substituir a representação analítica da entrada e da saída como funções no tempo, pelos seus equivalentes em um domínio onde a variável independente é a frequência. Intuitivamente essa transformação pode ser entendida como a equivalência entre expressar uma série de medidas ao longo do eixo do tempo e expressar a mesma série como valores ao longo deo eixo das frequências. Por exemplo, se uma série temporal tem a forma

$$u(k) = A.sen(\omega k), \tag{1.11}$$

em que k representa o tempo discreto e ω, uma frequência constante, é possível representar a mesma série como um único ponto em um plano onde uma das coordenadas é a amplitude a outra coordenada é a frequência, ficando o tempo implícito:

$$U(\omega) = (A, \omega). \tag{1.12}$$

Séries mais complexas no tempo podem ser representadas como uma soma de pares amplitude-frequencia incluindo valores complexos que representam componentes defasados. Essa é a noção associada a série de Fourier usada para representar sinais periódicos no domínio da frequência.

Na coleção de trabalhos analizados nesse estudo, é possível observar poucos casos de

uso de métodos de domínio da frequência para tratar de problemas de sistemas computacionais; mesmo os exemplos que fazem uso de tais métodos, abordam-nos como artifício matemático para resolver equações diferencias, especialmente em problemas envolvendo modelos temporais autoregressivos. A utilização padrão de operadores de Laplace e Z é um expediente para solucionar problemas de domínio do tempo que se tornam mais simples de serem tratados no domínio da frequência. O procedimento comum é transpor o problema do domínio do tempo para a frequência, resolvê-lo nesse domínio, e então transpô-lo novamente para o domínio do tempo. Para sistemas lineares invariantes no tempo, esse trabalho de mudança de domínios tende a ser menos trabalhoso e mais simples do que o esforço exigido para resolver o problema diretamente no tempo.

Por exemplo, os sistemas dinâmicos que correspondem às equações diferenciais lineares e invariantes no tempo são convertidos em equações puramente algébricas no domínio da frequência. Conforme o ilustrado, o exemplo anterior da equação 1.6 cujo modelo dinâmico no tempo é

$$T(t) = Ce^{-K_1 t} + e^{-K_1 t} \int e^{K_1 t} K_2 T_a(t) dt \tag{1.13}$$

e requer uma longa sequência de passos para sua solução, torna-se

$$T(s) = \frac{K_1 K_2}{s + K_1} T_a(s) \tag{1.14}$$

no domínio da frequência[4]. A razão entre entrada e saída, que no domínio do tempo é uma relação diferencial, no domínio da frequência torna-se razão de polinômios, denominada função de transferência. Note-se que essa é uma relação puramente algébrica e portanto que, se dois sistemas desse tipo são acoplados em sequência, de modo que a saída de um é a entrada de outro

$$\begin{aligned} \dot{y}(t) + ay(t) &= u(t) \\ \dot{u}(t) + bu(t) &= x(t) \end{aligned} \tag{1.15}$$

então, para saber $y(t)$ como função de $x(t)$, no domínio do tempo é necessário solucionar o sistema de equações diferenciais para se chegar ao resultado não trivial

$$y(t) = C_1 e^{-at} + e^{-at} \int e^{at} \left[C_2 e^{-bt} + e^{-bt} \int e^{bt} x(t) dt \right] dt \tag{1.16}$$

Se outros sistemas são adicionados em série ou paralelo, a rede torna-se crescentemente complexa, especialmente se sistemas de ordem mais alta são adicionados.

[4] A variável s é uma frequência complexa, tal que $e^s = e^{\alpha + i\omega} = e^{\alpha}.e^{i\omega}$, e utilizando a fórmula de Euler, resulta em $e^{\alpha}(\cos\omega + i\sin\omega)$. Note-se que a seoma dos conjugados $e^{\alpha+i\omega} + e^{\alpha-i\omega}$ é uma senoide real defasada e amortecida ($\alpha < 0$) no tempo, representando a dinâmica de uma mola real abordada no exemplo.

Por outro lado, no domínio da frequência a expressão torna-se apenas a multiplicação das funções de transferência dos sistemas,

$$Y(s) = \frac{a}{s+a}\frac{b}{s+b}X(s) \tag{1.17}$$

pois a operação $f \circ g$ no domínio do tempo torna-se a multiplicação algébrica FG na frequência. Arranjos em série e paralelo podem ser solucionados também algebricamente.

Com respeito ao uso da transformação para domínios da frequência como artifício para a solução de equações diferenciais no domínio do tempo, uma contribuição da presente pesquisa é propor uma abordagem para a análise de desempenho dinâmica de sistemas computacionais diretamente no domínio da frequência, e utilizar essa visão do sistema para fazer previsões acerca do comportamento estacionário e transiente do sistema. A abordagem desenvolvida pontua conclusões úteis que podem ser obtidas diretamente da perspectiva do domínio da frequência, as quais são consideravelmente mais difíceis, senão impraticáveis, de serem apreciadas no domínio do tempo. A abordagem também oferece um arcabouço conceitual para a avaliação de desempenho não estacionário que formula a noção de desempenho dinâmico, capacidade dinâmica e caracterização de dinâmica de carga, como extensões de suas equivalentes estáticas. A proposta objetiva tornar essas ferramentas metodológicas acessíveis a projetistas de experimentos de avaliação de desempenho.

1.1 Objetivo

Formular um arcabouço de avaliação de desempenho não estacionário de sistemas computacionais baseado em resposta em frequência, que auxilie apreciar as propriedades dinâmicas do sistema e fazer previsões acerca de transientes no desempenho em presença de perturbações do regime de operação. O arcabouço inclui um modelo conceitual para apresentação da dinâmica de desempenho; um método prático para obtenção experimental do modelo de desempenho; a eliciação de requisitos para o projeto e implementação de instrumentação experimental; e um método de análise para realizar previsões a partir do modelo analítico. Em adição, a aplicabilidade do arcabouço deve ser empiricamente demonstrada por meio de exemplos e estudos de caso.

1.2 Contribuições

O foco elaborado nessa seção introdutória, contribuição central do trabalho, é a formulação de uma abordagem baseada em resposta em frequência para modelagem e avaliação de desempenho não estacionária de sistemas computacionais que atendam às propostas delineadas nos objetivos. Enquanto existe uma extensa teoria da análise em frequência aplicada a outras áreas, durante o desenvolvimento do presente trabalho, o estudo enfrentou desafios que emergem da especialização de resultados gerais para particularidades de sistemas computacionais,

especificamente para avaliação de desempenho. Dentre esses, destacam-se a não linearidade, parâmetros variantes do tempo, indeterminismo, agravados pela complexidade de sistemas em larga escala. Enquanto alguns subsistemas menores podem ser analisados sob uma série de hipóteses simplificadoras — por exemplo, Teoria de Fila e suas extensões — não é imediatamente claro se essas aproximações são úteis ou viáveis na prática para sistemas do mundo real. Uma outra contribuição da pesquisa, portanto, é validar o uso de aproximações experimentalmente por meio de uma série de estudos de caso — alguns projetados especificamente para testar as hipóteses desse estudo, alguns outros parte de pesquisas com as quais o trabalho contribuiu — os resultados demonstram dados práticos em que a dinâmica de sistemas computacionais complexos e de larga escala podem ser razoavelmente aproximados por modelos simples lineares e invariantes no tempo de baixa ordem, e que são capazes de produzir predições úteis.

Uma contribuição adicional, ainda, é a formulação de uma arquitetura de separação de responsabilidades para o projeto de instrumentação para obtenção de modelos de desempenho dinâmico. Essa arquitetura elicia requisitos para o desenvolvimento de ferramentas para obtenção experimental dos dados necessários para ajuste do modelo dinâmico. Essa arquitetura conceitual foi utilizada para orientar o projeto de implementação e duas ferramentas, uma para simulação, implementada pelo autor e outra para *benchmarking*, implementada como parte do trabalho de pesquisa relacionado (MAMANI, 2016). O *framework* de simulação não estacionário é uma extensão para o sistema CloudSim (CALHEIROS *et al.*, 2011), que adiciona uma série de capacidades úteis para análise transiente. O sistema é descrito em um estudo de caso em que é utilizado para obtenção de um metamodelo analítico de um provedor IaaS (*Infraestructure as a service.*). A ferramenta *benchmarking* é uma extensão do sistema Bench4Q (ZHANG *et al.*, 2011), um *framework* para avaliação estacionária de aplicações SaaS (*Software as a service.*). A ferramenta é utilizada para obter um modelo de desempenho dinâmico de uma infraestrutura física de computação em nuvem, e é também coberta nos estudos de caso. Ambos os sistemas são disponibilizados sob licença *open source*. Para a análise de dados alguns scripts implementados para software de simulação numérica compatíveis com GNU Octave / Matlab foram também produzidos e disponibilizados da mesma forma. Resultados da pesquisa têm sido reportados em veículos científicos e incluem, até o momento da elaboração deste documento:

- PEREIRA, L. A.; MAMANI, E. L. C.; SANTANA, M. J.; SANTANA, R. H. C.; NOBILE,P. N.; MONACO, F. J. *Non-stationary simulation of computer systems and dynamic performance evaluation: A concern-based approach and case study on cloud computing*. In:ComputerArchitecture and High Performance Computing (SBAC-PAD), 2015 27th InternationalSymposium on. [S.l.: s.n.], 2015. p. 130–137. ISSN 1550-6533.

- PEREIRA JR., L. A.; MAMANI, E. L. C.; SANTANA, M. J.; SANTANA, R. H. C.; MONACO, F. J.; NOBILE, P. N. *Extending discrete-event simulation frameworks for non-stationaryperformance evaluation: Requirements and case study*. In:Proceedings of

the 2015 WinterSimulation Conference. Piscataway, NJ, USA: IEEE Press, 2015. (WSC '15), p. 3150–3151.ISBN 978-1-4673-9741-4.

- MAMANI, E. L. C.; PEREIRA, L. A.; SANTANA, M. J.; SANTANA, R. H. C.; NOBILE,P. N.; MONACO, F. J. *Transient performance evaluation of cloud computing applications anddynamic resource control in large-scale distributed systems*. In:High Performance ComputingSimulation (HPCS), 2015 International Conference on. [S.l.: s.n.], 2015. p. 246–253

- PEREIRA, L. A.; SANTANA, M. J.; SANTANA, R. H. C.; MONACO, F. J. *Dynamic modeling of computer systems performance*. Submetido para o Elsevier Performance Evalation.

1.3 Metodologia

Baseado na motivação apresentada, o estudo prossegue com a revisão do estado da arte em análise transiente em problemas dos sistemas computacionais. O resultado dessa revisão é sumarizado no Capítulo 2.

A abordagem proposta é então introduzida no Capítulo 3, iniciando com uma fundamentação teórica necessária para conduzir o estudo e tecendo o arrazoado para a abordagem proposta. O método é então detalhado nos próximos capítulos. Os principais elementos da abordagem proposta são exemplificados por meio de estudos de caso oferecidos tanto como demonstração de utilização prática, como de validação da aplicabilidade da abordagem em cenários realísticos. O método de identificação de sistema para obtenção do modelo dinâmico de desempenho é um primeiro passo comum em qualquer utilização do *framework* desenvolvido, e é exemplificado em todos os estudos de caso, tanto por meio de simulação quanto por meio de *benchmarking* sobre um protótipo físico implementado no LaSDPC. A instrumentação desenvolvida no estudo é descrita no Capítulo 4. Um dos estudos de caso, descrito no Capítulo 5, elabora sobre métricas de desempenho dinâmico, estendendo-se sobre ambos, análise (estimação do modelo dinâmico de desempenho) e síntese (projeto de sistema para atender requisitos de desempenho dinâmico). Os Capítulos 6 e 7 focam na análise de resposta em frequência e predição de transientes a partir da análise espectral. Finalmente, o Capítulo 8 discute as principais observações e conclusões da pesquisa destacando as capacidades do método em permitir predições úteis, suas limitações e anotações finais sobre contribuições e trabalhos futuros.

CAPÍTULO

2

REVISÃO BIBLIOGRÁFICA

2.1 Análise transiente de Redes de Filas

A modelagem por Teoria de Filas foi utilizada com sucesso por Kleinrock (1975) com a finalidade de representar sistemas de rede de computadores. Houve uma adoção em massa pela comunidade computacional, sendo utilizada como ferramenta analítica padrão para modelagem e avaliação de desempenho. No entanto, a base matemática para formulação do modelo é estacionária. Portanto, se sistemas apresentarem comportamento transiente apreciável, é necessário que equacionamentos e abordagens extras sejam feitas a fim de que o modelo ainda seja útil. Nesse sentido, pesquisas que visam encontrar soluções para o comportamento dinâmico em Redes de Filas é tema de estudo na comunidade científica. Feng Yang e Jingang Liu enumeraram as principais abordagens para a solução desse problema (LIU; YANG, 2008; YANG; LIU, 2010; YANG; LIU, 2012). Segundo Yang e Liu (2012), é possível atacar o problema de duas formas: métodos analíticos ou simulação. Os métodos analíticos são divididos em duas categorias de métodos, cujos os resultados são *exatos* ou *aproximados*.

Os métodos **exatos** restringem-se a Redes de Filas Markovianas, das quais a solução para encontrar o transiente é a resolução de Equações Diferenciais Ordinárias (EDO) com parâmetros temporalmente dependentes. O sistema neste caso é um sistema probabilístico baseado em estados. As soluções para este caso são raras. Exceções a este caso incluem os modelos M(t)/G/∞ e M/M/1 (KLEINROCK, 1975; GROSS; HARRIS, 1985) e Ph(t)/Ph(t)/∞ (NELSON; TAAFFE, 2004a; NELSON; TAAFFE, 2004b). Ingolfsson *et al.* (2007) apresenta um estudo sobre soluções numéricas para essas EDOs.

Rothkopf e Oren (1979) descrevem métodos computacionais para calcular a saída de sistemas de fila M/M/s cujos tempos de serviço e chegada variam no tempo. A abordagem é uma generalização para a solução do caso do modelo M/M/1. São utilizados como testes sistemas com 1 e 3 servidores (variável s) com tempos de serviços constantes e variação

senoidal da taxa de chegada. O estudo é teórico e específico para esse modelo de fila. O método apresenta restrições na forma com que o estacionário é encontrado e é limitado pela aplicabilidade e/ou generalização para outras distribuições. Clark (1981) apresenta uma solução para essa limitação ao utilizar a distribuição Eggenberger-Polya (JOHNSON; KOTZ, 1969), que possui uma variância menor e, por isso, produz resultados melhores para a aproximação. As duas soluções apresentadas servem para simulações contínuas. Segundo Taaffe e Ong (1987), essas abordagens consistem da solução de um grande conjundo de equações diferenciais de primeira ordem a serem resolvidas; ele apresenta uma solução baseada na *surrogate distribution approximation* (SDA) que reduz a quantidade de equações a serem resolvidas, porém o resultado é menos generalizável, especialmente para sistemas com capacidade finita.

Gross e Miller (1984) apresentam uma solução mais generalizável para o caso discreto, mas com tempo contínuo. A solução é baseada na extensão de um método denominado *randomization* (GRASSMANN, 1977). A ideia principal é fazer uma matriz de transições do sistema markoviano. A formulação e a descrição do problema é generalizável para modelos de filas, porém a solução é computacionalmente cara, pois envolve métodos iterativos para a resolução de sistemas lineares.

A fim de quantificar a qualidade da aproximação utilizada para obter o comportamento transiente, Green, Kolesar e Svoronos (1991) conduzem uma análise nesse sentido. Os autores mostraram empiricamente que a falta de precisão pode impactar negativamente na previsão de atrasos (*delays*) e também na representação dos sistema quando em regime estacionário. Ou seja, ainda que com boa precisão e pequenos tempos de regime transiente, há um impacto na acurácia do modelo em relação ao regime estacionário. Um método para solucionar o sistema é apresentado em (GREEN; KOLESAR, 1991), em que é argumentado ganhos nessa acurácia. É importante mencionar que o tempo entre as chegadas é tratado como senoidal. Também nesse sentido, (EICK; MASSEY; WHITT, 1993b; EICK; MASSEY; WHITT, 1993a) apresentam um estudo para o caso de fila $M_t/G/\infty$ com variação periódica no intervalo entre as chegadas.

Uma abordagem que explorou planejamento de capacidade e com intervalo entre as chegadas dependente do tempo foi (JENNINGS *et al.*, 1996). A solução proposta aborda o modelo $M_t/M/s_t$ e provê um meio para especificação dinâmica da quantidade de centros de serviço para atender uma demanda variável e não estacionária. O método proposto é baseado em aproximação e leva consideração que há infinitos centros de serviços disponíveis. No entanto, a solução desse sistema ainda exige a mesma abordagem de resolver equações diferenciais. O caso de uma quantidade finita de servidores foi abordado em (MASSEY; WHITT, 1997), especificamente os autores trataram da formulação de como encontrar os valores de picos (valores máximos) para a quantidade de servidores ocupados e taxa de chegadas. Esses valores de pico acarretam em gargalos. O modelo estudado foi $M_t/G/\infty$ com baixas alterações nas condições operacionais, mas que mesmo assim acarretem em transiente.

É importante mencionar que (1) as soluções apresentadas por esses autores são funda-

mentadas pela característica Markoviana dos modelos e (2) as equações são computacionalmente caras de serem resolvidas dadas as soluções numéricas existentes.

A categoria dos métodos baseados em **aproximações** são baseados em métodos numéricos de dinâmica de fluídos computacionais. As soluções admitem pouca variabilidade e forte aderência a modelos de adoção (BASS, 1969), de forma que a aplicabilidade restringe-se a sistemas em condições operacionais de alta carga (CHEN; MANDELBAUM, 1994; MANDELBAUM, 1995).

Há uma série de outros métodos para resolução pontual de problemas da mesma natureza daqueles descritos até o momento, porém com um foco maior em sistemas de manufatura, em que a modelagem do comportamento transiente é importante. Pode-se citar o método de equações que faz a caracterização do transiente baseado na Lei de Little (RIANO, 2002); a descrição do fluxo de requisições reentrante em uma fábrica através de um modelo hiperbólico (ARMBRUSTER *et al.*, 2006); e estratégias para representar a dinâmica do sistema no domínio do tempo considerando sistemas de filas (HACKMAN, 2008).

Em suma, esses métodos possuem uma relação inversa entre acurácia e tempo de execução. Ou seja, métodos com melhores resultados são mais lentos de serem executados, ao passo que métodos com melhores tempos de execução produzem resultados não tão bons. De qualquer forma, eles apresentam restrições de utilização que podem inviabilizar situações práticas. Por exemplo, o caso de um sistema em que a demanda diminui em situações de alta utilização não seria factível nesses modelos, pois o processo de chegada não é um processo de chegada Markoviano. Exemplos de outras variáveis que influenciam nesses resultados: tempo de serviço, processo de falha de centro de serviços e requisições reentrantes.

A solução desses tipos de modelos por **simulação** computacional é uma alternativa, pois é possível gerar o comportamento transiente de um sistema. O problema desse tipo de abordagem é que ela exige múltiplas replicações para obter um resultado estatisticamente confiável, fazendo com que ela seja computacionalmente intensiva para aplicações de tempo real. Outro ponto é que execuções do tipo "e-se" (*what-if*) carecem de configurações adicionais. Esses pontos tornam a adoção de simulação como solução para o modelo de um sistema impraticáveis de serem adotadas em aplicações reais.

Como exemplo, seja um sistema M/M/1 configurado inicialmente com taxa de serviço $\mu = 1/25\,\text{ms}$ e taxa de chegada $\lambda = 1/31\,\text{ms}$. A resolução deste modelo foi feita através de simulação, utilizando-se o SIMPACK (FISHWICK, 1992). Os resultados são apresentados com intervalo de confiança de 95%. Foi simulado um tempo de operação equivalente 500 segundos, em outras palavras, tempo de simulação igual a 500 segundos. Após o instante de simulação que corresponde a 100 segundos, o taxa de chegada foi alterada para $\lambda = 1/27\,\text{ms}$. Isso significa que houve um aumento brusco na quantidade de requisições que chegaram ao sistema, pois o intervalo entre as chegadas foi menor (de $\Delta\lambda = 31\,\text{ms}$ para $\Delta\lambda = 27\,\text{ms}$). A variável de saída foi o tempo de resposta. Durante a simulação foram coletadas três informações, adicionadas a um

arquivo CSV (*Comma Separeted Value*). Ou seja, cada réplica executada gerou um arquivo com três colunas: (1) tempo de simulação, (2) intervalo entre as chegadas, e (3) tempo de resposta. A amostragem foi periódica com intervalo de 300 ms. A valor coletado corresponde ao valor atual das variáveis.

Os experimentos foram replicados várias vezes para obter uma confiança estatística maior. Neste caso, a intenção é apresentar um comparativo da quantidade de replicações e seu impacto na qualidade e o tempo de execução necessário para obter o resultado. O número total de réplicas constou de duas execuções de diferentes quantidades: 250 e 10.000. As sementes usadas pelo gerador de números pseudo-aleatório foram obtidas através do sítio *random*[1]. Os números foram verificados para evitar duplicidade.

A Fig. 6 apresenta os valores empíricos obtidos através da execução dos experimentos. Após o instante de tempo 100 s há a diminuição no intervalo em que novas requisições chegam ao sistema. A Fig. 7 apresenta a respectiva saída do sistema, tempo de resposta. A forma com que o relacionamento de causa e efeito das variáveis acontece revela que há um período transiente apreciável, pois o novo patamar de estacionariedade da variável de saída não é instantâneo. Convenientemente, a Fig. 8 apresenta as duas variáveis no mesmo gráfico para ficar mais claro os momentos em que há a transição do patamar estacionário inicial para o segundo.

(a) 250 réplicas. (b) 10.000 réplicas.

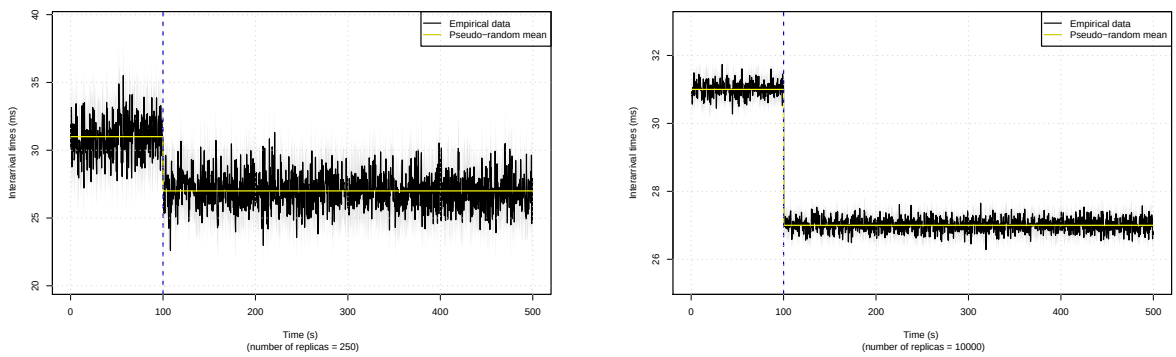


Figura 6 – Entrada do sistema: intervalos entre as chegadas. Os valores foram gerados conforme uma distribuição exponencial com média 31 ms até o instante de tempo 100 s e 27 ms no restante da simulação. A linha vertical no instante de tempo 100 s destaca o momento em que o degrau aconteceu.

Ao verificar os resultados de uma única execução (Fig. 9), observa-se que eles são diferentes daqueles predominantes quando todas as outras réplicas foram consideradas. Ambos os resultados, replicados e execução única, são válidos e representativos, pois dada uma determinada entrada, há a saída correspondente, porém fica evidente a necessidade de várias execuções para a obtenção do comportamento médio do sistema. Esse tratamento estatístico é bastante difundido e conhecido por textos clássicos de avaliação de desempenho, como (JAIN, 1990), bem como para aqueles específicos para o tema desta tese (HELLERSTEIN *et al.*, 2004).

[1] Sítio: random.org <http://random.org>

(a) 250 réplicas. (b) 10.000 réplicas.

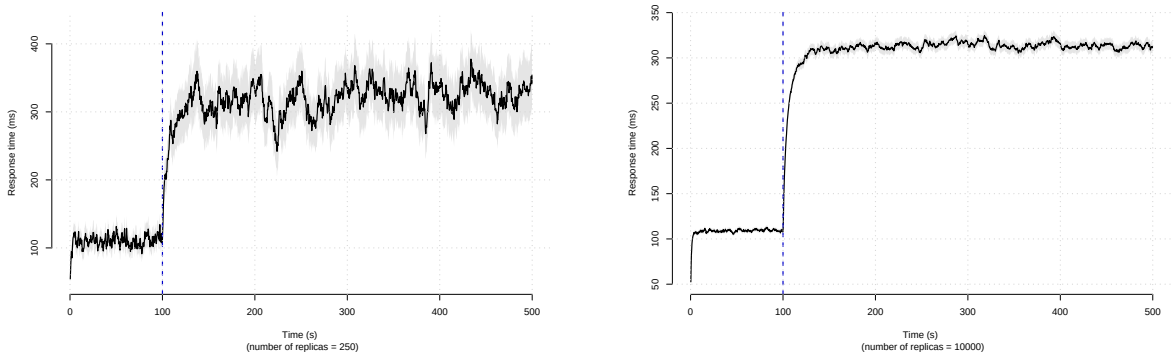


Figura 7 – Saída do sistema: tempo de resposta. Após uma perturbação brusca na entrada do sistema, a variável tempo de resposta levou um tempo até que seu novo patamar de estacionariedade fosse atingido. A linha vertical no instante de tempo 100 s destaca o momento em que o degrau aconteceu.

(a) 250 réplicas. (b) 10.000 réplicas.

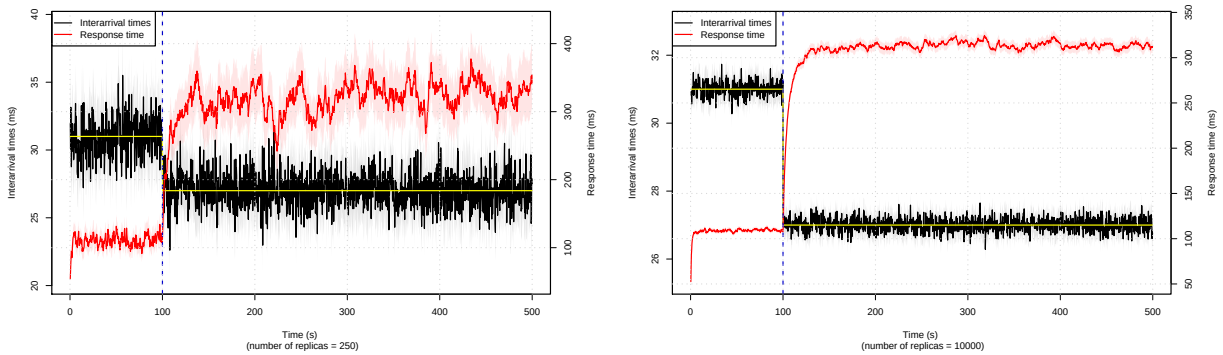
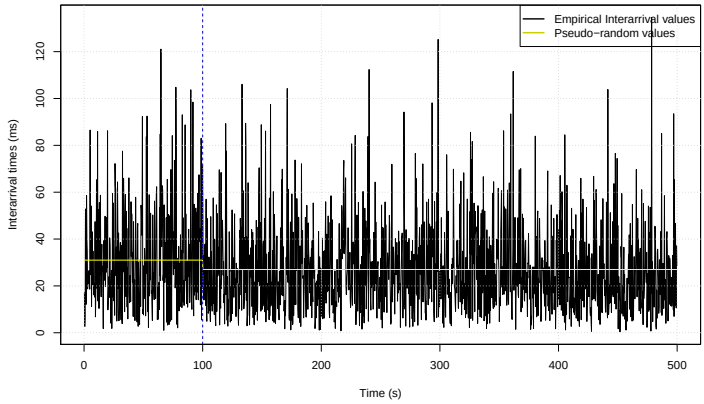


Figura 8 – Entrada e saída juntos. Após a mundança no regime de requisições (entrada), o tempo de resposta (saída) aumenta, apresentando comportamento transiente. A linha vertical no instante de tempo 100 s destaca o momento em que o degrau aconteceu.

O tempo médio necessário para executar o experimento com 250 réplicas foi de 42 s e para o com 10.000 foi de 8 horas e 10 minutos. Esses tempos foram obtidos da média de 20 execuções em uma máquina cuja configuração é: Intel(R) Core(TM) i5-4430 CPU @ 3.00 GHz com 16 GB 1333 GHz de RAM. O espaço em disco necessário para o armazenamento dos resultados foi de 32,4 MB e 1.290 MB para 250 e 10.000 réplicas respectivamente.

Este exemplo demonstra a aplicação de simulação para a obtenção do comportamento transiente de um modelo de fila mais simples e suas implicações. É necessário que várias execuções sejam realizadas a fim de obter um resultado estatisticamente confiável, porém com um custo computacional que pode tornar a aplicação dessa abordagem inviável na prática. Outro ponto é que caso a amplitude do degrau fosse diferente (i.e de 31 para 29), mais replicações seriam necessárias. Caso outra forma de carga de trabalho servisse de entrada para o sistema, igualmente, mais replicação seriam necessárias. Ou seja, a verificação do comportamento transiente do sistema para diferentes casos, exige a execução de experimentos. Em modelos mais complexos e

(a) Entrada.



(b) Saída.

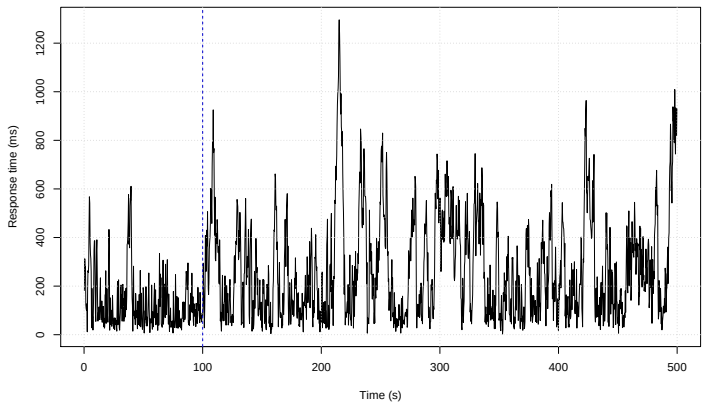


Figura 9 – Entrada e saída do sistema (execução única). Para a entrada (a), os valores foram gerados conforme uma distribuição exponencial com média 31 ms até o instante de tempo 100 s e 27 ms no restante da simulação. Para a saída (b), o padrão de tempo de resposta é alterado após a perturbação. A linha vertical no instante de tempo 100 s destaca o momento em que o degrau aconteceu. Resultados de uma única execução.

compostos (redes de filas) a simulação tende a ser mais custosa computacionalmente.

É nesse sentido que Yang e Liu (2012) propõem a abordagem por função de transferência como ferramental matemático para a modelagem do comportamento transiente. Nesse contexto, a função de transferência é classificada como metamodelo (BARTON, 2015) e utilizada para extrair informação em regimes operacionais em que um sistema de filas está em transiente. Para a identificação do sistema (estimação dos parâmetros do modelo de função de transferência), experimentos específicos são realizados. Desse ponto em diante a simulação de condições operacionais diversas podem ser realizadas com operações menos complexas e que representam bem o sistema em estudo. Essa abordagem tem sido utilizada para aplicações de Teoria de Controle em sistemas computacionais e é o tema detalhado na seção 2.2.

Há estudos sobre a solução de transiente para sistemas não markovianos: Redes de Petri (GERMAN, 2000; HORVáTH *et al.*, 2012) e Modelos de Filas (JAIN; MOHANTY;

BÖHM, 2006). As duas propostas baseiam-se na mesma abordagem para a formalização do comportamento transiente, isto é, equações diferenciais parciais com resolução no domínio do tempo. No caso das Redes de Petri é necessário utilizar a extensão estocástica para a variável independente tempo seja adicionada à fórmula e, com isso, seja viável uma equação capaz de representar a evolução do sistema no decorrer do tempo. O problema com essas abordagens ainda continua sendo a solução numérica das equações.

2.2 Abordagem baseada em Função de Transferência

Em meados da década de 1990 a Teoria de Controle começa a ser aplicada no domínio de sistemas computacionais. No início, a teoria de controle foi aplicada a sistemas computacionais para resolver, principalmente, problemas de garantias de parâmetros de QoS para aplicações distribuídas. Esse foi o primeiro passo para que se criasse um ponto inicial no qual a aplicação das técnicas de controle nesse domínio se consolidasse como uma linha de pesquisa.

Como já apontado pela literatura (MüLLER; KIENLE; STEGE, 2009; HELLERSTEIN *et al.*, 2005), a computação autônoma tem seus princípios fundamentados em teoria de controle realimentado. Um requisito básico para sua implementação é o monitoramento de uma variável de saída periodicamente a fim de verificar seu comportamento em relação a um valor de referência desejado. A discrepância entre esses valores é utilizada como sinal de autocorreção. A variável de entrada é então manipulada para que o sistema tenha o desempenho desejado e diminua a diferença do valor de referência e o valor atual. Esse processo é mapeado no modelo conceitual *Monitoring, Analyze, Plan, Execute - Knowledge Base* (MAPE-K) da IBM (HORN, 2001). No entanto, em contraste com essa abordagem, ferramentas de análise e síntese de dispositivos autoadaptativos desenvolvidas ao longo de várias décadas na teoria de controle, que de fato correspondem às técnicas operacionais para síntese desses dispositivos, são ainda timidamente difundidas e empregadas nos domínios da computação (MüLLER; KIENLE; STEGE, 2009; HELLERSTEIN *et al.*, 2005).

A teoria de controle tem se desenvolvido como um ramo proeminente nas engenharias e ciências naturais, e conta com um rico arcabouço de ferramentas de modelagem matemática para descrever o comportamento de sistemas dinâmicos em termos da resposta à diferentes estímulos e para verificar propriedades de estabilidade, observabilidade de estados e controlabilidade. Seu instrumental analítico oferece extensiva fundamentação para o projeto de estratégias de controle aplicáveis a sistemas lineares, não lineares, contínuos e discretos. Um ponto importante é observar o ferramental analítico e de simulação oriundo dessa área de conhecimento dispõe de métodos que permitem a análise do comportamento transiente de sistemas dinâmico, possibilitando a compreensão no mapeamento causa e efeito entre as variáveis de entrada e saída do sistema em estudo.

No entanto, a aplicação dessa teoria como ferramenta para análise de desempenho é

menos explorada. Exemplos dessas aplicações podem ser observados em soluções para gerenciamento de recursos computacionais agrupados (*cluster*) com a finalidade de controle de *throughput* em mecanismos de controle de congestionamento em redes de comunicação (ABDELZAHER *et al.*, 2003; LU *et al.*, 2001; LU *et al.*, 1999) e também em técnicas de escalonamento retroalimentado (*feedback scheduling*) para sistemas de tempo real (LU *et al.*, 2003; SHA *et al.*, 2004; ABDELZAHER *et al.*, 2003; LU *et al.*, 2001).

Teoria de controle tem sido utilizada para controle de admissão adaptativo em arquiteturas multicamadas(KAMRA; MISRA; NAHUM, 2004), controle de latência em servidores Web (HENRIKSSON; LU; ABDELZAHER, 2004), bem como aplicações em computação em nuvem (LIM *et al.*, 2009), alocação de recursos dinamicamente (LEVA; PAPADOPOULOS; MAGGIO, 2013), redes de computadores (XIA; SUN; TIAN, 2009), computação verde (WANG *et al.*, 2011) (*green computing*). Considerada ainda uma formação mais específica das engenharias, o reconhecimento da ampla aplicabilidade aos sistemas computacionais, especialmente os de grande escala e complexidade, as oportunidades têm inspirado também iniciativas explicitamente dedicadas a tornar mais acessível a Teoria de Controle a cientistas da computação (HELLERSTEIN *et al.*, 2005) por meio de publicações especialmente dirigidas a esse fim (ZHU *et al.*, 2009; LUNZE; LEHMANN, 2010; HARJUNKOSKI; NYSTRöM; HORCH, 2009).

Além das propostas que usam métodos estatísticos e que envolvem técnicas de inteligência artificial, a teoria de controle vem sendo usada recentemente pela computação como uma proposta de solução reativa para gerenciamento de recursos quando sistemas são submetidos a situações imprevisíveis (LU *et al.*, 2003; CHEN *et al.*, 2005; URGAONKAR *et al.*, 2005; PADALA *et al.*, 2007; LIM *et al.*, 2009; XIONG *et al.*, 2010; FILIERI; HOFFMANN; MAGGIO, 2014). Há de se mencionar que vários problemas de sistemas de computação podem ser resolvidos usando teoria de controle como, por exemplo, problemas de contenção na camada de enlace de dados para redes sem fio (BOGGIA *et al.*, 2007), uso otimizado de recursos computacionais em um *cluster* de processadores e problemas de escalonamento de tarefas em servidores (WANG; CHEN, 2008).

No contexto de arquiteturas orientadas a serviço de larga escala, a adaptação automática constitui um conceito muito atraente e um desafio é desenvolver mecanismos que consigam simultaneamente garantir o cumprimento de ambos requisitos funcionais, concentrados toda a lógica do negócio, e não funcionais, concentrados os requisitos do negócio. No Laboratório de Sistemas Distribuídos e Programação Concorrente, onde desenvolve-se este trabalho, duas teses de doutorado abordaram teoria de controle para otimização de recursos em ambientes distribuídos: (1) um controlador proporcional adaptativo é utilizado para economia de recursos e energia em datacenters(NOBILE, 2013); outro (2) sobre o levantamento de requisitos para o desenvolvimento de *benchmarks* capazes de gerar de carga de trabalho não estacionários (MAMANI, 2016).

O que há em comum nesses trabalhos, cuja abordagem é fundamentada em Teoria de Controle, é que todos eles baseiam-se no conceito de Função de Transferência como modelo

capaz de representar a dinâmica de um sistema computacional. Ao especificar um sistema em que a variável de saída é em função da variável de entrada e essa relação ocorre temporalmente (ou seja, há inércia), as condições transientes podem ser modeladas (PAPADOPOULOS *et al.*, 2015). O ponto de entrada para solução do problema é diferente daquele apresentado na seção anterior (2.1). A forma com que o problema é abordado é diferente. Uma Função de Transferência se traduz em uma equação diferencial no domínio do tempo, mas como sua solução é feita no domínio da frequência (Laplace ou Z—apresentados em forma de aplicação nos capítulos seguintes desta tese), a álgebra envolvida nas equações são simplificadas. Na sequência estão a descrição de trabalhos importantes que contribuíram para a proposta e abordagem da presente tese.

Diao, Hellerstein e Parekh (2006) argumentam que o rápido crescimento da escala em que sistemas computacionais têm realizado destaca a importância lidar com o desafio de controlar tais sistemas. Um arcabouço para descrever problemas de controle em sistemas de grande escala é introduzido. São considerados duas dimensões de expansão o aumento da escala sistema em si, expressos em forma de atuadores (entrada) e sensores (saída), e a política, estruturada em objetivos e requisitos dos usuários. Ao usar este arcabouço, é proposta um arquitetura de controle que abrange abordagens centralizadas e distribuídas. São apontados desafios de pesquisa nessa área como sendo de natureza de latência e decomposição das políticas. A estratégia adotada foi a de decompor o problema maior em pequenos sistemas controlados e compor essas partes em uma arquitetura hierárquica.

Os desafios apontados que dizem respeito a **Políticas** são: (1) *decomposição dos objetivos da política*: dado um requisito muitas vezes abstrato e de alto nível como traduzi-los de forma a serem factíveis de implementação; (2) *disparidade entre atuadores*: diferentes sistemas possuem diferentes interfaces de interação, a forma com que configura-se o número de processos em um servidor web pode ser diferente para cada implementação (i.e: Apache vs. Ngnx); (3) *otimização multi-usuário*: pode existir conflito entre os interesses de usuários diferentes; (4) *métricas para políticas*: muitas métricas relacionadas a QoE são difíceis de serem medidas e integradas com o sistema a ser controlado.

Em relação ao **sistema**: (1) *integração*: quando os sistemas crescem em escala a decomposição do fluxo pode implicar em comportamento operacional diferente do sistema sem a integração; (2) *restrições temporais*: problemas de tempo-real no contexto de sistemas distribuídos; (3) *lidar com atrasos e não determinismos*: lidar com a estocasticidade inerente dos sistemas, principalmente em relação à latência; (4) *disponibilidade de métricas e atuadores*: extrair informações úteis e que não estão disponíveis e desenvolver interfaces que permitam alterar a configuração do sistema em operação.

Liu *et al.* (2003) investigaram como a otimização dos parâmetros de configuração podem aumentar o desempenho de um servidor web, especialmente em ambientes operacionais em que há variações na carga de trabalho. Os autores apresentam abordagens para tratar de otimização

do servidor Apache com foco no parâmetro `MaxClients`, que controla o número máximo de processos concorrentes para atender requisições. Através de medidas empíricas e modelagem analítica foi possível abordar o problema de otimização como *hill climbing*. Três estratégias de solução foram propostas uma baseada no método de Newton, outra como *fuzzy* e outra que explorava o relacionamento entre os gargalos de utilização e minimização do tempo de resposta. Houve melhora significativa nas implementações, mas com alguns problemas o método de Newton mostrou-se sensível a variações; *fuzzy* é robusto mas converge lentamente e a heurística é de difícil generalização. O problema mostrou-se mais complexo do que o enunciado, pois a configuração é um processo no qual componentes são integrados ou ajustados para atender determinados níveis de desempenho. A complexidade de configuração é o maior impedimento para *deploy* e gerenciamento de sistemas computacionais. Brown e Hellerstein (2004) descrevem uma abordagem para quantificar a complexidade da configuração de modo a servir como requisitos para especificação de *benchmark* que explore essa característica. A abordagem é baseada em procedimentos, ou seja, tenta explorar a quantidade de interação humana que deve existir para que o sistema atenda uma nova configuração.

O alto custo operacional de sistemas computacionais de larga escala, tem chamado a atenção para soluções que diminuam a interação humana no gerenciamento desses sistemas. Teoria de Controle muito utilizada em outras áreas da engenharia pode ser aplicada para atacar esse problema, pois apresenta um conjunto ferramental elaborado para que auxilia na construção de sistemas com características de autorreparo e autodiagnóstico (DIAO *et al.*, 2005b). E ainda fornecem meior para quantificação de estabilidade, diminuição do tempo de transição e autorregulação precisa. Diao *et al.* (2005a) apresentam um arquitetura baseada no modelo MAPE-K da IBM (HORN, 2001) que mostrou-se genérico o suficiente para que servisse de referência para novas implementações. É discutido os benefícios da utilização dessa abordagem analítica para descrever como controlar sistemas computacionais.

Uma aplicação dessa abordagem foi feita para o caso de Banco de Dados Lightstone *et al.* (2007). Os autores argumentam que Teoria de Controle é pouco explorada no domínio de aplicação de sistemas de gerenciamento de banco de dados devido à falta de expertise dos desenvolvedores e também da ausência de modelos e estudos anteriores para tal aplicação. Nessa linha, é mostrada a robustez dessa técnica de controle e contribuem ao destacar questões importantes no desenvolvimento de banco de dados auto gerenciáveis. Discute-se sobre problemas e limitações, há o contraste da implementação entre os métodos tradicionais e malha fechada de controle.

Um problema é lidar com a não linearidade dos sistemas (COADY *et al.*, 2005). Muito embora os modelos empregados em Teoria de Controle clássico sejam lineares, eles funcionam bem para sistemas computacionais. Pode-se fazer uma aproximação linear em uma região de operação que comporta-se de forma linearmente aproximada, caso em que o modelo é útil. No entanto, pode haver situações em que os sistemas saem dessa região e entram em condições

operacionais não contempladas no processo de identificação. Essa é uma área que ainda possui muitos problemas abertos e cuja solução depende de um estudo mais detalhado dos sistema a fim de se obter uma maior cobertura operacional. Nessa linha, Casolari, Tosi e Presti (2012) propõem uma abordagem adaptativa para a detecção de alterações em tempo de execução. O modelo de análise é em forma de fluxo (*stream*), de forma que uma variável de interesse é extraída do sistema web em questão. Continuamente, é feita a monitoria dessa variável, em que há presença de altos níveis de variabilidade e não estacionariedade. É feita uma análise de métodos tradicionais para atacar esse tipo de problema, avaliando inclusive a eficiência computacional dos métodos. O método proposto utiliza *wavelet* para remover o ruído do sinal e um parâmetro regra adaptativa usada para a detecção. A abordagem mostrou-se robusta e possível de identificar variações em diferentes cenários.

A necessidade de Diferenciação de Serviço (*service differentiation*) em aplicações de Internet motivam a utilização de aplicações web multi-camadas com desempenho controlado (DIAO *et al.*, 2006). Os autores apresentam a arquitetura T2T (*tier-to-tier*) que possibilita o gerenciamento descentralizado de atuadores, bem como uma implementação (*testbed*). São exploradas questões de procedimentos experimentais próprios e modelos analíticos que consideram a eficiência e a granularidade do controlador. Foi mostrado que para uma implementação efetiva de estratégia de controle, foi necessário instrumentar o ponto exato em que os gargalos ocorrem, em vez de tentar atacar a camada por completo. A implementação utilizou o IBM WebSphere como servidor de aplicação e o IBM DB2 para camada de dados. O esquema de controle foi descentralizado, o que permitiu desacoplar componentes do sistema que operam de forma independente — isso foi possível ao aumentar a granularidade das variáveis de controle utilizadas.

Considerando o fato de que a Função de Transferência é capaz de modelar o desempenho de um sistema computacional e fornece métricas temporais, como por exemplo o tempo de assentamento (*settling time*), Stankovic *et al.* (1999) foi um dos primeiros pesquisadores a utilizar essa técnica para escalonamento de tempo real. A proposta foi a de adicionar a política clássica *Earliest Deadline First* — EDF uma propriedade de retroalimentação. Dessa forma, a *Feedback Control — Earliest Deadline First* — FC-EDF incorpora um controlador do tipo Proporcional Integral Derivativo — PID como mecanismo para a implementação dessa política. Todas as tarefas são consideradas independentes, e a arquitetura proposta pode ser verificada na Figura 10. A métrica de desempenho considerada foi a taxa de perdas (*Deadline Miss Ratio* — DMR), cujo a referência é preferencialmente zero ($DMR = 0$). O mecanismo de atuação é concentrado em um controle de admissão e um priorizador na fila de requisições (controlador de nível de serviço).

Estes dois controladores (admissão e nível de serviço) foram incluídos no escalonador servem para lidar com a sobrecarga de requisições que podem aumentar indefinidamente a taxa de utilização sentida pelo sistema. O controlador PID usa o erro da DMR para calcular quantos processadores são necessários para processar as requisições que aguardam para serem

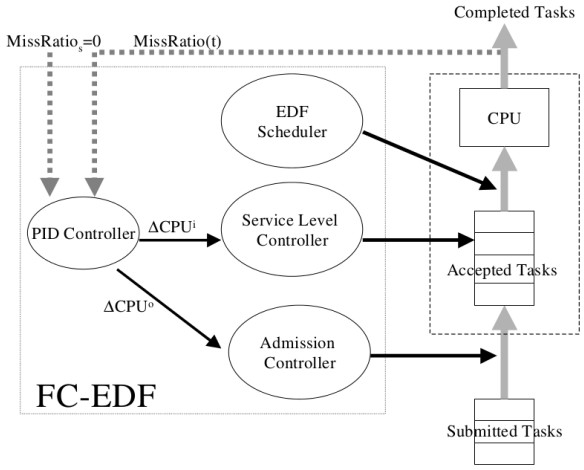


Figura 10 – Arquitetura do FC-EDF (STANKOVIC *et al.*, 1999).

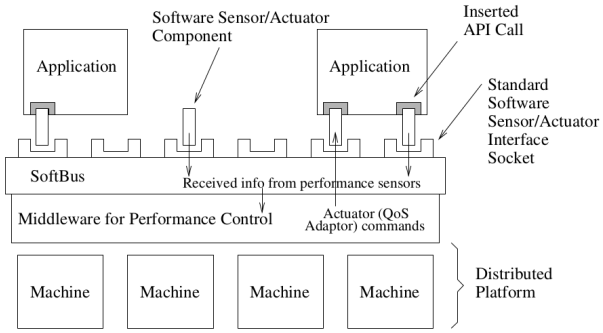


Figura 11 – Visão geral da arquitetura do ControlWare (ABDELZAHER; STANKOVIC, 2002).

processadas no sistema. Outros trabalhos envolvendo escalonamento realimento foram propostos por (LU *et al.*, 1999; STANKOVIC *et al.*, 2001; LU *et al.*, 2002; LU *et al.*, 2006).

Abdelzaher e Stankovic (2002) descrevem o projeto e a implementação de um *middleware* para controle de parâmetros QoS. A Figura 11 ilustra a arquitetura do ControlWare. A arquitetura possui a figura de um "barramento" (SofBus) que é um protocolo de comunicação distribuído. O SoftBus serve de ligação entre a aplicação e as máquinas (servidores, sensores, atuadores e controladores) que hospedam o serviço.

Garantir o atraso de QoS relativa em servidores Web (especificamente Apache) é o tema da pesquisa de Lu *et al.* (2003). É proposto um arcabouço de um controlador que possui características tradicionais (*feedback*) e preditivas (*feedforward*). A arquitetura é ilustrada na Figura 12. A ideia é que, (1) dada uma determinada carga de trabalho, uma previsão é realizada na tentativa de prever o tamanho das filas e (2) o controle realimentado para alocação de recursos. Cada classe de serviço possui um previsor que antecipa o tamanho das filas nos servidores. Uma quantidade de recursos é calculada e sugerida para o sistema computacional em estudo para atender uma classe de serviço, considerando os parâmetros de QoS. É feita uma combinação

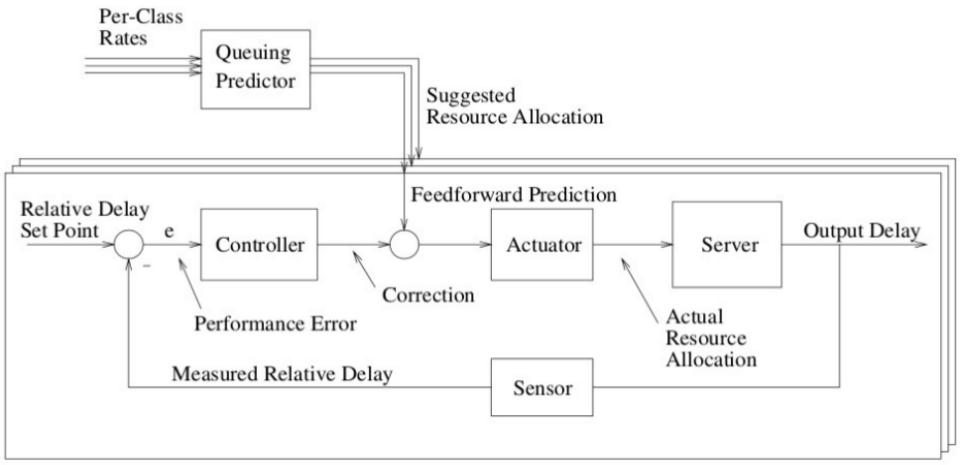


Figura 12 – Arquitetura genérica que combina filas de controle realimentado (LU *et al.*, 2003).

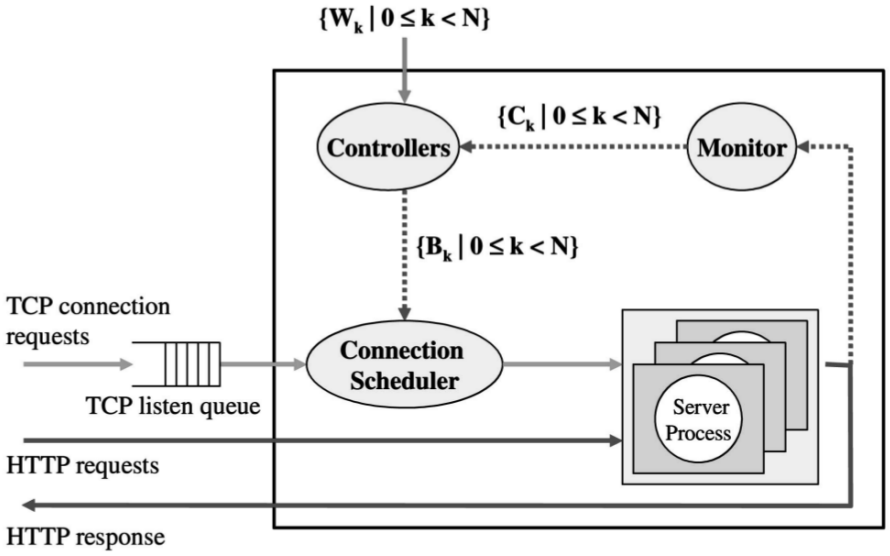


Figura 13 – Arquitetura para garantia de atrasos usando controle realimentado (LU *et al.*, 2006).

entre a quantidade de recursos que o sistema necessita, oriundas do controlador, e a quantidade estimada.

Ainda no tema de garantia de QoS absoluta em servidores Web, Lu *et al.* (2006) apresentam como solução uma arquitetura adaptativa que tenta mitigar o *overhead* de conexões TCP e assim tenta controlar o atrasos entre as requisições feitas ao servidor por um cliente. Há argumentação que uma parte considerável do tempo é tomada pelo algoritmo de conexão do protocolo TCP. Nesse sentido, controlam a quantidade de processos que atenderão as requisições.

Urgaonkar *et al.* (2005) apresenta um proposta para a alocação dinâmica de recursos para aplicação Web de múltiplas camadas. É utilizado teoria de filas para estimar a capacidade estacionária e estratégias preditivas e reativas são fornecidas para que seja feito o ajuste dessa estimativa em tempo de execução.

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

A alocação dinâmica de recursos como ela é implementada em provedores de computação em nuvem (Amazon e Azure por exemplo) serve como aplicação prática para a estratégia de trabalho que modelam sistemas computacionais como uma Função de Transferência. Com características de utilizações globais e flutuações na carga de trabalho, a capacidade torna-se variável também. Neste caso, entender o comportamento transiente da aplicações é fundamental para que custos sejam minimizados (LAI *et al.*, 2005; BUYYA; RANJAN; CALHEIROS, 2009; JIANG; CUI; CHEN, 2009; AN; LESSER, 2010; LEWIS; MARROW; YAO, 2010; WOLSKI *et al.*, 2001).

Em (KUSIC; KANDASAMY, 2007) criam um framework para alocação dinâmica de recursos usando um *Limited Lookahed Control* (LLC), juntamente com um modelo de previsão, com o objetivo de maximizar o rendimento. A figura 14 ilustra o modelo do sistema para as três classes de serviço propostas pelos autores. Cada classe possui um *cluster* de máquinas virtuais e uma fila de requisições.

O aumento e a diminuição da capacidade de cada *cluster* é feita pelo controlador ilustrado na Figura 15. As informações de intervalo entre as chegadas e tempo médio de processamento são utilizadas para alimentar um modelo que prediz a capacidade necessária em cada *cluster*. O controlador observa o sistema real e toma decisões que ajustam a saída gerada pelo modelo.

Aplicações Web multicamadas são adequadas para serem implantadas (*deploy*) em ambiente de nuvem, e muita pesquisa tem sido feita em como pode ser o gerenciamento de cada uma das camadas que compõem a aplicação. Como um dos desafios apresentados por Huang, He e Miao (2014) está adequação de modelos para serem utilizados no projeto de controladores que tenham esse propósito. Outro ponto levantado foi sobre como projetar sistemas robustos em que sejam capazes de lidar com cargar diferentes daquelas iniciais. Apontam que a identificação de sistemas é uma forte ferramenta para aumentar a qualidade dos modelos gerados. Nessa linha, modelagens direcionadas a *datacenters* podem ser realizadas, conforme *framework* apresentado por Klems, Nimis e Tai (2009).

Em (FITO; GOIRI; GUITART, 2010) há a proposta de uma arquitetura completa de hospedagem de nuvem, como foco nos provedores, com possibilidade de negociação entre data centers para que os recursos sejam vistos pelo consumidor como ilimitados. A arquitetura proposta permite que data centers tenham seus recursos aumentados temporariamente por

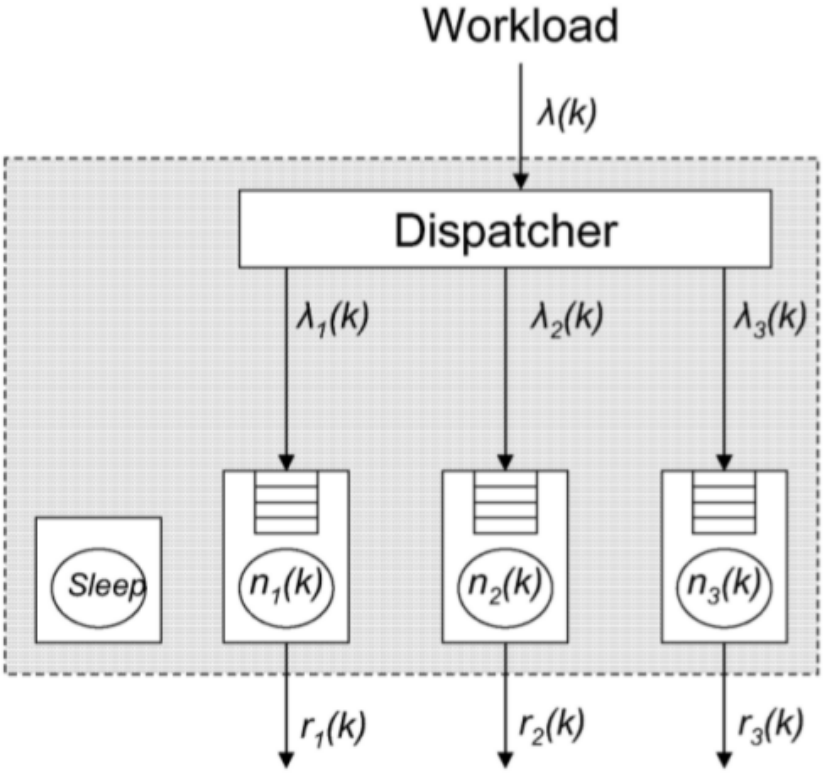


Figura 14 – Modelo do sistema para as classes de serviço ouro, prata e bronze (KUSIC; KANDASAMY, 2007).

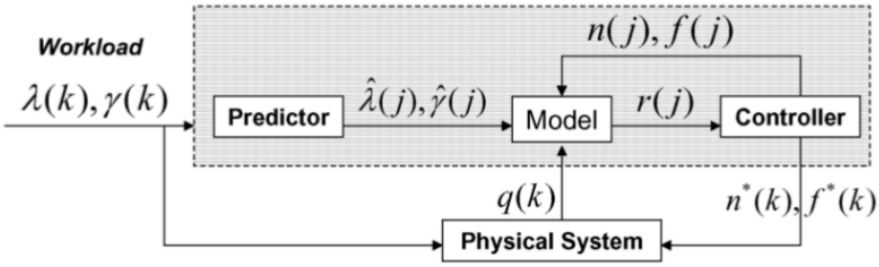


Figura 15 – Diagrama de blocos do controlador para as classes ouro, prata e bronze (KUSIC; KANDASAMY, 2007).

negociação com outros provedores. A arquitetura proposta visa diminuir as penalizações sofridas pelo provedor nos casos de quebra de contrato de prestação de serviços.

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

A totalidade dos trabalhos revistos que utilizam alguma técnica de modelagem dinâmica em problemas computacionais o fazem como artifício auxiliar matemático para resolução de sistemas de equações, e como ferramenta para projeto de sistemas de controle. A presente tese, por outro lado, dá foco nos aspectos relacionados à abordagem metodológica para a modelagem de comportamento transiente, especialmente voltado para avaliação de desempenho, contribuindo com a proposta da análise de resposta em frequência como ferramenta para predições da relação dinâmica entre desempenho e carga de trabalho.

CAPÍTULO

3

MODELAGEM DINÂMICA

A obtenção do modelo dinâmico de desempenho por métodos experimentais é baseado em técnicas de identificação de sistema. Este capítulo apresenta os aspectos conceituais para formulação do modelo e o método empírico de sua obtenção por abordagem de caixa-preta. A fim de elaborar cada passo, é interessante desenvolver o protocolo experimental por meio de um exemplo hipotético comumente estudado em modelagem de sistemas computacionais: um modelo de filas simples.

3.1 Exemplo de um modelo M/M/1

Com o intuito de introduzir o estudo da dinâmica de desempenho do comportamento de um sistema discreto sob variação de carga, considere-se um modelo simplificado de um serviço conforme ilustrado na Figura 16.

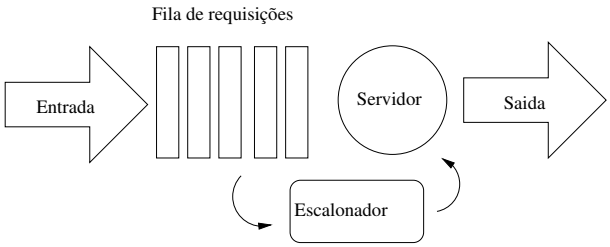


Figura 16 – Modelo simplificado de estudo.

Seja o objetivo o de obter um modelo analítico transiente a partir de dados obtidos experimentalmente por método de caixa-preta.

No exemplo, a carga de trabalho (entrada) é dada por uma distribuição exponencial que caracteriza o fluxo de requisições que chega ao sistema, o qual é parametrizado por dois valores: tempo médio entre chegadas (taxa de chegada) $\Delta\lambda$ e tempo médio de processamento (taxa de execução) $\Delta\mu$ das requisições. Quando a taxa de chegada é superior à capacidade de

atendimento do servidor, as requisições pendentes aguardam em uma fila de tamanho infinito até que sejam processadas. Uma vez atendidas, as requisições deixam o sistema. Não há preempção ou recirculação de requisições. O bloco "Escalonador" representa a política de fila FIFO que seleciona qual a próxima requisição é enviada para o servidor sempre que este termina o processamento da requisição anterior. Este sistema é amplamente descrito como M/M/1, segundo a notação de Kendall.

Para o estudo realizado neste ensaio, considera-se que existe apenas um único usuário que inicialmente envia requisições a uma taxa constante. O contrato de nível de serviço estabelecido com o usuário especifica um limite superior de tempo médio de resposta calculado em uma janela deslisante de tamanho w (i.e. a média das w requisições anteriores), configurada para tamanho 1.

Os objetivos do ensaio é (1) realizar a modelagem deste sistema e representá-lo como um sistema de primeira ordem segundo o modelo auto-regressivo ARX (*Autoregressive with Exogenous Inputs*) e (2) avaliar seu comportamento dinâmico após uma perturbação de carga do tipo degrau.

3.2 Protocolo experimental

O referencial teórico básico utilizado na elaboração deste método é abordado em (LJUNG, 1999; SOUZA; PINHEIRO, 2008; ZHU *et al.*, 2009; HELLERSTEIN *et al.*, 2004) em contextos diversos. Esta seção sistematiza os conceitos gerais e técnicas úteis em um protocolo experimental para avaliação de desempenho, compreendendo a formulação do modelo dinâmico, identificação da sua estrutura e obtenção da sua parametrização, a partir de experimentos de caixa-preta. O exemplo materializa os passos para o exemplo $M/M/1$, bem como tece as ressalvas que devem ser observadas durante a execução de cada etapa.

3.2.1 Passo 1: Determinação do sistema

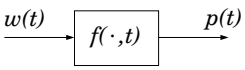


Figura 17 – Modelo dinâmico de desempenho.

Conforme delineado nas considerações introdutórias, o objetivo do arcabouço de avaliação de desempenho é produzir um modelo dinâmico como da Figura 3 (página 29), reproduzida convenientemente na Figura 3 em que

$$y(t) = f(x(t), t)$$

Nesse *framework* conceitual, a definição de sistema é simplesmente a relação entre o sinal $x(t)$ (carga de trabalho) e o sinal $y(t)$ (desempenho). Assim, o sistema em questão pode representar algum dispositivo tal como CPU, memória ou uma plataforma computacional completa, bem como pode representar simplesmente a conversão entre duas medidas, não necessariamente vinculadas a um aparato físico, como a relação entre o número de *threads* e a energia dissipada. O ferramental proposto para sua modelagem requer a satisfação de algumas hipóteses, mais precisamente, que o sistema seja linear e invariante no tempo (LTI), ou possa ser razoavelmente aproximado sob essas restrições.

Dessa forma, o objetivo do primeiro passo é gerar um artefato que identifique a variável de entrada e a variável de saída, bem como verificar se as condições para uma aproximação LTI é válida. No exemplo corrente, o objetivo é avaliar o comportamento do tempo de resposta das requisições quando há uma alteração do tipo degrau na demanda por um serviço. Portanto, $y(t)$ é o **tempo de resposta** e $u(t)$ é **tempo entre as chegadas** de requisições no sistema.

Caso o relacionamento entre as duas seja do domínio do analista de desempenho, as hipóteses podem ser assumidas por conhecimento *a priori*. Caso seja desconhecido, deve-se realizar uma avaliação preliminar que permita inferir como o valor de y é alterado em função de u. Isso pode ser feito por meio de uma avaliação estacionária com finalidade exploratória.

No caso do modelo M/M/1, sabe-se (JAIN, 1990) que o tempo médio de resposta é obtido por

$$R = \frac{1/\mu}{1-\rho} \tag{3.1}$$

e que

$$\rho = \frac{\lambda}{\mu} \tag{3.2}$$

sendo λ a taxa de chegada de requisições por unidade de tempo e μ, a taxa de serviço por unidade de tempo. A taxa de utilização do sistema é denotada por ρ.

Pode-se escolher alguns valores para o tempo médio de serviço, $\Delta\mu$, em unidades de tempo, *ut*, aplicar a fórmula 3.1, variar o valor do tempo médio entre as chegadas, $\Delta\lambda$, e produzir o gráfico da Figura 18. Vale mencionar que os valores para a fórmula 3.2 são descritos em taxa, portanto são obtidos pelo seu valor inverso: $\rho = (1/\Delta\lambda)/(1/\Delta\mu)$.

Os valores obtidos por esta análise são importantes porque eles ditarão os possíveis valores para a região de operação do sistema. A escolha de um valor depende da aplicação em questão. Neste caso, como se trata-se de um sistema genérico, pode-se optar arbitrariamente pelo valor de $\lambda = 28$.

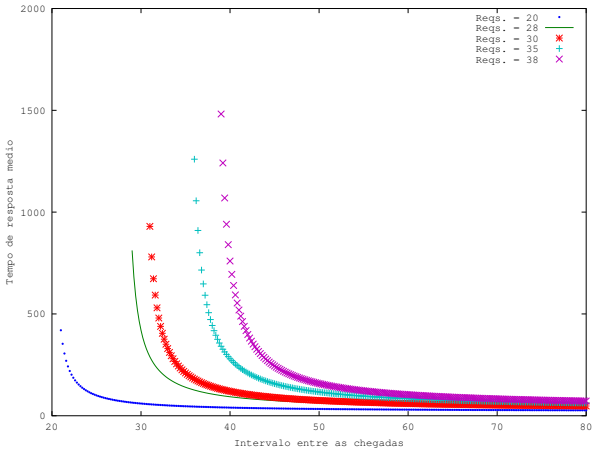


Figura 18 – Relacionamento entre u em função de y. O sistema é estável à medida que aumenta-se os valores de u. O valor mínimo deve ser observado, pois caso $u < y$ o sistema é instável. Apenas sistemas estáveis devem ser considerados.

Uma vez escolhido o parâmetro, deve-se dividir a curva em três regiões. Para $\mu = 28$, tem-se:

1. Linear: correspondente ao comportamento linear. Faixa de valores de 29–30;
2. Quase linear: maior parte da curva é linear, o restante por ser aproximado. Faixa de valores de 29–31;
3. Não linear: a curva não apresenta comportamento linear. Faixa de valores 29–34.

Neste ponto, conhece-se o sistema e sabe-se o quão sensível a variável de saída é em relação àquela de entrada. Sabem-se também as regiões dos valores de entrada para que o sistema possa ser aproximado por um modelo LTI. As predições do modelo dinâmico de desempenho serão, potencialmente, melhores, nas regiões mais lineares. A despeito do fato de que em grande parte dos casos, sistemas computacionais são marcadamente não lineares, os resultados obtidos durante o desenvolvimento desta pesquisa demonstram que, mesmo em condições longe de ideais, os resultados obtidos pela abordagem são (ao contrário da primeira intuição) bastante razoáveis. De fato, observou-se que, em muitos casos, sistemas mais complexos, embora tenham componentes altamente não lineares, possuem uma dinâmica resultante que presta-se, muitas vezes, à modelagem na forma de sistemas simples de baixa ordem.

3.2.2 Passo 2: Determinação do escopo

O escopo do modelo refere-se a três características a serem consideradas: estocasticidade, estrutura e carga de trabalho.

É muito comum em sistemas computacionais que haja uma forte presença de elementos que adicionam incertezas no comportamento de uma variável, por exemplo, a forma com que requisições chegam a um serviço, requisições feitas a um servidor *web* e o tráfego de pacotes

em uma rede baseada em TCP/IP. Podem existir ciclos nas requisições a um serviço, onde a quantidade pode aumentar durante o período comercial e ficar praticamente inativo durante a noite; requisições feitas a um servidor podem oscilar conforme o tempo de pensamento (*think time*) de um usuário e ter uma carga muito alta em função de uma data festiva (compras em época de natal); a forma com que os pacotes de uma rede são gerados incluem o tempo de espera baseado em um valor não determinístico.

Quando se trata de um sistema com características aleatórias, as execuções experimentais devem ser repetidas várias vezes, porque o resultado de uma única execução pode não representar o sistema de forma fidedígna. O ideal é realizar várias réplicas de um experimento, e, dos valores coletados, realizar um tratamento estatístico em que é possível extrair valores de intervalo de confiança.

No estudo de caso, o sistema é basicamente regido por componentes estocásticos. O sistema M/M/1 é aqui solucionado através de simulação de eventos discretos com a biblioteca Simpack (FISHWICK, 1992). Com os parâmetros $\Delta\mu = 28$ e $\Delta\lambda = 31$. A simulação teve duração 300 K unidades[1] de tempo e no instante 100 K sofreu um degrau negativo de amplitude -2 (entrada u tempo entre as chegadas; saída y tempo de resposta) . O gráfico 19 apresenta o resultado de uma execução em contraste com outro referente à média de 250 replicações com intervalo de confiança de 95%.

(a) Resultado de uma execução.

(b) Resultado da execução de 250 réplicas. Intervalo de confiança de 95%.

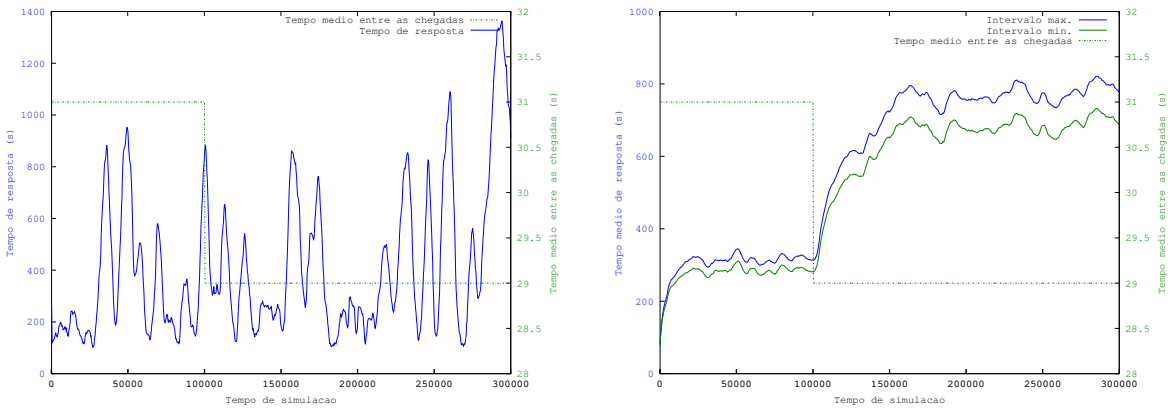


Figura 19 – Efeito da estocasticidade do sistema.

A estrutura do modelo refere-se à ordem da dinâmica presente. Uma dinâmica de ordem zero significa que a saída segue a entrada imediatamente: $y(t) = Cu(t)$. Uma dinâmica de primeira ordem significa que a saída é influenciada pela sua própria taxa de variação. Essa relação foi intuitivamente delineada no Capítulo 1 por meio do exemplo do equilíbrio térmico (página 30), em que o modelo contínuo da dinâmica de temperatura do sistema é da forma

[1] Utiliza-se a notação n K para referir-se ao valor $n \times 10^3$, p.ex. 300 K = 300.000.

$\dot{y}(t) + ay(t) = u(t)$. No domínio de tempo discreto k, a derivada da variável $y(t)$ (denotada $\dot{y}$), corresponde à diferença $y(k) - y(k-1)$, conforme é possível concluir a partir da relação entre a diferenciação contínua e discreta

$$\dot{y}(t) = \lim_{\Delta t \to 0} \frac{y(t+\Delta t) - y(t)}{\Delta t} \tag{3.3}$$

$$\dot{y}(k) = \lim_{\Delta t \to T_s} \frac{y(t+\Delta t) - y(t)}{\Delta t} \qquad = \frac{y(k+T_s) - y(k)}{Ts} \tag{3.4}$$

onde T_s é o tempo de amostragem. Nesse estudo, a escala de tempo k é sequência dos índices das amostras, de modo que assume-se $T_s = 1$, chegando à relação $\dot{y}(k) = y(k+1) - y(k)$, ou equivalentemente $\dot{y}(k) = y(k+1) - y(k)$. Dessa forma, a equação de diferenças (diferencial discreta) de primeira ordem do exemplo pode ser apresentada por um modelo autorregressivo com variável exógena (ARX)

$$y(k) = ay(k-1) + bu(k-1) \tag{3.5}$$

A revisão bibliográfica realizada durante essa pesquisa revela que a introdução do modelo ARX para modelagem de sistemas computacionais, no geral, é feita sem tais considerações, e a partir da proposta de modelagem de séries temporais. A proposta de desenvolver o arcabouço conceitual para avaliação de desempenho empreendida na presente pesquisa tem o intuito de apresentar não meramente uma ferramenta matemática, mas um modelo intuitivo para compreender o fenômeno estudado. Por essa razão, o modelo ARX é introduzido a partir da perspectiva de sistema dinâmico e as observações sobre sua conexão com equações de diferenças no domínio contínuo é ressaltada.

A conversão do modelo ARX para a forma da função de transferência pode ser feita algebricamente considerando-se a variável z como um operador de atraso, tal que $z^{-1}y(k) = y(k-1)$. A partir daí, tem-se

$$y(k) = ay(k-1) + bu(k-1) \tag{3.6}$$

$$y(k) = az^{-1}y(k) + bz^{-1}u(k) \tag{3.7}$$

e, denotando a transformada Z de $y(k)$ por $Y(z)$, ou abreviadamente Y

$$Y = azY + bzY \tag{3.8}$$

$$Y(1 - az^{-1}) = bz^{-1}Y \tag{3.9}$$

$$F = \frac{Y}{Y} = \frac{b.z^{-1}}{1 - az^{-1}} = \frac{b}{z - a} \tag{3.10}$$

Tabela 1 – Planejamento de experimentos para escolha do tempo de amostragem adequado.

Utilização	Tempo de amostragem		
70% (28/40)	300	1200	9000
95% (38/40)	300	1200	9000

Novamente, esta é uma passagem algébrica prática. A noção intuitiva que não deve ser perdida de vista é que a variável z é uma forma compacta de representar, no domínio da frequência, a solução da equação de diferenças no domínio do tempo. Conforme ilustrado pelos exemplos do Capítulo 1 (como o sistema térmico e o sistema massa-mola), sistemas LTIs estáveis apresentam dinâmica geral oscilatória amortecida. Matematicamente, $z = Ae^{-i\omega k} = A(\cos\omega k + isin\omega k)$, o que fisicamente corresponde ao comportamento oscilatório de amplitude A e frequência ω embutido na relação diferencial. Formalmente, a transformada Z de $y(k)$ é $Y(z) = \sum_{k=0}^{\infty} z^{-1} y(k)$.

Para selecionar estrutura do modelo, a proposta sugere o princípio de parcimônia na escolha do modelo mais simples que possa representar a dinâmica observada. A simplicidade, nesse caso, é a ordem do sistema. Assim, deve-se optar pela ordem mais baixa e, caso essa não represente bem o sistema, recomenda-se aumentar sua ordem gradativamente. Preferencialmente, modelos de primeira e segunda ordem devem ser considerados, pois simplificam seu tratamento e as conclusões que podem ser obtidas diretamente a partir deles.

A carga de trabalho refere-se a alterações na variável de entrada u. O esforço maior neste caso é identificar o melhor tempo de amostragem para a variável de saída y. Ao passo que tempos de amostragens muito baixos acarretam em uma variabilidade maior dos valores obtidos (e pode capturar excessivamente o ruído estocástico), um tempo de amostragem baixo suaviza, porém elimina a caracterização do acompanhamento de uma variável em função da outra.

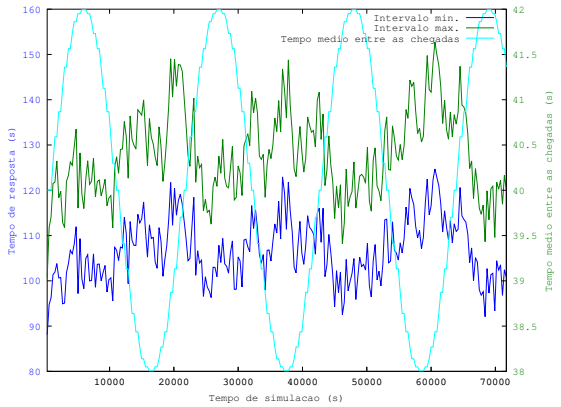
3.2.3 Passo 3: Excitação do sistema

Para realizar a escolha do tempo de amostra, deve-se implementar um gerador de carga de trabalho que seja capaz de excitar o sistema de forma adequada. Um bom exemplo de gerador de cargas nesse sentido é uma onda senoidal (HELLERSTEIN *et al.*, 2004), pois excursiona repetidamente pela faixa de operação selecionada. É possível explorar uma larga faixa de valores. O Capítulo 4 discute requisitos para projeto de instrumental para geração de carga e medição experimental do sinal de saída.

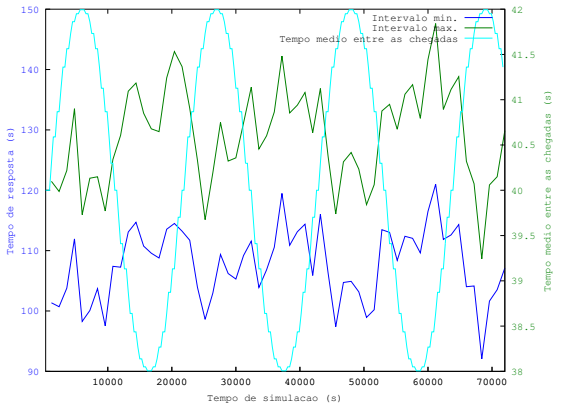
Uma vez implementado o gerador, deve-se realizar experimentos que identifiquem o comportamento da variável de resposta em relação a diferentes valores para o tempo de amostragem. Neste exemplo, planejou-se os experimentos conforme descrito na Tabela 1. A variável de saída y corresponde ao tempo de resposta de cada requisição.

A excitação ao sistema foi feita por meio de uma onda sinoidal de período 2100 *ut* com uma variação do valor da métrica de entrada de 5%. Os valores são coletados por meio de simulação com duração de 72000 *ut*. O resultados são apresentados na figura 20.

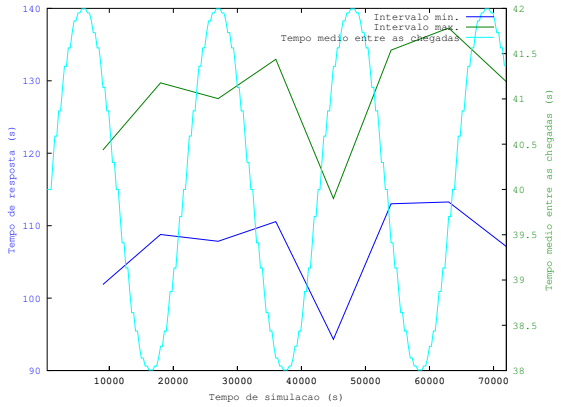
(a) $\rho = 0.7$ e tempo de amostragem $\Delta s = 300$.



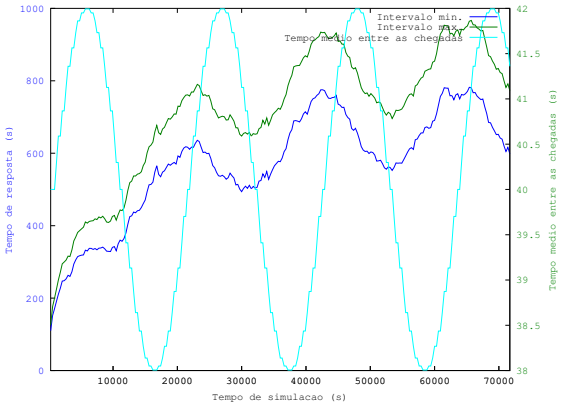
(b) $\rho = 0.7$ e tempo de amostragem $\Delta s = 1200$.



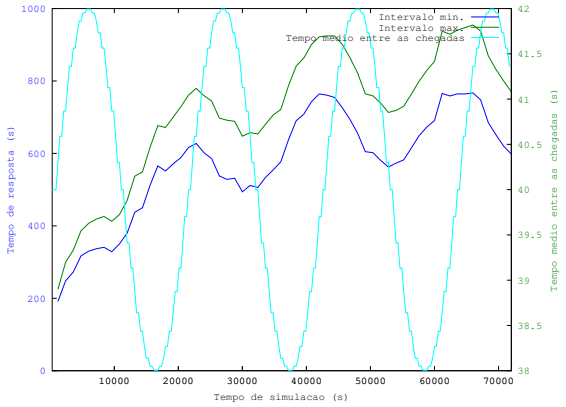
(c) $\rho = 0.7$ e tempo de amostragem $\Delta s = 9000$.



(d) $\rho = 0.95$ e tempo de amostragem $\Delta s = 300$.



(e) $\rho = 0.95$ e tempo de amostragem $\Delta s = 1200$.



(f) $\rho = 0.95$ e tempo de amostragem $\Delta s = 9000$.

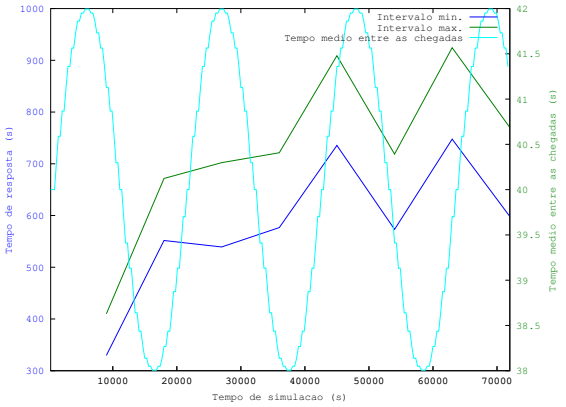


Figura 20 – Diferentes tempos de amostragem para a variável de saída y. Valores médios computados de experimentos com 250 réplicas e intervalo de confiança 95%

Tabela 2 – Planejamento dos experimentos exploratórios

Parâmetros	Valores
Tempo de simulação	300 K
Período da onda	87,5 K
Alterações em $\Delta\lambda$ por período	10 K
Tempo médio de serviço $\Delta\mu$	28
Tempo de amostrgem	300
Faixas exploradas $\Delta\lambda$	29–32
	29–35
	29–41
	*35–40

Para uma carga de trabalho de 70% ($\rho = 0.7$), os valores variam muito. À medida que se aumenta o tempo de amostragem, há um suavização dos valores obtidos. Devido a essa grande variação, não se pode obter uma decisão sobre qual o melhor tempo de amostragem. Para $\rho = 0.95$, tem-se uma nitidez maior dos valores e, novamente, a suavização dos valores é proporcional ao tamanho do tempo de amostragem. Pode-se escolher o valor 300 (escolhido neste trabalho), porém poder-se-ia escolher o valor 1200 igualmente. Esta última escolha dar-se-ia em casos onde o custo de obtenção dos valores é alto.

3.2.4 Passo 4: Identificação do modelo

A identificação do sistema trata do mapeamento de valores de execuções do sistema em um modelo de regressão linear que o represente fidedignamente. Para se alcançar essa identificação são necessárias a obtenção dos dados e seu processamento e a execução de um algoritmo para obtenção de parâmetros para aplicação no modelo. O resultado é que, uma vez feita a estimativa dos parâmetros, será possível realizar a previsão dos valores na instância de uma nova execução.

3.2.4.1 Etapa 4a: experimentação

Esta etapa corresponde à realização dos experimentos para a modelagem. No exemplo, de posse das faixas de valores obtidos na fase de conhecimento do sistema, tem-se os três níveis. Esses níveis correspondem a três regiões de operação que devem ser exploradas. A ideia principal é conduzir experimentos exploratórios em cada uma dessas faixas a fim de produzir dados para a estimativa dos parâmetros. No estudo de caso abordado, estipula-se o planejamento descrito na tabela 2.

Os resultados dos experimentos exploratórios das três primeiras faixas são mostrados na Figura 21. O principal problema com eles é a falta de simetria quando o valor de u está em sentido crescente ou decrescente. Quando está no sentido crescente, há um padrão, ou seja, os valores são agrupados, como pode ser observado nas Figuras 20d, 20e e 20f. Este fato motiva a escolher preferencialmente uma faixa de valores com comportamento mais previsível para o

modelo estudado. Assim, especifica-se adicionalmente a faixa de valores 35–40. Observa-se, entretanto, que os resultados da pesquisa sugerem que, mesmo face à assimetria sob entradas positivas e negativas, os sistemas estudados nos capítulos posteriores, produziram resultados satisfatórios quando aproximados por modelos lineares, o que é uma observação interessante.

Os mesmos experimentos exploratórios foram repetidos para a faixa de valores 35–41 e os resultados são mostrados na figura 22. Pode-se observar (Figura 21a) que a variável de resposta y (tempo médio de resposta) responde às alterações feita na variável de entrada u (tempo médio entre as chegadas de requisições) com alguma variabilidade no valores obtidos. A gama de valores explorados (Figura 21b) é uniforme dependendo do valor de u. Intuitivamente, um modelo com parâmetro estimados com estes dados provavelmente terá melhores resultados.

Uma observação deve ser feita em relação à quantidade de alterações nos valores da variável de entrada u que são feitas durante um período da onda gerada como variação na carga de trabalho. A escolha de um valor baixo para a alteração no período é um problema, como mostrado na figura 23, pois explora fracamente a gama de valores possíveis para diferentes entradas. Pode-se observar que determinados valore para uma entrada estão indeterminados.

3.2.4.2 Etapa 4b: parametrização

Esta etapa corresponde à estimação dos parâmetros do modelo. De posse dos dados, pode-se iniciar a estimativa do parâmetros para o modelo ARX através do método *least square regression* (DRAPER; SMITH, 1998). A estimativa é feita por meio de 5 coeficientes como mostrado nas equações de 3.11 a 3.17.

$$S_1 = \sum_{k=1}^{N} y^2(k) \tag{3.11}$$

$$S_2 = \sum_{k=1}^{N} u(k)y(k) \tag{3.12}$$

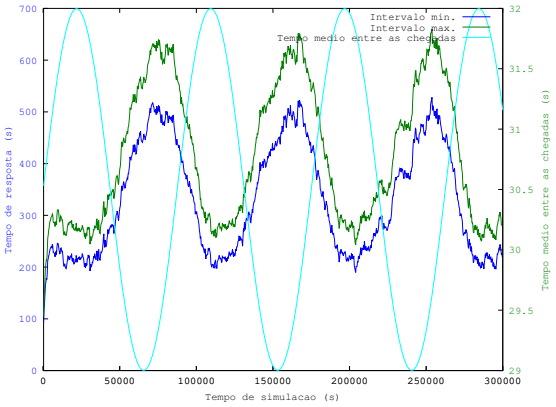
$$S_3 = \sum_{k=1}^{N} u^2(k) \tag{3.13}$$

$$S_4 = \sum_{k=1}^{N} y(k)y(k+1) \tag{3.14}$$

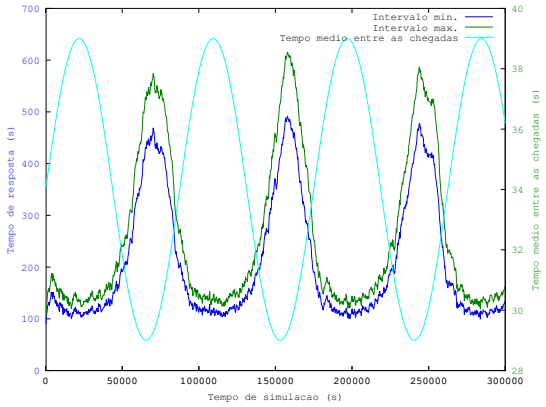
$$S_5 = \sum_{k=1}^{N} u(k)y(k+1) \tag{3.15}$$

$$a = \frac{S_3S_4 - S_2S_5}{S_1S_3 - S_2^2} \tag{3.16}$$

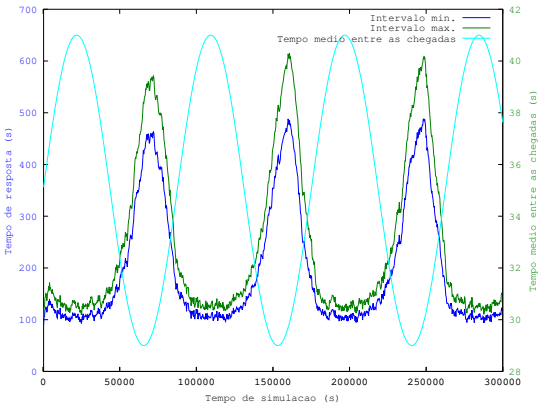
(a) Faixa 29–32, taxa de utilização muito alta ($\sim 90\%$)e simetria durante a alternância de $\Delta\lambda$.



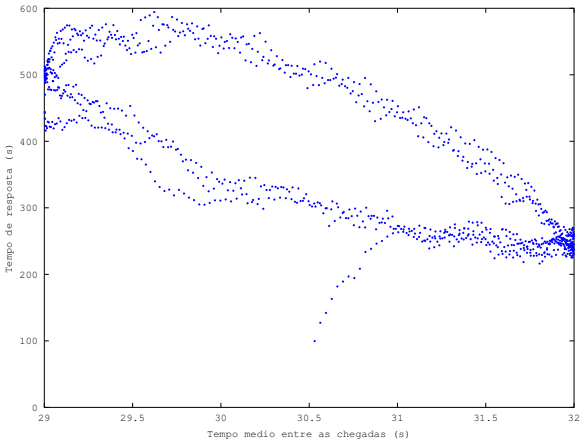
(b) Faixa 29–35, taxa de utilização alta ($\sim 80\%$) e assimetria durante a alteranância de $\Delta\lambda$.



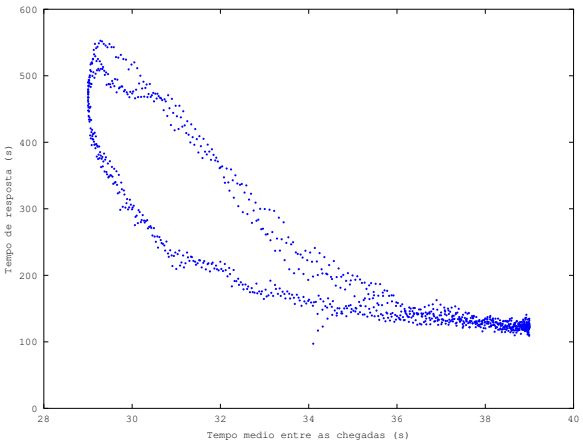
(c) Faixa 29–41, taxa de utilização alta ($\sim 78\%$) e assimetria durante a alternância de $\Delta\lambda$.



(d) Faixa 29–32, alteração da variável de saída y em função da variável de entrada u. Comportamento linear durante a subida e a descida, porém há presença de agrupamentode valores.



(e) Faixa 29–35, alteração de y em função de u. Comportamento assimétrico durante a subida e a descida e com agrupamento de valores.



(f) Faixa 29–41, y em função de u. Comportamento assimétrico durante a subida e a descida e com agrupamento de valores.

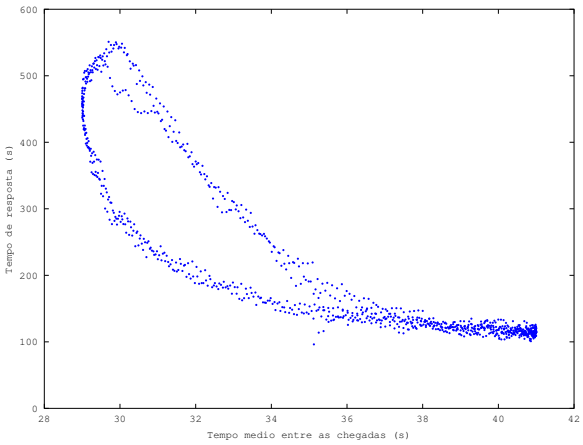
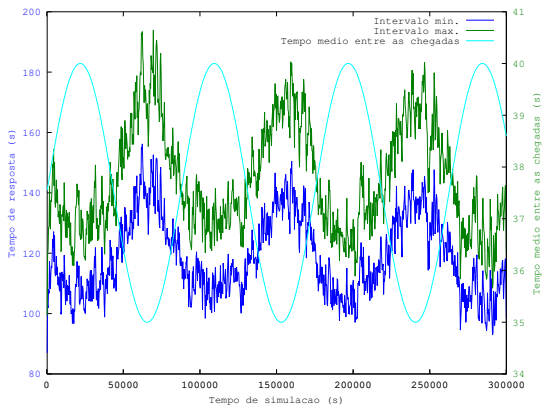


Figura 21 – Experimentos exploratórios. Comportamento não linear para as faixas escolhidas. (250 execuções; intervalo de confiança de 95%)

(a) A variável de resposta y responde aos estímulos de y.



(b) Comportamento simétrico de y em função de u.

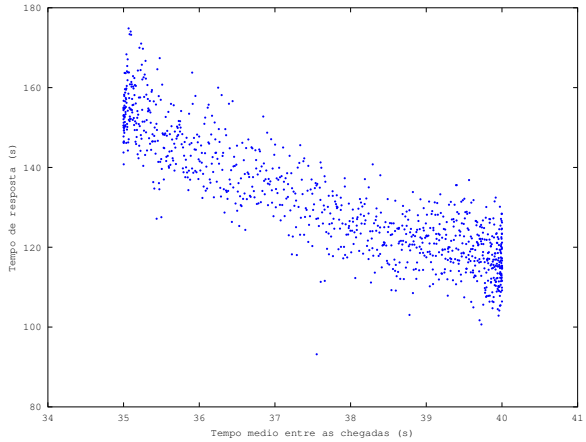
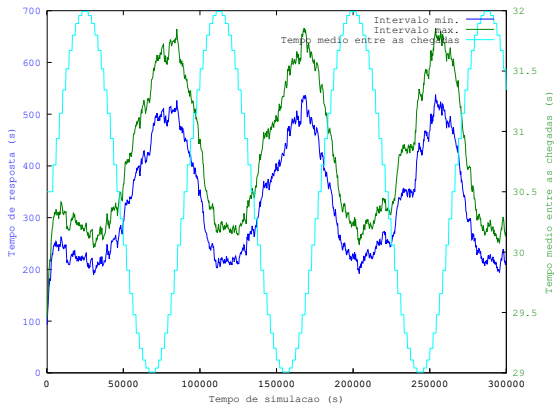


Figura 22 – Experimentos exploratórios para a faixa 35–40.

(a) texto aqui.



(b) texto aqui.

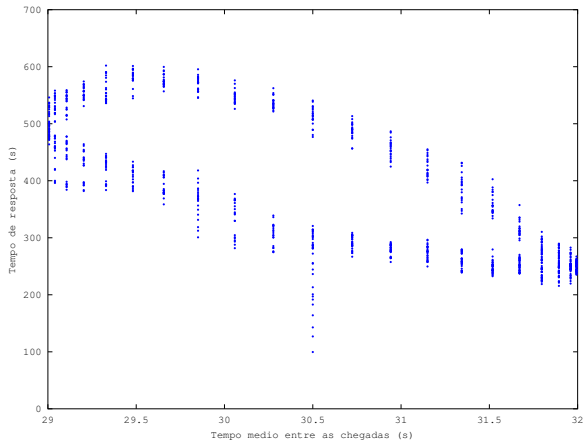


Figura 23 – Um total de 42 alternâncias para cada período da onda e a baixa exploração dos possíveis valores de y em relação a u.

$$b = \frac{S_1 S_5 - S_2 S_4}{S_1 S_3 - S_2^2} \tag{3.17}$$

A tabela 3 sumariza os valores para a e b do modelo ARX do exemplo corrente. Todos os valores de a foram positivos e menores que 1, o que significa que os valores para a variável de saída y (tempo de resposta) não irão oscilar durante o período transiente[2]. Neste caso, o valor negativo para b significa que o ganho será negativo. Ou seja, aumentar o valor de u significa diminuir o valor de y.

Com os valores dos coeficientes do modelos, pode-se calcular os valores das métricas de estado transiente.O Teorema do Valor Final pode ser aplicado, pois $|a| < 1$. A Equação 3.18

[2] Os parâmetros associados à variável de saída y correspondem aos pólos da função de transferência. Pode ser demonstrado que a região de convergência (ROC) de um LTI discreto é o círculo unitário no domínio Z. Esse fato é explicado pela teoria de variáveis complexas e pode ser consultada na bibliografia referenciada.

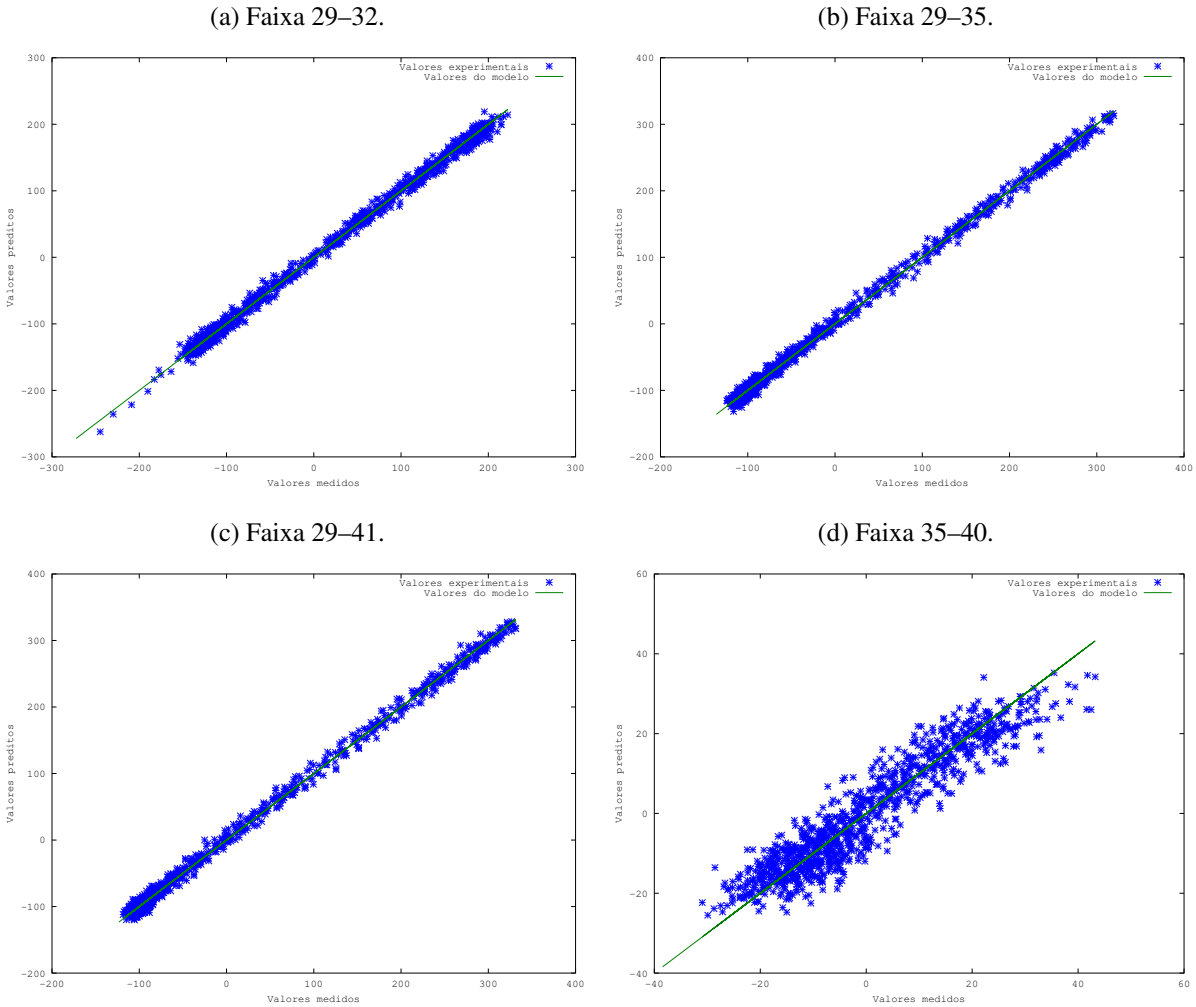


Figura 24 – Comparação entre os valores experimentais e os preditos. Quando mais os valores preditos (*) coincidirem com os experimentais (-), melhor será a capacidade de predição do modelo na sua região de operação.

refere-se ao ganho, ou seja, se alterado o valor de entrada em uma unidade o quanto o valor de saída é modificado; Equação 3.19 é o valor final da variável de saída após uma alteração de seu valor, sendo u_{ss} o novo patamar de valor que a entrada foi direcionada; Equação 3.20 estima a duração, em intervalos de amostragem, do período transiente; Equação 3.21 o máximo valor que a variável irá atingir quando um estado de transição se iniciar. A dedução desses resultados pode ser encontrada em (HELLERSTEIN *et al.*, 2004):

$$G(1) = \frac{b}{1-a} \tag{3.18}$$

$$R' = \frac{b}{1-a} u_{ss} \tag{3.19}$$

$$\Delta T_{tr} = \frac{-4}{log|a|} \tag{3.20}$$

Tabela 3 – Coeficientes encontrados.

Região	a	b
29–32	0.96486	-4.87183
29–35	0.97072	-1.16624
29–41	0.97395	-0.87718
35–40	0.64910	-2.69454

Tabela 4 – Identificação do sistema para diversas regiões de entrada

Região	Taxa de utilização $\overline{\rho}$	$(\overline{u}, \overline{y})$	(a, b)	RMSE	R^2
29–32	91.41%	(30.632, 377.15)	(0.96486, -4.87183)	7.0705	0.99718
29–35	81.30%	(34.440, 231.69)	(0.97072, -1.16624)	7.2164	0.99703
29–41	78.81%	(35.528, 218.47)	(0.97395, -0.87718)	6.9266	0.99723
35–40	74.23%	(37.720, 131.60)	(0.64910, -2.69454)	5.2386	0.88191

$$M_p = (1-a)R' \tag{3.21}$$

3.2.5 Passo 5: verificação

Esse passo corresponde à verificação do ajuste do modelo. A avaliação estatística pode ser feita pelos métodos *root-mean-square-erros* — RMSE (Equação 3.22) e variabilidde R^2 (Equação 3.23). Quanto menor o valor de RMSE, melhor é o modelo; quanto mais próximo de 1 o valor de R^2, melhor é o modelo.

$$RMSE = \sqrt{\frac{1}{N}\sum_{k=1}^{N}[y(k+1)-\hat{y}(k+1)]^2} \tag{3.22}$$

$$R^2 = 1 - \frac{var(y-\hat{y})}{var(y)} \tag{3.23}$$

Pode-se obter o valor médio dos valores de entrada $\overline{u}$ e saída $\overline{y}$ de forma a se estimar o valor médio das entradas e das saídas e com isso encontrar a taxa média de utilização média $\overline{\rho}$.

No exemplo, as faixas de valores de 29–32 e 29-35 possuem taxa de utilização muito alta, ao passo que as duas última, 29–41 e 35–50, possuem valores para RMSE mais baixo e utilização menor. Muito embora subjetivo, possivelmente, elas podem representar melhor uma situação em que o sistema em observação pode vir a ficar em estado crítico de execução, por exemplo, subir do patamar de 75% para 95% de taxa de utilização.

Tabela 5 – Planejamento de experimentos de teste

Fator	Nível	Antes	$\overline{\rho}$ antes	Degrau	Depois	$\overline{\rho}$ depois
Alteração	Δt	100 K	–	–	100 K	–
Entrada	$\Delta\mu$	28	–	–	28	–
Faixa	29–32 $\Delta\lambda$	30.632	91.41%	−1.1583	29.4737	95%
	29–35 $\Delta\lambda$	34.440	81.30%	−4.9663	29.4737	95%
	29–41 $\Delta\lambda$	35.528	78.81%	−6.0543	29.4737	95%
	35–40 $\Delta\lambda$	37.720	74.23%	−8.2463	29.4737	95%

3.2.6 Passo 6: Validação

Esse passo corresponde aos experimentos realizados para avaliar as predições realizdas pelo modelo. Nesta fase, experimentos que realizam um alteração do tipo degrau na carga de trabalho são conduzidos nos sistema físico. Os resultados resultados servirão para confrontar o resultados obtidos do sistema real (físico ou simulado) com aqueles do modelo analítico.

O trabalho maior nesta etapa é desenvolver um gerador para a carga de trabalho com as alterações desejadas. Esse desenvolvimento é dependente da aplicação e deve ser validada. No presente estudo de caso, a validação foi feita através dos valores médios do tempo de resposta do estado estacionário do sistema. A validação ficará clara nos gráficos mostrados nos passos seguintes.

Os experimentos constaram da inicialização do sistema com uma carga constante dos valores médios de $\overline{u}$ e $\overline{\mu} = 28$ para todas as faixas de valores. A execução durou 300 K *ut*, tendo a taxa de utilização elevada ao patamar de 95% após o instante de tempo 100 K *ut*. Os demais parâmetros foram mantidos iguais aos da tabela 2. O planejamento dos experimentos de teste é mostrado na tabela 5.

Os experimentos são executados 250 vezes e é calculada uma média com intervalo de confiança de 95%.

Em seguida aos experimentos, é feita uma análise comparativa dos dados obtidos pela execução dos testes com os dados que o modelo prediz. Antes de a comparação ser possível é necessário um entendimento sobre como a alteração da carga se passou e como adicioná-la ao modelo.

A função de transferência de um sistema pode ser obtida pela Equação 3.24.

$$G(z) = \frac{Y(z)}{U(z)} \tag{3.24}$$

Como deseja-se obter o valor de resposta y, primeiro deve-se isolar este termo e realizar as devidas alteraçãoes. (Equação 3.25)

$$Y(z) = G(z)U(z). \tag{3.25}$$

Tabela 6 – Equações de resposta no domínio do tempo

Faixa	Equação
29–32	$y(k) = -4.87183(0.96486)^{k-1}$
29–35	$y(k) = -1.16624(0.97072)^{k-1}$
29–41	$y(k) = -0.87718(0.97395)^{k-1}$
35–40	$y(k) = -2.69454(0.64910)^{k-1}$

Um sistema de primeira ordem é dado pela Equação 3.26.

$$G(z) = \frac{b}{z-a}. \tag{3.26}$$

A Transformada Z que representa um sinal de alteração do tipo degrau é representada pela Equação 3.27.

$$U(z) = \frac{z}{z-1}. \tag{3.27}$$

Portanto a função de transferência que representa o comportamento da alteração de carga realizada nos experimentos de testes é feita como na Equação 3.28.

$$Y(z) = G(z)U(z) = \frac{b}{z-a}\frac{z}{z-1} = \frac{bz}{(z-a)(z-1)} \tag{3.28}$$

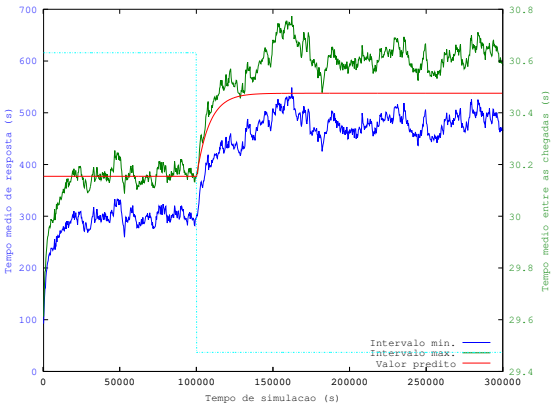
Para este caso, é possível obter a sua inversa e gerar a equação no domínio do tempo, como mostrado na Equação 3.29.

$$y(k) = b(a)^{k-1} \tag{3.29}$$

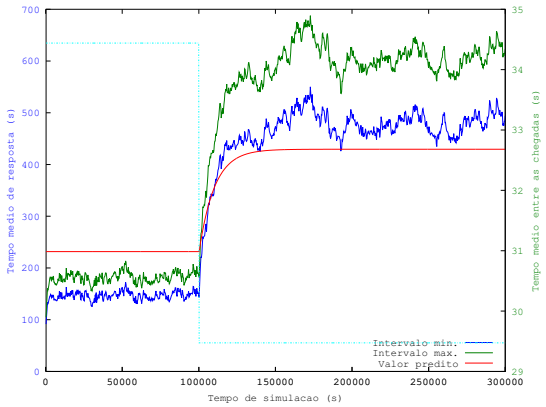
A tabela 6 relaciona as faixas de valores e suas respectivas equações no domínio do tempo, após a 100ª K *ut*.

A Figura 25 apresenta os resultados obtidos entre a execução do sistema real com o modelado. Os modelos com o melhor desempenho para este são os das faixa 29–41 e 29–35. Com estes testes pode-se visualizar o tempo necessário que o sistema leva até que haja absorção de uma alteração na carga de trabalho e com os parâmetros encontrados para o modelo estimar os valores do tempo de assentamento, o que contribui para o projeto de sistemas autocientes desempenho. Deve-se ressaltar que os valores obtidos não são satisfatórios. Isso provavelmente se deve ao fato de o novo patamar em que a taxa de utilização chegou foi fora de sua região de operação.

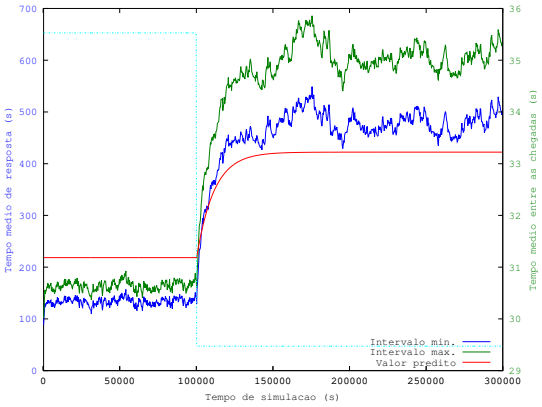
(a) Faixa 29–32.



(b) Faixa 29–35.



(c) Faixa 29–41.



(d) Faixa 35–40.

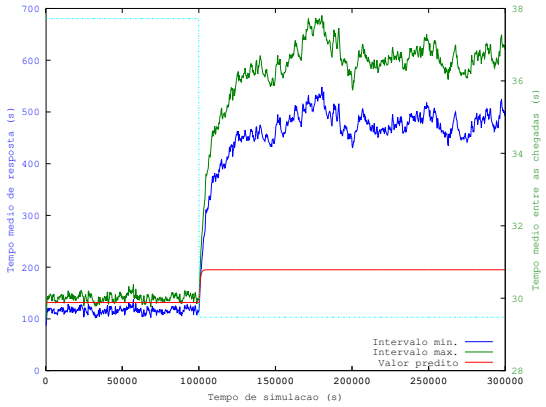


Figura 25 – Contraste entre os dados obtidos pelas execuções experimentais com aqueles obtidos através de simulação com o modelo.

CAPÍTULO

4

INSTRUMENTAÇÃO EXPERIMENTAL

Esta seção introduz um modelo referencial de requisitos em que são separadas responsabilidades para projeto de experimentos de avaliação de desempenho não estacionária de sistemas computacionais com o objetivo de obtenção de modelos dinâmicos de desempenho.

O modelo foi desenvolvido a partir das necessidades observadas durante a pesquisa, quando foi preciso estender as funcionalidades de *frameworks* de simulação e benchkarking a fim de realizar os experimentos. A aplicação para esse modelo foi testada para simulações e também em um benchmarking de aplicação de *e-commerce*, o que evidencia sua generalização e sua utilidade. A contribuição possui natureza metodológica e é fundamentada na racionalização sistemática de problemas encontrados durante a análise e execução desta tese (PEREIRA JR. *et al.*, 2015; PEREIRA *et al.*, 2015; MAMANI *et al.*, 2015).

4.1 Modelo de responsabilidadese MEDC

A base conceitual para esta proposta está diretamente ligada ao conceito de desempenho como expressão quantitativa de quão bem um sistema atende uma demanda, quer seja medido como *throughput*, taxa de utilização, taxa de falhas ou outra medida de interesse. Por sua vez, a demanda é oriunda do esforço imposto pela carga de trabalho em que o sistema está exposto e pode ser parametrizada por fatores de interesse, por exemplo, taxa de requisições, custo computacional etc. Nessa conceituação proposta, a capacidade é a função que transforma demanda em desempenho. Logicamente, o desempenho é dependente do recursos empregados para que a funcionalidade seja implementada (processadores, canais de comunicação, memória e assim por diante).

Atualmente, devido à presente complexidade da interligação entre diferentes sistemas e, internamente, de seus próprios mecanismos, bem como ao não determinismo da execução concorrentes dos processos, é desafiador adotar uma metodologia dedutiva baseada na abordagem

das partes para o todo (*bottom-up*) e assim construir modelos analíticos para sistemas de grande escala. Quando esse é o cenário, há, pelo menos, duas alternativas. Uma trata-se da criação de modelos "sintéticos" expressos em forma de simulação. Diferentemente dos modelos analíticos, cujos resultados são fundamentalmente teóricos, os modelos de simulação que tem como foco modelar sistemas computacionais que imitam o comportamento operacional, e os resultados são produzidos a partir de deduções estatísticas. Usualmente, muitas réplicas são executadas para obter um resultado confiável. O problema dos cenários em que alguns parâmetros são alterados (o caso "what-if" já discutido no Capítulo 2), formam a base para a segunda alternativa, que consiste em obter medidas de entrada e saída do sistema em estudo e, a partir dos dados empíricos, deduzir um modelo capaz de explicar como o relacionamento causa-efeito entre as variáveis envolvidas ocorre. Testes controlados podem ser executados no sistema em estudo para produzir os dados com a qualidade necessária para que bons modelos sejam estimados. O modelo obtido é considerado como metamodelo (YANG; LIU, 2010)

Os modelos de interesse desta tese são aqueles capazes de capturar a dinâmica do sistema. Essa propriedade faz com que o modelo seja capaz de prever como alterações internas (mudanças na capacidade) e externas (carga de trabalho) impactam no desempenho, temporalmente. A análise em regime transiente permite observar que mudanças abruptas podem levar o sistema a situações de instabilidade, ou até mesmo, sua inutilização. Analogamente, pode indicar que o desempenho insatisfatório em determinado instante de tempo irá retornar à sua normalidade após os efeitos do evento transitório se dissipem.

A maioria das ferramentas encontradas para avaliação de desempenho, no sentido abordado nesta tese, é direcionada à avaliação estacionária e não descreve as abstrações de demanda e capacidade como requisito básico. Portanto, como iniciativa há o esforço de identificar como pode ser feito o relacionamento de diferentes entidades conceituais para que seja possível expressar a implementação de um ambiente de experimentação para que dados de qualidade sejam produzidos para a correta identificação e análise do sistema em estudo.

Considere-se um um experimento em que deseja-se modelar o comportamento dinâmico do desempenho de sistema computacional de interesse no decorrer do tempo, após uma mudança brusca e duradoura na carga de trabalho — um degrau. As abordagens convencionais estacionárias são insuficientes para essa análise. Não se pode calcular a média de todo o experimento para servir como base para estimação de parâmetros do modelo desejado. Mesmo que o resultado seja dividido em duas partes antes e depois do degrau, os dados que correspondem ao transiente são excluídos do cálculo.

Um protocolo de experimentação básico pode ser aplicado para esse exemplo: iniciar a execução do experimento com uma carga de trabalho estacionária, e após algum tempo alterar sua intensidade para o segundo nível estacionário; com capacidade inalterada, realizar amostras periodicamente das variáveis de desempenho de interesse. Para implementar esse protocolo, o analista deve ser capaz de alterar os parâmetros de geração de carga durante a execução do

experimento. Ou seja, ser capaz de estipular que em determinado instante de tempo o conjunto de parâmetros a_1 para geração de carga deve ser alterado para a_2. É fácil observar que com essa abordagem novos tipos de cargas podem ser pensados como crescimento em rampa e exponencial etc. Isso possibilita também a excitação do sistema baseada em senóides, como estudada no Capítulo 6.

Considerando o mesmo protocolo experimental, a capacidade pode ser alterada em função do monitoramento do desempenho observado no sistema. Caso uma função de transferência do sistema computacional tenha sido identificada, é possível elaborar reações proporcionais, integrais e derivativas desse comportamento. O resultado é que, ao alterar a capacidade em tempo de execução (algo comum em Computação em Nuvem), a quantidade de elementos computacionais pode ser ajustada para que atenda de forma adequada as condições operacionais daquele instante de tempo.

Um ponto importante a ser destacado é sobre como as alterações ocorrem em tempo de execução do experimento. O ato de gerenciar máquinas virtuais ou provisionar recursos não é instantâneo, pode haver atrasos entre o instante de tempo que o comando ocorre e a efetiva realização desse comando. A carga de trabalho pode ser atrasada para emular comportamento de redes autônomas, Internet. Falhas e operações de rotina (backup) podem ocorrer — é importante considerá-los em ambientes controlados para adicionar mais realismo ao contexto operacional. Dessa forma, é interessante que seja possível ao projeto de experimentos adicionar tal funcionalidade.

A forma proposta para atender a esses requisitos é base para a especificação do modelo referencial *Monitor*, *Effector*, *Demand* e *Capacity* (MEDC), ilustrado na Fig. 26. O diagrama ilustra quatro responsabilidades essenciais que devem ser atendidas. A responsabilidade **demanda** permite que seja configurada como as requisições serão realizadas no sistema computacional, de forma que seja possível observar parâmetros em tempo de execução, ou seja, deve ser possível especificar como a carga de trabalho é alterada durante o tempo de execução. Analogamente, a responsabilidade **capacidade** permite que os componentes computacionais sejam providos em função de condições operacionais em tempo de execução. O **monitor** atua como um mecanismo sensor e de aquisição de dados temporal, com a responsabilidade de deixar disponível para demanda e capacidade as informações coletadas (por exemplo, utilização média, tempo de residência, taxa de perca de *deadline*, e outras informações dependentes de aplicação). **Effector** representa o mecanismo de atuação, servindo como a interface entre o MEDC e o sistema computacional, afetando como as alterações serão propagadas pelo sistema. É através dele que os comandos de alocação/desalocação ocorrem, que políticas de escalonamento, padrões de falhas etc. ocorrem.

O fluxo operacional consiste em periodicamente adquirir dados do sistema computacional, manter uma estrutura para armazenar e disponibilizar esses dados; a cada instante de tempo uma interrupção é gerada, avisando os moduladores de capacidade e demanda que a nova informação

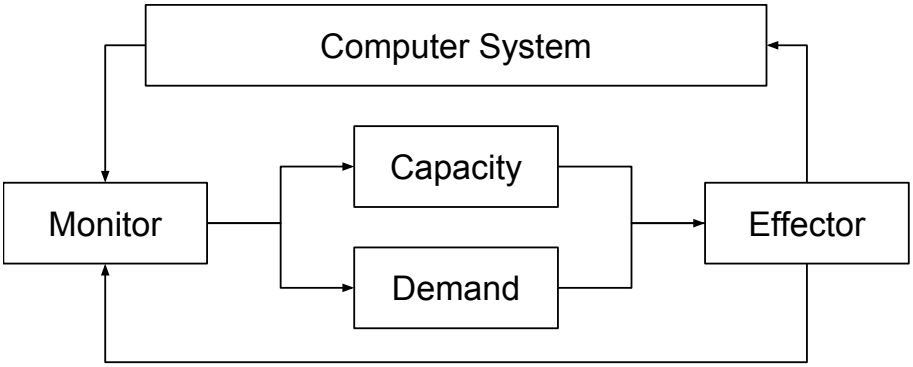


Figura 26 – Modelo referencial MEDC.

está disponível; ações são realizadas por estes últimos e direcionadas ao *effector*. O *effector* decide em adicionar as perturbações desejadas nas informações por ele recebidas. O monitor pode obter informações localizadas no *effector*, por exemplo, no caso de máquinas virtuais em estado de *boot* — essa informação pode ser mantida nele. Dessa forma, cada responsabilidade tem uma ação associada a ela que deve ser encadeada de forma que a cada evento do monitor aconteça a execução de todo fluxo, porém cada execução do fluxo é independente, ou seja, no instante de execução daquele fluxo, o estado atual do sistema é observado e as ações são tomadas.

Define-se as operações: `Monitor.get`, `Demand.update`, `Capacity.update` e `Effector.set`. `Monitor.get` amostra o estado atual do sistema; `Demand.update` e `Capacity.update` determinam quanta carga de trabalho e unidades computacionais, respectivamente, devem ser adicionadas ou subtraídas do sistema; `Effector.set` envia os eventos para o sistema computacional.

4.2 OnlineBroker: Uma extensão para o CloudSim

Como exemplo de implementação esta seção descreve como o simulador CloudSim (CALHEIROS *et al.*, 2011) foi estendido para a implementação de avaliação de desempenho não estacionária. CloudSim é um conjunto de classes implementadas em Java que permite a criação de simulações focadas em Computação em Nuvem. Trata-se de um motor de simulação orientado a eventos (*Discrete-event simulation*) tradicional adicionado de muitas entidades que representam elementos reais dos ambientes de nuvem comerciais. Sua concepção tem como prerrogativa básica a flexibilidade no sentido de ser possível expressar diferentes configurações e características de provedores distintos. As características inovadoras incluem a possibilidade de simular ambiente de grande escala, exigindo baixa potencia computacional; ambiente de simulação auto-contido; suporte a descrição de topologia de rede; suporte a simulação intercloud; representação de virtualização de recursos; e políticas de alocação de tempo compartilhado ou reserva de recursos.

O núcleo do simulador trabalha com uma lista de eventos futuros. Cada evento (*SimEvent*) possui uma entidade (*SimEntity*) da simulação associada a ele. Durante a simulação, os

eventos são removidos da lista e invocam funções que são implementadas nas entidades para o respectivo processamento. Para implementar sua própria simulação o usuário pode utilizar de uma série de exemplos já disponíveis com o CloudSim. O ponto de entrada para a simulação é o `DataCenterBroker`, que provê uma interface (*Application Programming Interface* (API)) orientada a execução em lote. Isso significa que o desenvolvimento de uma simulação consta da instanciação de um objeto `Datacenter`, em que deve ser descrito os atributos do provedor do serviço, tais como, especificação de máquinas físicas, tecnologia de virtualização, política de compartilhamento de recursos etc.; e da instanciação de um objeto `DataCenterBroker`, que representa o usuário na requisição de recursos computacionais (denominado `Vm`) ao *datacenter* e também na geração da carga de trabalho (cada unidade denominada `Cloudlet`).

No início da simulação o *broker* deve receber todas as `Vms` e todas as `Cloudlets` e as enviar para o *datacenter*. A simulação consiste em consumir toda essa carga até sua exaustão. Ao final um relatório contendo a descrição da execução é gerado, o que inclui um conjunto de métricas disponíveis. O Código 1 apresenta em pseudo-código de um simulador básico que utiliza o CloudSim.

Código-fonte 1: Código simplificado de uma simulação mínima com o `DatacenterBroker`

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

Na página principal do projeto[1] há uma série de extensões organizadas em outro projeto com o nome CloudSimEx, as quais adicionam funcionalidades extras para aplicações específicas como sistemas baseados em web, MapReduce, aplicações que são sensíveis a latência de

[1] CloudSim: <https://github.com/Cloudslab>

rede, políticas de escalonamento etc. Outras extensões existem também, como é o caso do DynamicCloudSim (BUX; LESER, 2015), que adiciona características de desempenho variáveis dos elementos computacionais disponíveis. Isso reforça a característica extensível do CloudSim.

Não há meios de alterar a quantidade de `Cloudlets` ou `Vms` disponíveis durante a execução da simulação de modo a especificar como a demanda e a capacidade mudam durante o tempo. Para que a análise do comportamento dinâmico seja feita, a funcionalidade de modular tais responsabilidades é essencial, o que implicaria mudar o paradigma de lote para *online*. Portanto, é requisito que seja possível a geração de carga de trabalho e o redimensionamento e também o monitoramento de desempenho, todos eles durante a execução da simulação. A proposta para a avaliação não estacionária baseada em simulação de eventos discretos implementada no CloudSim, tem a intenção de adicionar um *broker* diferente que, em vez de trabalhar com os eventos em modo de lote, trabalha em modo *online*. O diagrama de classes da extensão é mostrado no Fig. 27. A extensão é feita a partir da classe `SimEntity`. Cada responsabilidade do modelo MEDC foi traduzida para uma classe específica. Essas classes são abstratas e é trabalho do desenvolvedor da simulação instanciá-las conforme o domínio da aplicação.

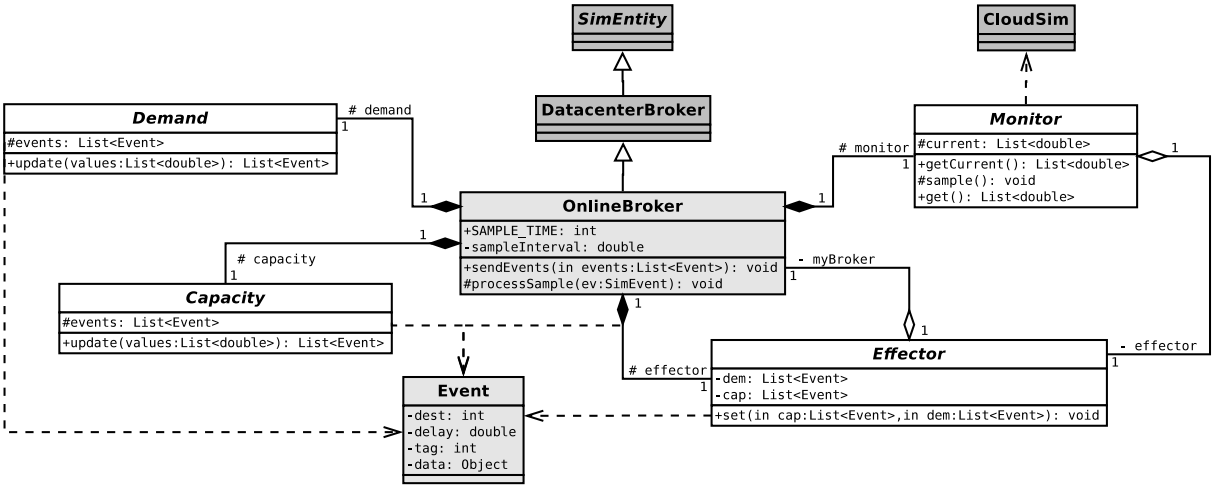


Figura 27 – Diagrama de classes da extensão OnlineBroker.

A classe original do CloudSim `DatacenterBroker` implementa o ciclo de atividades ilustrado na Fig. 28. Há a convenção de que deve existir um `Datacenter` registrado. O `DatacenterBroker` requisita quais `Datacenters` estão disponíveis e na sequência requisita suas características. De posse dos `Datacenters` disponíveis, cria todas as `Vms` e após a criação envia todas as `Cloudlets`. Ao término de cada `Cloudlet` é gerado um evento de término. Quando todas as `Cloudlets` tiverem terminado, as `Vms` são destruídas. Para que esse encadeamento seja feito com sucesso foi necessário um código complexo e altamente acoplado. A estratégia para a implementação da extensão, foi criar um novo *broker* que mantém a maioria das atividades do original, porém altera-se a forma com que há o gerenciamento de eventos relativos a `Vm` e `Cloudlet`. O resultado foi um código mais coeso.

A classe `OnlineBroker` contém o ponto de entrada em que o motor de simulação invoca

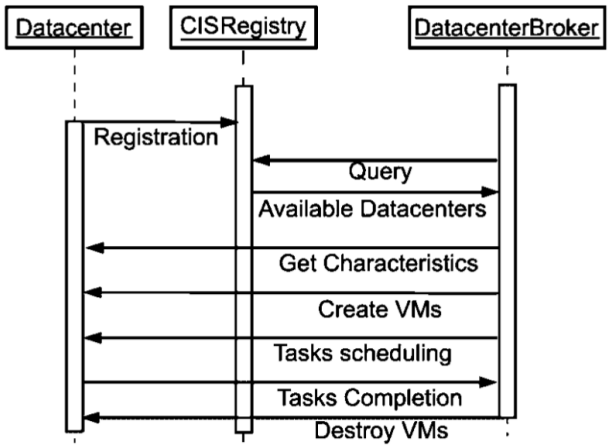


Figura 28 – Ciclo de vida de uma simulação implementado pelo `DatacenterBroker`.

Fonte: Calheiros *et al.* (2011).

toda vez que um evento destinado a uma de suas instâncias é escolhido da lista de eventos futuros. Dada essa característica, o desenvolvimento se deu ao adicionar um evento que ocorre periodicamente cuja finalidade é realizar o fluxo de processamento MEDC, a esse evento deu-se o nome de `SAMPLE`. O Código 2 apresenta essa função. A cada amostragem é coletada as informações de desempenho, feita a atualização dessas informações para as responsabilidades de capacidade e demanda, seguida da efetivação dos eventos produzidos. Os valores atuais são descartados, e o próximo evento de amostragem é agendado.

A estrutura para armazenar as informações coletadas a cada amostragem é feita pela estrutura de dados `Map<Key,Value>`. `Key` é um valor inteiro que serve como chave identificadora para o tipo da informação. Por exemplo, seja os eventos que marcam o retorno de `Cloudlets` ou a confirmação de criação de `Vm`. Durante um tempo de amostragem e outro podem ocorrer vários eventos do mesmo tipo. `Value` corresponde à lista de eventos que aconteceu naquele intervalo de tempo. Portanto, a cada período de amostragem haverá um objeto do tipo `Map` que possuirá potenciais várias `Keys` e para cada `Key` uma lista de valores guardados no objeto `Value`.

Código-fonte 2: Função que encadeia a execução do fluxo MEDC.

```
1 function processSample(SimEvent ev) {
2   monitor.get();
3   effector.update(
4     capacity.update(monitor.getValues()),
5     demand.update(monitor.getValues())
6   );
7   monitor.clearValues();
8   send(this.brokerId(), nextSample, SAMPLE);
9 }
```

Para fins de ilustração, o Código 3 exemplifica como o comportamento do `DatacenterBroker`

pode ser reproduzido a partir do `OnlineBroker`. O algoritmo apresenta que no instante de tempo $t = 0$ as `Vms` são criadas e quando elas estiverem prontas as `Cloudlets` são enviadas. O método `Effector.set` apenas passa adiante os eventos. A próxima seção 4.3 apresenta a implementação que foi utilizada para produzir resultados de parte das publicações desta tese.

Código-fonte 3: Pseudo-código para a reimplementação do `DatacenterBroker` a partir do `OnlineBroker`.

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

4.3 Exemplo de uso: DynamicController

Como contextualização e motivação desta implementação, cita-se que Nobile (2013) propõe uma abordagem de implementação de um mecanismo adaptativo de alocação de recursos em arquitetura de computação em nuvem, a partir das técnicas de análise e projeto inspiradas em conceitos de controle clássico. O objetivo do mecanismo desenvolvido é regular a *taxa de utilização* dos recursos do *datacenter*, definida como a fração da capacidade do sistema que, em média, é utilizada. A taxa de utilização pode ser aumentada ou reduzida pelo respectivo aumento ou diminuição do desempenho do *datacenter*, proporcional ao número de VMs (recurso elástico). Assim, a estratégia de gerenciamento adaptativo de recursos consiste em ativar ou desativar máquinas virtuais a fim de manter a taxa de utilização regulada frente a perturbações na carga de trabalho.

A Figura 29 oferece uma representação em alto nível do sistema proposto. No diagrama, utilizando a notação convencional, os blocos representam as funções de transferência de cada um dos subsistemas, e as setas indicam seus respectivos sinais de entrada e saída. No diagrama o *datacenter* é representado pelo bloco G, sendo a entrada x a quantidade de máquinas virtuais desejadas, e a saída y a taxa de utilização resultante da carga de trabalho imposta ao sistema. O sinal d é uma perturbação imposta à carga causando o aumento ou diminuição na taxa de utilização final o. O bloco H é um sensor que mede a taxa de utilização e aplica uma média móvel exponencial, sendo m seu resultado. O sinal r é o valor da taxa de utilização desejada, sendo e o erro de $r-m$. O bloco C é o controlador que utiliza o erro e para gerenciar a quantidade de máquinas virtuais necessárias, x.

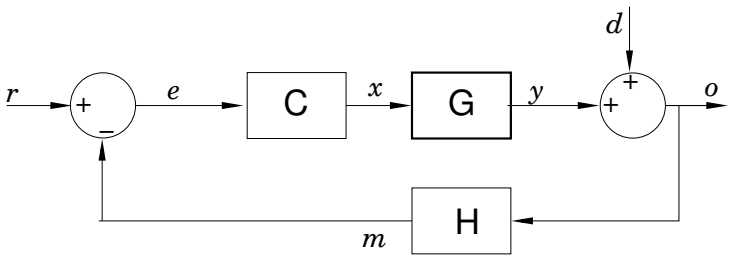


Figura 29 – Diagrama da arquitetura de controle de Nobile.

Fonte: Nobile (2013).

Nobile determina experimentalmente que a planta, i.e *datacenter*, representada pelo bloco G, tendo como entrada e saída os sinais indicados, pode ter seu comportamento dinâmico aproximado por um sistema linear invariante no tempo (LTI) de primeira ordem, logo, respondendo assintoticamente na forma exponencial a uma entrada na forma de degrau, mudança súbta no número de máquinas virtuais (VMs). O laço de realimentação (sinal m), eleva a ordem

do sistema, de modo que em malha-fechada, o sistema tem comportamento oscilatório. Os parâmetros do controlador foram ajustados para que a dinâmica seja a de um sistema estável, com oscilações subamortecidas.

A implementação de um simulador que satisfaça essa descrição segue conforme ilustrado no diagrama de clases da Fig. 30. O diagrama contém duas classes pertencentes ao CloudSim. As demais classes são do `OnlineBroker` e do `DynamicController`. Ou seja, a parte específica que atende os requisitos da aplicação de Nobile correspondem às classes cujo fundo são de cor branca. Basicamente, faz a criação das classes abstradas definidas no `OnlineBroker` e adicionar um gerenciador de Máquinas Virtuais. Nos próximos parágrafos serão descritos como as funções do modelo MEDC foram implementadas e discutido como foi feita a gerência de recursos computacionais. O intuito é destacar como seria uma implementação real e mostrar o esforço necessário. Apenas parte do código será comentado aqui. O projeto completo pode ser obtido a partir do Github[2].

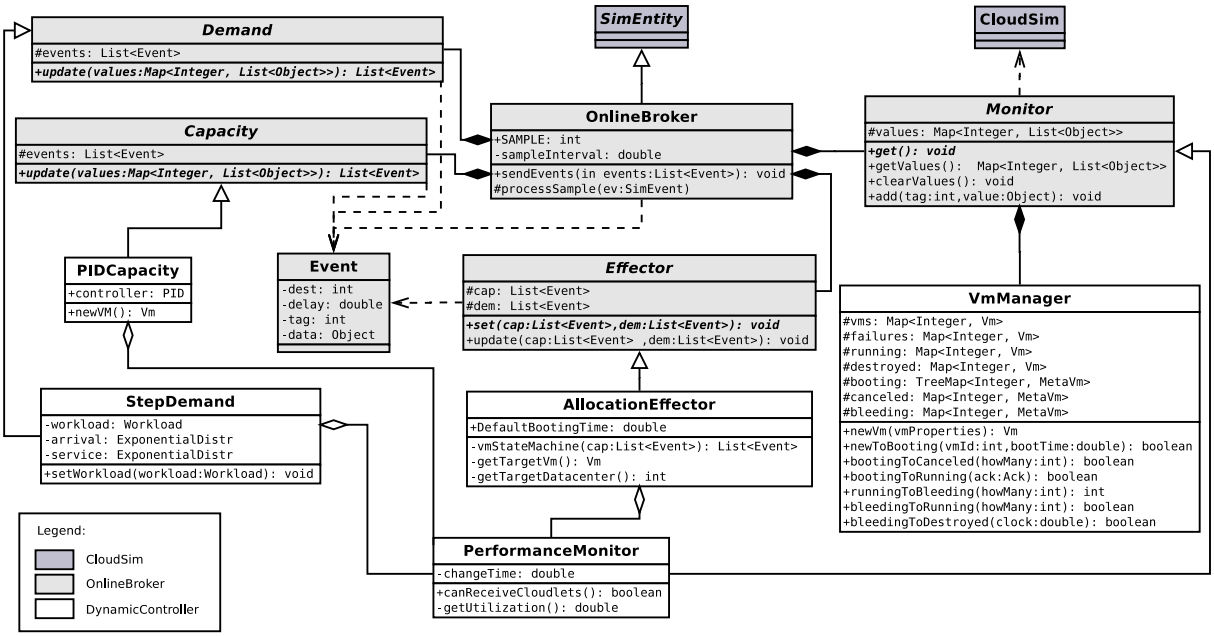


Figura 30 – Diagrama de classes do `DynamicController`.

O Código 4 apresenta a implementação da função `update` da classe responsável pela geração da demanda do `DynamicController`. Inicialmente, os eventos da iteração anterior são descartados e é verificado se a simulação já possui `Vms` instanciadas para receber carga de trabalho. Em determinado ponto durante a execução da simulação haverá um degrau, de forma que um novo patamar é especificado pelo intervalo entre as chegadas e o tempo de serviço das tarefas (`Cloudlet`). A cada intervalo de amostragem, todas as chegadas para aquele intervalo devem ser geradas. Por isso, há um laço de repetição que irá gerar todas as tarefas daquele intervalo e terminará com o instante de tempo de acontecimento da primeira tarefa do próximo intervalo já especificada. A função `newRequest` irá criar uma `Cloudlet` cujo tempo de processamento

[2] OnlineBroker: <https://github.com/ljr/OnlineBroker>

será `service.sample()`. Há dois objetos que instanciam a distribuição Exponencial, `service` e `arrival`. O método `sample()` retorna o próximo número da série. Durante a criação de um objeto dessa classe, `StepDemand`, são especificadas as sementes.

Uma nota deve ser feita em relação à carga de trabalho. Ela segue a mesma ideia do modelo M/M/1 da Teoria de Filas e corresponde à intensidade do tráfego. Por exemplo, se o intervalo entre as chegadas for 10 e o tempo de serviço for 70, a intensidade do tráfego será 7. Ou seja, a taxa de utilização do sistema será de 700%. A aplicação tem a funcionalidade de alterar a quantidade de servidores em tempo de execução. Especificamente, caso seja especificado que a taxa de utilização média dos servidores seja de 70%, serão instanciadas 10 VMs para atender ao tráfego gerado, correspondente a um sistema M/M/10. No entanto, devido à estocasticidade, mais máquinas podem ser adicionadas ou removidas até que o sistema entre em estado estacionário. A perturbação mencionada no parágrafo anterior, faz com que a intensidade do tráfego aumente abruptamente. Isso significa que o desenvolvedor da simulação deve passar os parâmetros λ e μ de antes e depois do degrau no instante de inicialização da simulação.

Código-fonte 4: Gerador de carga de trabalho exponencial para o `StepDemand.update`.

```
1 public List<Event> update(Map<Integer, List<Object>> values) {
2   cloudlets.clear();
3
4   if (!mon.canReceiveCloudlets()) {
5     return cloudlets;
6   }
7
8   if (CloudSim.clock() >= workload.getChangeTime()) {
9     setWorkload(workload.getLambdaAfter(), workload.getMuAfter());
10  }
11
12  happenAt = first;
13  do {
14    cloudlets.add(new Event(happenAt,
15            CloudSimTags.CLOUDLET_SUBMIT,
16            newRequest(service.sample()));
17      happenAt += arrival.sample();
18    } while (happenAt < monitor.SAMPLE_INTERVAL);
19    first = happenAt - monitor.SAMPLE_INTERVAL;
20
21  return cloudlets;
22 }
```

O controlador (`PIDController`) foi implementado conforme esboçado no Código 5. Os eventos da iteração anterior são descartados. É calculado quanto a utilização atual está diferente daquela especificada (`setPoint`). Caso seja o início da simulação uma quantidade

especificada de VMs será iniciada. Se a realização de requisições estiver em andamento, o controlador calculará quantas VMs devem estar em operação, e a diferença será repassada para provisionamento ou desprovisionamento. Novos objetos da classe `Vm` são criados para esse fim através do método `newVm`. Caso seja necessário remover máquinas, apenas a quantidade desejada é passada a diante. O método `processAck` altera o estado de uma `Vm` de *booting* para *running*, seu segundo parâmetro especifica essa ação pois o etiqueta `VM_CREATE_ACK` significa que a solicitação de criação de uma máquina virtual foi atendida. A função `controller` implementa as operações a serem feitas pelo controlador PID, os valores para os parâmetros `kp`, `ki` e `kp` são passados pelo desenvolvedor da simulação e fogem do escopo desta tese. Para maiores detalhes, esse assunto é abordado detalhadamente por Nobile (2013).

Código-fonte 5: Controlador de Máquinas Virtuais para o `PIDCapacity.update`.

```
1 public List<Event> update(Map<Integer, List<Object>> values) {
2   events.clear();
3
4   double error = setPoint - (double) values.get(Tags.UTILIZATION).get
     (0);
5   long howManyVms;
6
7   if (CloudSim.clock() == 0) {
8     howManyVms = vmsAtStart;
9   } else if (mon.canReceiveCloudlets()) {
10    howManyVms = controller(error) - mon.vmsInSystem();
11  } else {
12    howManyVms = 0;
13  }
14
15  if (howManyVms > 0) {
16    for (int i = 0; i < howManyVms; i++) {
17      events.add(new Event(/*delay = */ 0, CloudSimTags.VM_CREATE_ACK
     ,
18          newVm()));
19    }
20  } else if (howManyVms < 0) {
21    events.add(new Event(/*delay = */ 0, Tags.DESTROY, -howManyVms));
22  }
23
24  processAck(values, CloudSimTags.VM_CREATE_ACK);
25
26  return events;
27 }
28
29 private long controller(double error) {
30   double diff = kd * (error - lastError);
31   double prop = kp * error;
```

```
32   integral += ki * error;
33   lastError = error;
34   return Math.round(Math.floor(prop + integral + diff));
35 }
```

O método set do AllocationEffector está descrito no Código 6. Para cada VM da lista de capacidades (cap) é escolhido um *datacenter* através do método getTargetDc e enviada o evento para o *broker*, neste caso trata-se de um algoritmo simples que implementa a política *round-robin* na lista de *datacenter*s disponíveis. Para cada Cloudlet na lista de demanda (dem) uma Vm de destino é escolhida; a Cloudlet é enviada em sequência ao *broker*. A Vm é escolhida pelo método getTargetVM que implementa a política *round-robin* na lista de VMs disponíveis.

Código-fonte 6: Método set do AllocationEffector.

```
1 public void set(List<Event> cap, List<Event> dem) {
2   for (Event e : vmStateMachine(cap)) {
3     e.setDest(getTargetDc());
4     mybroker.sendEvent(e);
5   }
6
7   for (Event e : dem) {
8     Cloudlet cloudlet = (Cloudlet) e.getData();
9     cloudlet.setVmId(getTargetVm().getId());
10    e.setDest(getTargetVm().getHost().getDatacenter().getId());
11    mybroker.sendEvent(e);
12  }
13 }
```

Um ponto importante da implementação do AllocationEffector é a presença da máquina de estados que representa o ciclo de vida das Máquinas Virtuais durante a simulação. A Fig.31 ilustra esses estados e as possíveis transições entre eles. Assim que uma nova VM é criada, há a tentativa de alocá-la em um *datacenter* disponível. Caso não seja possível ela vai para o estado *created*. Após ser criada, a VM existe no sistema, mas não contribui para a produção de resultados, pois ela é abstraída como em estado de inicialização (*booting*). Após o *boot*, há o estado de execução (*running*). Caso haja a necessidade de desligar alguma VM, será dado preferência àquelas em estado de *booting*, portanto, há a transição para *canceled*. Enquanto em estado de execução, a VM recebe requisições, processa e envia o resultados. Caso esteja processando e uma nova requisição chegue, esta é enfileirada para posterior processamento. Caso seja necessário desligar uma VM em execução, ela é colocada em estado *bleeding*, isto é, há requisições pendentes na fila de execução, e a VM será desligada após a exaustão dessa lista. Caso seja necessário aumentar a quantidade de VMs em execução, será dado preferência às VMs em estado de *bleeding*. Após executar todas as Cloudlets de sua lista, a VM é desprovisionada, vai para o estado *destroyed*.

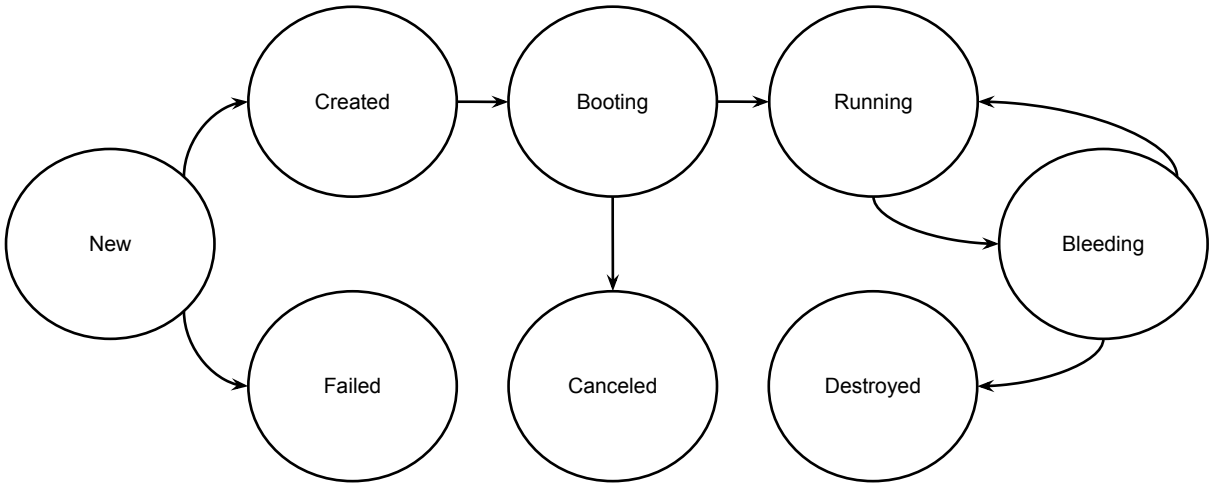


Figura 31 – Máquina de estados que representa o ciclo de vida das Máquinas Virtuais.

Para a implementação do código que traduz essa máquina de estados, considere a Fig. 32. A linha horizontal preta representa a quantidade de VMs criadas no sistema, independente de seu estado. Para a operação de adição de novas VMs (Fig. 31a), deve-se considerar a quantidade de máquinas em estado de *bleeding* (`inBleed`), a quantidade de máquina a serem iniciadas (`howMany`) e a diferença (`left`) caso a quantidade em `inBleed` seja insuficiente. Se não houver máquinas em `inBleed`, basta iniciar `howMany` `Vms`. Se `howMany` for menor ou igual a `inBleed`, marcar a respectiva quantidade para o estado de *running*. Caso contrário, criar a diferença (`left`) e marcar todas VMs de *bleeding* para *running*.

O seguinte quadro faz uma análise dos valores possíveis de acontecerem para o caso da adição de novas VMs e apresenta um pseudo-algoritmo para quais operações deve ser realizadas.

```
left = howMany - inBleed

howMany      inBleed          left        toCreate    toRun
positivo     zero             howMany     howMany     zero
positivo     positivo
  howMany   <   inBleed       negativo    zero        howMany
  howMany   =   inBleed       zero        zero        howMany
  howMany   >   inBleed       positivo    left        inBleed

Portanto:
if inBleed == o
  toCreate = howMany
else if howMany > inBoot
  toCreate = left
  toRun = inBleed
```

```
else
  toRun = howMany
```

Analogamente, é o caso da remoção de VMs (Fig. 31a). Deve-se comparar as variáveis `howMany` e `inBoot`, bem como sua diferença (`left`). Caso não haja VMs em estado de *boot*, deve-se marcar `howMany` VMs em estado de *bleeding*. Caso `howMany` seja menor ou igual à quantidade de VMs `inBoot`, `howMany` deverão ser abortadas do estado de *booting*. Caso contrário, cancelar todos os processos de *boot* e marcar a diferença (`left`) para o estado de *bleeding*. Para o caso da remoção de VMs este quadro faz a análise de quais ações devem ser tomadas a partir da possibilidade de todas as combinações, o pseudo-algoritmo é apresentado.

```
left = howMany - inBoot

howMany     inBoot        left        toBleed   toCancel
positivo    zero          howMany     howMany   zero
positivo    positivo
  howMany  <  inBoot      negativo    zero      howMany
  howMany  =  inBoot      zero        zero      howMany
  howMany  >  inBoot      positivo    left      inBoot

Portanto:
if inBoot == 0
  toBleed = howMany
  toCancel = 0
else if howMany > inBoot
  toBleed = left
  toCancel = inBoot
else
  toBleed = 0
  toCancel = howMany
```

Uma vez analisado como devem ser as ações para adição e remoção de VMs, o Código 7 apresenta o pseudo-código da implementação. Caso haja eventos na lista `cap`, é verificado se devem ser retiradas ou adicionadas `Vms` ao sistema. Para cada caso, as ações são tomadas para aquele fim. Ao final as VMs que já inicializaram (`bootCompleted`) são adicionadas a eventos que as tornem operacionais.

Código-fonte 7: Implementação da Máquina de Estados do ciclo de vida das VMs.

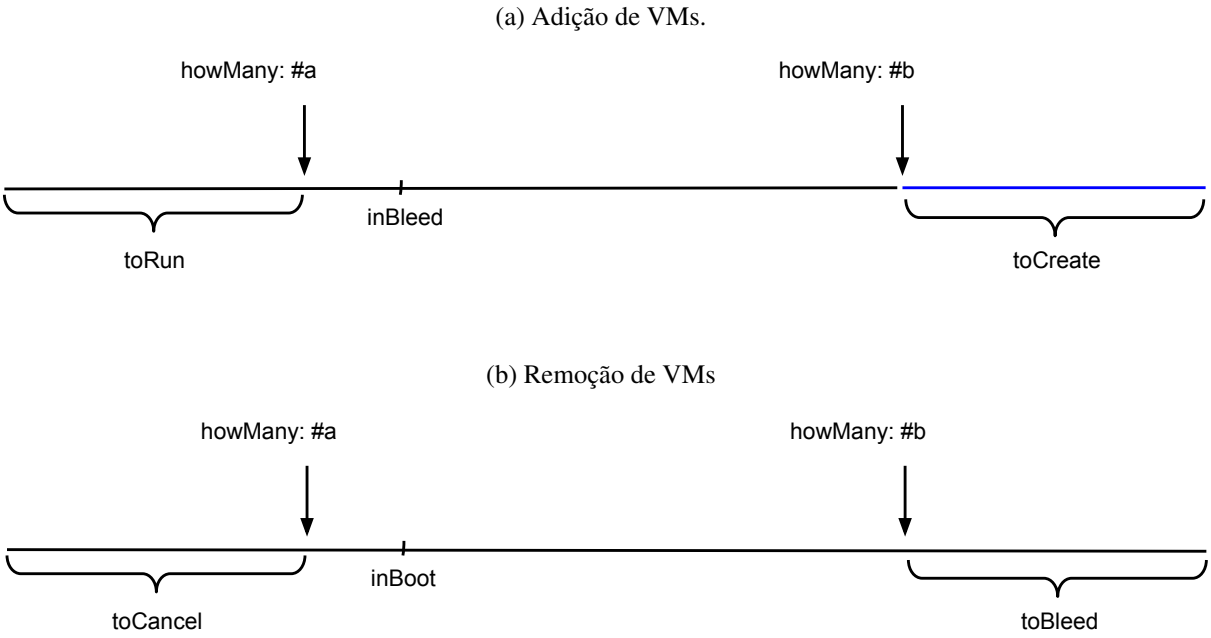


Figura 32 – Análise das operações de Adição e Remoção de VMs da aplicação.

```
1 private List<Event> vmStateMachine(List<Event> cap) {
2   if (!cap.isEmpty()) {
3     if (cap.get(0).getTag() == Tags.DESTROY) {
4         removeVms(howMany = (long) cap.get(0).getData())
5     } else {
6       addVms(cap);
7     }
8   }
9
10  cap.clear();
11  for (Vm vm : mon.getVmManager().bootCompleted(CloudSim.clock())) {
12    cap.add(new Event(/*delay=*/ 0, CloudSimTags.VM_CREATE_ACK, vm));
13  }
14
15  return cap;
16 }
```

O método `get` da responsabilidade monitor é implementado conforme Código 8. A taxa de utilização das VMs é obtida através de sua média simples, o qual é passado para uma média móvel exponencial para suavização. O método `save` adiciona as informações para identificação do sistema em um arquivo *Comma Separated Values* (CSV).

Código-fonte 8: Implementação do método `Monitor.ger`.

```
1 public void get() {
2     utilization = getUtilization();
```

```
3   add(Tags.UTILIZATION, utilization);
4   save(CloudSim.clock(), vmManager.getRunning(), utilization);
5 }
6
7 private double getUtilization() {
8   double u = 0;
9
10  for (Vm vm : vmManager.getRunning().values()) {
11    u += vm.getCloudletScheduler().getTotalUtilizationOfCpu(CloudSim.
      clock());
12  }
13
14  return ema(u / vmManager.getRunning().size());
15 }
```

O Código 9 apresenta um exemplo de instanciação da classe `OnlineBroker`. As instâncias das classes abstratas são passadas para o método construtor. Deve-se observar que para esta aplicação são necessários alguns parâmetros, como a especificação da intensidade da carga de trabalho, antes e depois do degrau; o instante de tempo em que haverá o degrau; a semente utilizada no gerador de números pseudo-aleatórios; quantidade de VMs no início da execução; e parâmetros para o controlador PID.

Código-fonte 9: Exemplo de simulação.

```
1   double lambdaBefore = 20;
2   double lambdaAfter = 10;
3   long muBefore = 14;
4   long muAfter = 7;
5   double changeTime = 400;
6   long seed = 12345;
7   int vmsAtStart = 7;
8   double setPoint = .7;
9   double kp = 18.5446;
10  double ki = -0.085535;
11  double kd = 0;
12
13  OnlineBroker ob = new OnlineBroker("Broker", 10,
14      new PerformanceMonitor(),
15      new AllocationEffector(),
16      new StepDemand(seed, StepDemand.newWorkload(lambdaBefore,
17          lambdaAfter, muBefore, muAfter, changeTime)),
18      new PICapacity(vmsAtStart, setPoint, kp, ki, kd)
19  );
20
21  CloudSim.startSimulation();
```

4.3.1 Estudos de caso utilizando o `DynamicController`

Além dos artigos que propõem o modelo MEDC e sua implementação com o CloudSim (PEREIRA JR. *et al.*, 2015; PEREIRA *et al.*, 2015), dois artigos foram publicados fazendo uso da extensão. Nesta seção estes últimos serão comentados, destacando-se a implementação como ferramenta para a produção de resultados que evidenciam a utilização da abordagem dinâmica para modelagem como útil no desenvolvimento de um mecanismo adaptativo de gerenciamento de recursos de computação em nuvem (MAMANI *et al.*, 2015); e também sobre o tratamento do ruído no processo de identificação e modelagem de sobrepassagem de VMs em ambiente de computação em nuvem (LUZ *et al.*, 2016).

A hipótese de linearidade dentro da faixa de operação foi verificado ao excursionar o sinal de entrada em torno da região de operação e verificar se a propriedade de superposição no sinal de saída se mantinha. Por meio de replicação dos experimentos, o ruído estocástico do sinal de saída U foi removido. Um modelo ARX de primeira ordem que relacionou a taxa de utilização média (U) das máquinas virtuais presentes no sistema em função do número de máquinas virtuais operantes (N). Os parâmetros encontrados permitiram extrair os parâmetros $a_1 = -0.57961$ e $b_1 = 0.016263$, compondo o modelo

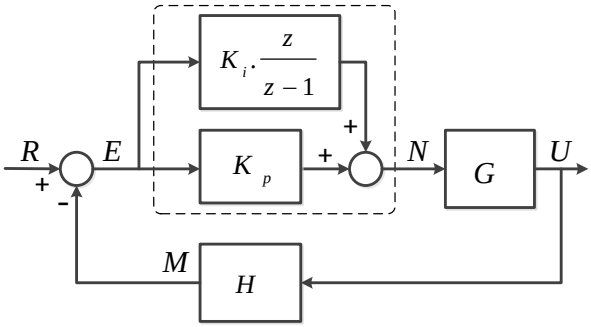
$$u(k+1) = 0.57961u(k) - 0.016263n(k), \tag{4.1}$$

em que $u(k)$ é a taxa de utilização no instante de tempo k e $n(k)$ é o número de máquinas virtuais no instante de tempo k. A Função de Transferência pode ser descrita como

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

O projeto do mecanismo de controle é baseado em um laço retroalimentado clássico, ilustrado na Fig. 32a. O bloco G corresponde à simulação, no entanto, deve-se observar que apenas duas variáveis são utilizadas. O bloco H corresponde à média móvel exponencial e foi implementada pela classe `PerformanceMonitor`. O laço de retroalimentação corresponde à classe `PIDCapacity`, a porção derivativa é configurada como zero, portanto não representada na figura. A demanda atua como uma perturbação a este sistema. O `Effector` é parte integrada de G e pertencente à simulação. O controlador foi configurado para que o desempenho tivesse o seguinte objetivo: se a referência for alterada de 70% para 80%, o controlador deve atuar de forma que a sobrepassagem máxima seja 10% e o tempo de assentamento menor que 300 ms. O parâmetros K_p e K_i foram calculados utilizando-se o método do lugar das raízes (*root locus method*). A Fig. 32b apresenta o desempenho do controlador após uma perturbação ter ocorrido. A modulação de recursos permitiu a identificação do sistema e a implementação do controlador. A modulação da demanda permitiu que uma perturbação fosse realizada. O monitoramento periódico foi essencial para que essa estratégia de controle fosse empregada.

(a) Malha fechada para o controlador de recursos.



(b) Comportamento da taxa de utilização após sofrer um degrau no número de requisições.

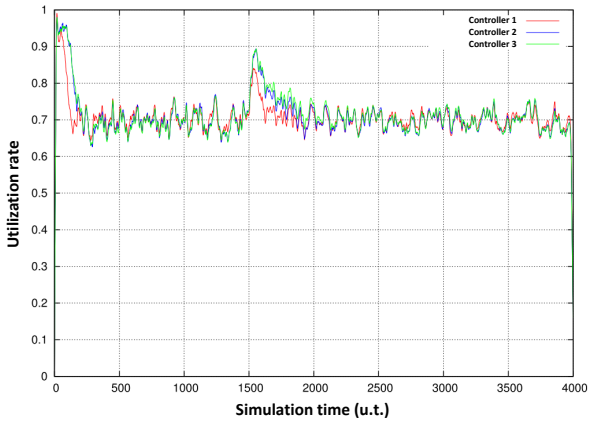


Figura 33 – Malha de controle para o gerenciador de recursos para computação em núvem e desempenho após uma perturbação na carga de trabalho.

4.3.2 Estudo de caso de sobrecargas transientes

Em um segundo estudo de caso (LUZ *et al.*, 2016) , a arquitetura de Nobile, esquematizada na Figura 29, é tomada como uma caixa preta. Objetiva-se obter um modelo que informe como o número de máquinas virtuais (VMs), instanciadas e desativadas pelo gerenciador adaptativo de recursos elásticos, varia em função de um aumento da carga de trabalho. Evidentemente há uma relação proporcional entre as duas variáveis (pela ação do controlador), mas este estudo demonstra que essa relação é dinâmica e que existem componentes transientes. Isso significa que, se o sistema opera inicialmente com um número médio de máquinas, uma perturbação na carga de trabalho pode produzir um pico transitório no número de máquinas virtuais, que pode perdurar até que as novas máquinas absorvam a demanda e levem a saída a um novo patamar estacionário.

Considerando que em sistemas de computação em nuvem não há cobrança apenas de máquinas em execução, mas também das máquinas em processo de inicialização(AMAZON, 2016), pode-se obter uma outra visão do sistema de controle estudado até o momento. A Fig. 34 ilustra esse sistema. A saída do modelo é tomada como o número de máquinas virtuais (VMs) alocadas, independente do seu estado. A variável de entrada do sistema escolhida é a carga de trabalho. Sendo N o número de máquinas virtuais e W a carga de trabalho dada por $W = \Delta\mu/\Delta\lambda$, em que $\Delta\mu$ é o tempo de processamento médio (em unidades de tempo, *ut*) e $\Delta\lambda$ o intervalo médio entre chegada das requisições (em unidades de tempo).

Para determinar como o controlador responde às variações da carga de trabalho e encontrar uma função que defina a transformação de W em N, o sistema (modelo simulado com o `OnlineBroker`) foi submetido a uma carga de trabalho em formato de degrau. O tempo de processamento foi fixado em $\mu = 70$ *ut*, e o intervalo entre chegadas foi inicialmente estabelecido em $\lambda = 10$ *ut*, produzindo uma carga de trabalho $W = 7$. No instante de tempo $k = 1000$, o

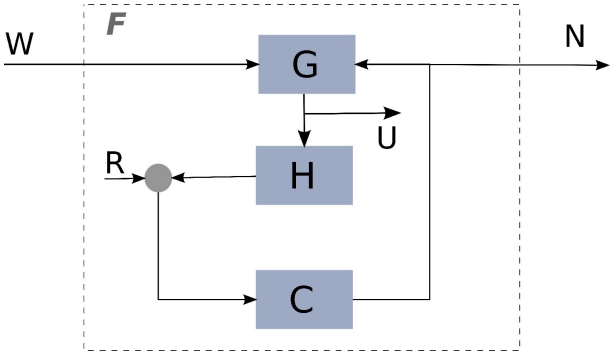


Figura 34 – Sistema que relaciona carga de trabalho e número de máquinas virtuais em um sistema de computação em nuvem com controle de recursos adaptativos.

intervalo entre chegadas foi reduzido para $\lambda = 5$ *ut*, o que aumentou bruscamente a carga de trabalho para $W = 14$. O experimento foi conduzido até o instante $k = 2000$, sendo efetuadas 300 replicações. A Figura 35 mostra a média entre as replicações de W e N para cada instante de tempo.

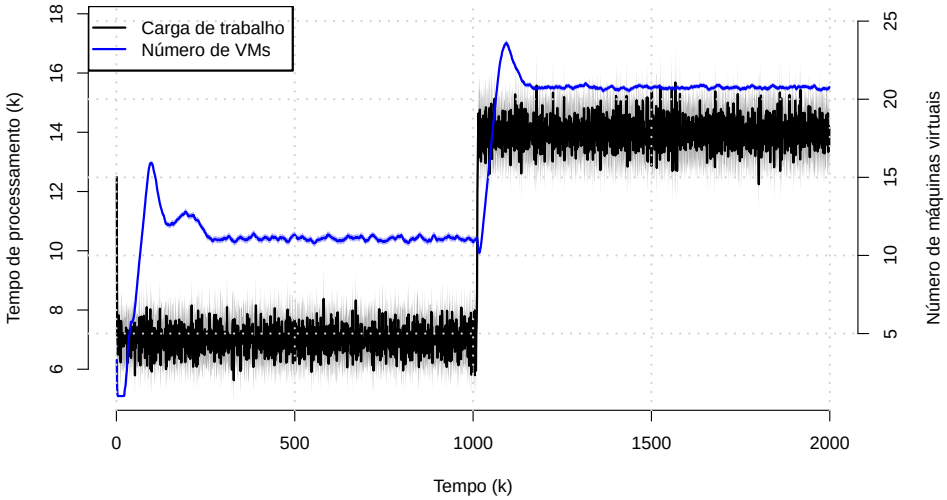


Figura 35 – Carga de trabalho (W) e número de VMs previstas pelo controlador.

Nota-se que ocorrem sobrepassagem no número de VMs iniciadas tanto nos instantes de tempo entre 0 e 240 *ut* quanto entre 1000 e 1200 *ut*.

Pelo comportamento da saída ao degrau demonstrado no gráfico da Figura 35, tendo poucas oscilações e baixo ruído proveniente da estocasticidade do sinal de entrada do sistema, o comportamento pode ser aproximado ao de um sistema de segunda ordem. Utilizando a técnica de identificação de sistemas proposta por (PAREKH; GANDHI; HELLERSTEIN, 2002) é possível definir um modelo ARX de segunda ordem que represente o comportamento do sistema, demonstrado pela equação (4.3), em que $\hat{y}(k+1)$ é a saída prevista no instante $k+1$ proveniente dos valores de entrada u e saída y nos instantes k e $k-1$. Os parâmetros a_1 e a_2 indicam a

influência das saídas anteriores em $\hat{y}$ e b_1 e b_2 a influência das entradas. Considerando que após a modelagem do sistema, para prever $\hat{y}$ haverá somente a variável de entrada W e as saídas anteriormente previstas com base na entrada, é necessário normalizar os valores de W e N de forma que o estado estacionário inicial fique em torno de 0.

$$\hat{y}(k+1) = -a_1\hat{y}(k) - a_2\hat{y}(k-1) + b_1u(k) + b_2u(k-1) \tag{4.3}$$

Denomina-se $F(z)$ a Função de Transferência

$$F(z) = \frac{b_1z^{-1} + b_2z^{-2}}{1 + a_1z^{-1} + a_2z^{-2}}. \tag{4.4}$$

Ao utilizar o método do mínimos quadrados os parâmetros para o modelo foram estimados conforme Eq. 4.5) e verificado o seu ajuste, apresentado na Figura 36. Nota-se visualmente o bom ajuste do modelo nos períodos estacionários e a aproximação da resposta ao degrau no modelo em relação a saída do controlador. O valor na Tabela 7 de R^2 (variabilidade) próximo de 1, do baixo valor do *Root-mean-squared Error* — RMSE e a forte correlação positiva da entrada em relação a saída do modelo também corroboram com o bom ajuste.

$$F(z) = \frac{-0.01024z^{-1} + 0.01312z^{-2}}{1 - 1.96z^{-1} + 0.9617z^{-2}} \tag{4.5}$$

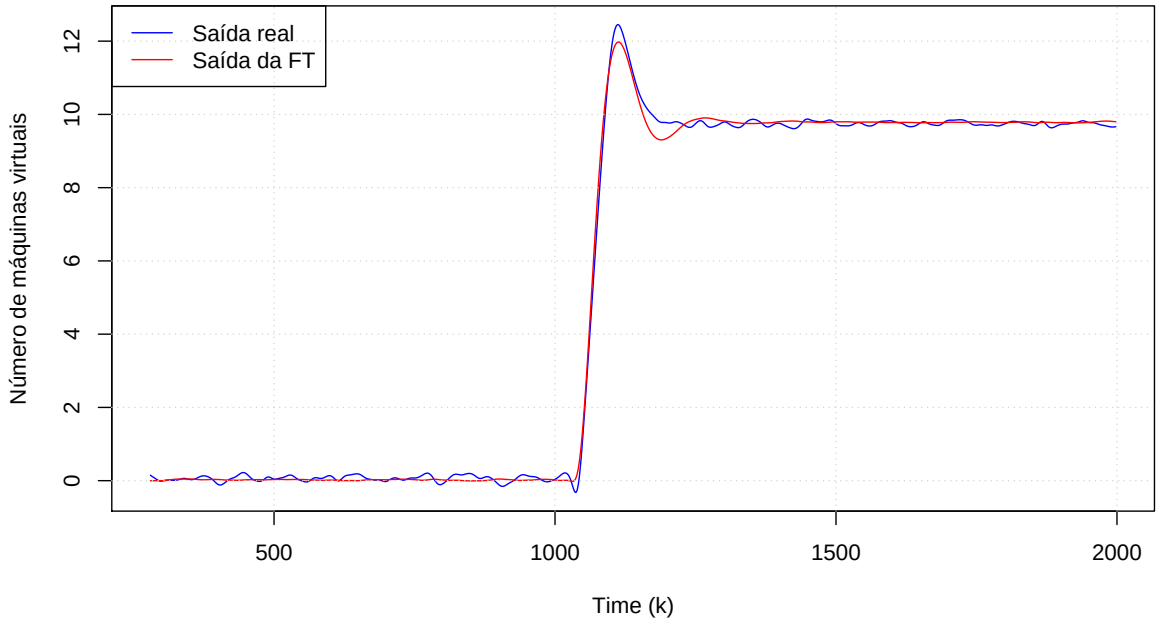


Figura 36 – Saída prevista pelo modelo e saída gerada pelo controlador.

Tabela 7 – Valores de R^2, *RMSE* e correlação entre entrada e a saída do modelo estimado

Teste	**Valor calculado**
R^2	0.99900
RMSE	0.15429
Correlação	0.97332

CAPÍTULO

5

MÉTRICAS DE DESEMPENHO DINÂMICO

Este capítulo retoma o exemplo estudado na Seção 4.3.1, focalizando o uso de métricas de desempenho transiente e ilustrando como a abordagem de modelagem dinâmica pode ser realizada para compreender e avaliar os resultados empíricos obtidos por simulação ou *benchmarking*.

5.1 Caso de estudo

A computação em nuvem tem se estabelecido como um modelo de implantação de recursos computacionais em que hardware e software são comercializados como serviços acessíveis por meio da rede. Um aspecto chave favorecendo a tendência industrial a esse paradigma é a elasticidade de recursos, ou a habilidade de dimensionar a capacidade do sistema em função da demanda conforme necessário para atender aos objetivos de eficiência. Enquanto capacidades adaptativas são em geral características desejadas em qualquer sistema computacional complexo, para a computação em nuvem, o auto gerenciamento de recursos é não apenas bem-vindo, mas inerente ao próprio modelo de negócios.

Em termos gerais, qualquer capacidade de auto dimensionamento baseia-se em identificar o excesso ou insuficiência de recursos, tomar contrarreações apropriadas. A estratégia essencial consiste em monitorar a carga de trabalho que chega (*feedforward*) ou a performance apresentada (*feedback*) e então alocar ou desalocar recursos de acordo em tempo real.

Para esse fim, uma abordagem trivial é projetar um mecanismo adaptativo que redimensione os recursos do sistema proporcionalmente à demanda presente. A razão pela qual essa tática pode não soar pouco engenhosa à primeira vista é porque tem funcionado razoavelmente bem em uma larga gama de aplicações. De fato, a técnica é efetiva quando os mecanismos internos do sistema não causam atraso significativo na ação corretiva, tal que para efeitos práticos, o impacto da entrada de controle aparece na saída imediatamente com respeito à resolução de tempo do

processo. Quando essa propriedade se aplica a simples política de contrarreação proporcional é efetiva, e o fato de que tem sido tradicionalmente suficiente em muitas situações tem propiciado que desenvolvedores assumam sem considerar alternativas.

Uma questão emerge quando a abordagem pragmática não se verifica. Diferente do que ocorre no nível funcional, a implementação de hardware e de software pode exibir atrasos. Conforme *datacenters* crescem em dimensão e sofisticação o resultado da combinação dos diversos subsistemas pode resultar em dinâmica significante.

Este capítulo discute sobre a análise de regime transiente em complemento à análise de regime permanente e exemplifica a abordagem proposta com um exemplo de um projeto de investigação duradouro sobre gerenciamento de recursos auto-adaptativo em computação em nuvem. Essa pesquisa objetiva a formulação de metodologias e desenvolvimento de técnicas e ferramentas para avaliação de desempenho de sistemas computacionais sobre carga não-estacionária, bem como o projeto de estratégias de controle para alocação eficiente de recursos. O trabalho, introduzido no presente capítulo como estudo de caso, foca em métricas de desempenho dinâmico, e trata o tópico a partir de duas perspectivas complementares: análise e síntese. Na primeira direção o uso do modelo dinâmico de desempenho é explorado para obter métricas de desempenho transiente tais como máxima degradação de serviço devido à perturbação no ponto de operação e tempo de restauração. Na perspectiva de síntese, ambas essas métricas (sobrepassagem e tempo de assentamento) são tomadas como requisito de projeto para criação de mecanismo de gerenciamento dinâmico de recurso .

5.2 Sistema-exemplo

A tecnologia habilitadora que suporta o desenvolvimento de *datacenters* de grande escala e virtualização de sistemas operacionais hoje exploradas para implementar a elasticidade de recursos tem se desenvolvido desde o modelo de computação em grade. Ao longo dos anos, o modelo de negócio tem amadurecido diferentes formas de comercialização de infraestrutura, plataforma ou software. Naturalmente para efetividade do negócio, a garantia de qualidade de serviço é uma preocupação importante. Historicamente, a pesquisa em QoS tem tratado com duas entidades envolvidas no SLA: o provedor e o consumidor do serviço. O modelo de computação em nuvem envolve duas instâncias dessa interação. De um lado, o datacenter provê infraestrutura para o cliente, por exemplo um operador de comércio eletrônico. Do outro lado, o cliente opera como um provedor de plataforma ou software para o usuário final, que representa a geração de receita. Substanciais contribuições de pesquisa tem focalizado ou o acordo entre datacenter e cliente, ou o acordo entre cliente e usuário final. A consolidação dessas três entidades tem sido de interesse da pesquisa na área.

Um dos trabalhos em desenvolvimento no grupo de pesquisa onde o presente estudo é realizado é uma arquitetura de gerenciamento de recursos adaptativos em computação em

nuvem alinhada com essa perspectiva (NOBILE, 2013). O trabalho considera que, de um lado, o cliente deseja entregar o melhor desempenho para o usuário final em benefício de sua fidelidade, e de outro lado, deseja atingir esse objetivo a um custo mínimo, consumindo a menor quantidade de recursos (máquinas virutais, VMs) do lado do *datacenter*. Uma situação ótima corresponde a um balanço eficiente no cumprimento desses dois objetivos, desempenho e custo. A abordagem investigada nesta pesquisa objetiva manter a taxa de utilização da infraestrutura em nível especificado. Uma utilização muito baixa significa que VMs estão em excesso e podem ser desligadas, enquanto uma utilização muito alta significa que o sistema está operando próximo do limite e sujeito a sobrecarga. Como encontrar a utilização ótima é um problema além do escopo deste exemplo e pode ser tratado por diferentes métodos.

Seja um cenário em que para maximizar o lucro do cliente, uma política de eficiência ótima calcule que o objetivo do controle deva manter a utilização de VMs em 70%, que define o *set point*, ou referência. Sem assumir um tratamento sistemático, uma ideia trivial seria adotar um mecanismo de contrarreação proporcional conforme descrito: monitorar continuamente a utilização e alocar ou desalocar VMs em razão proporcional direta. Se a reação for capaz de trazer a utilização ao *set point* imediatamente após uma perturbação, um modelo de desempenho estático não seria melhor do que um modelo dinâmico. Esse não é o caso, entretanto, do exemplo estudado: o processo de alocação de uma nova VM em um *hypervisor* não é instantâneo, dado o inevitável tempo de *booting* (carregamento) que, na tecnologia atual, não é desprezível. Em um cenário em que a utilização atual é ligeiramente superior à referência e a alocação de uma única VM é suficiente para diminuir a utilização abaixo do limite desejado, o resultado esperado é que mesmo com uma carga de trabalho estocástica estacionária, a política de adaptação proporcional pode levar a um regime de contínua alternância de ligamento e desligamento de VMs, o que é altamente ineficiente. Uma solução *ad hoc* para mitigar esse efeito é relaxar a ativação dessa regra: em vez de executar a ação cada vez que a utilização desvia do *set point*, uma faixa aceitável é definida. Por exemplo, o mecanismo de restauração pode ser ativado somente quando a utilização extrapole a faixa de $70\% \pm 10\%$. Essa heurística, conhecida como histerese, é utilizada por exemplo em controle térmico de refrigeradores para evitar o excessivo chaveamento do dispositivo de resfriamento. Se a utilização desce abaixo do limite inferior, o controle proporcional é ativado e começa a alocar VMs até chegar à referência — e, assim, analogamente, se a utilização exceder o limite superior. Enquanto a saída flutua na faixa 60% a 80%, o conrtole permanece inativo.

Uma estratégia alternativa e bastante diferente para mitigar o efeito liga-desliga excessivo, é tornar mais lenta a reação de controle. Isso pode ser implementado por um acumulador ponderado que, a cada ciclo, mede a utilização e compara com a referência, adicionando a diferença positiva ou negativa ao acumulador. Um fator ponderador menor que 1 tem o efeito de filtrar variações de curta duração na carga de trabalho à medida que o desvio da utilização em relação à referência precisa perdurar por alguns ciclos a fim de que o contador seja incrementado ou decrementado por um valor inteiro.

5.2.1 Simulação

A fim de comparar duas políticas de gerenciamento, um experimento de simulação para avaliação de desempenho foi projetado. As métricas de desempenho consideradas foram: utilização (objetivo do controle), quantidade de VMs ao longo da simulação, tempo médio de resposta e a proporção de violações do SLA — para esse propósito, foi considerado que uma requisição atende ao contrato se processada com atraso não superior a 50% do seu tempo esperado de execução; do contrário, uma falta é computada.

O tempo de *booting* das VMs foi assumido aproximadamente constante e fixadas em $70tu$[1]. A carga de trabalho foi caracterizada como um processo Poissson com taxa de chegada λ e taxa de serviço μ. O experimento foi replicado 16 vezes com diferentes sementes aleatórias e as variáveis de resposta foram estatisticamente tratadas para obter-se um intervalo de confiança de95%. A frequência de amostragem foi ajustada para coletar dados a cada $10tu$.

O experimento foi realizado com o auxílio do *framework* OnlineCloundSim introduzido no capítulo 3. O ambiente de simulação foi configurado com recursos suficientes para atender à carga de trabalho. O datacenter foi configurado com 200 servidores físicos independentes, com núcleos x86 de 1000 MIPS cada, 4096 MB de memória primária e 100GB de memória de armazenamento. As máquinas físicas foram conectadas por uma rede de 10Gbps, executando sistemas operacional GNU Linux com gerenciamento de VMs de memória compartilhada Xen. Na simulação a entidade OnlineBroker foi instanciada em suas classes implementadas. Considerou-se uma política de alocação de VMs obedecendo a um modelo de máquina de estado. Se uma VM está operacional, recebe carga de um despachador que distribui requisições ao conjunto de máquinas de forma balanceada. Se todas as VMs estão operacionais e o *broker* precisa reduzir o conjunto de recursos ativos, algumas VMs param de receber carga e são movidas para o estado *bleeding* (sangria), no qual permanece até que a carga restante seja totalmente processada; em seguida a VM é desligada. Uma VM em *bleeding* pode ser transferida de volta para o estado operacional antes de ser desligada se o *broker* precisa aumentar o conjunto de recursos ativos. Se não existem VMs nessa condição quando o *broker* precisa de mais recursos, uma nova VM é alocada e posta em estado *booting* pelo tempo necessário para completar a inicialização, quando torna-se operacional. Característica de software e hardware influenciando tempos de *booting* e *bleeding* são majoritariamente responsáveis pela dinâmica do *datacenter* simulado.

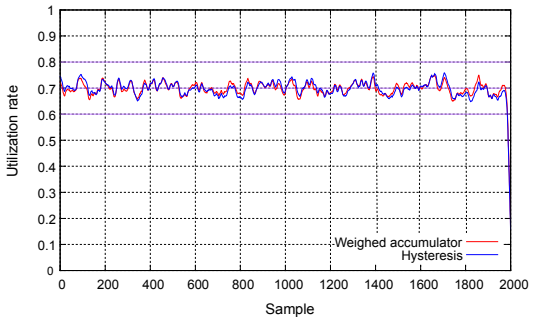
5.2.2 Avaliação de desempenho estacionária

Seguindo o protocolo convencional de análise de regime permanente, o experimento iniciou com a submissão de uma carga estacionária com tempo entre chegadas $1/\lambda = 5tu$ e tempo médio de execução $1/\mu = 70tu$. O experimento iniciou com 30 VMs e o fator ponderador

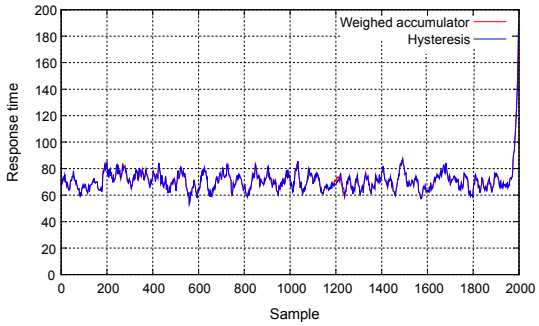
[1] Utiliza-se a medida *tu* para unidade de tempo.

do acumulador ajustado entre 0,1. A Figura 37 mostra os resultados da simulação.

(a) Taxa de utilização.



(b) Tempo de resposta.



(c) Máquinas Virtuais.

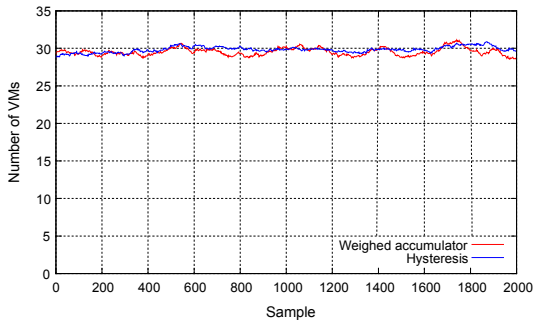


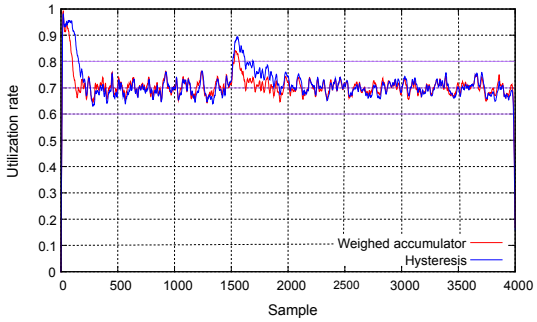
Figura 37 – Dados de regime estacionário.

Como visto na Figura 36a, não há diferença imediatamente visível entre os dois controladores com respeito à taxa de utilização mantida ao longo do tempo do experimento. O mesmo vale para o tempo médio de resposta na figura 36b, bem como para o número de VMs na figura 36c. Dado que a utilização é abaixo de 100% ao longo da simulação, é esperado que o sistema nunca seja sobrecarregado e nenhuma requisição seja penalizada no regime permanente. Conclui-se assim que ambos controladores possuem o mesmo desempenho e a escolha entre eles pode ser deixada a outros critérios do projeto.

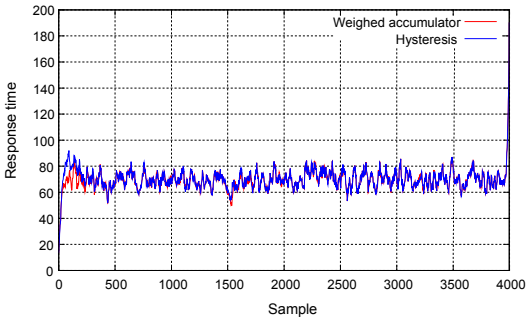
5.2.3 Avaliação de desempenho não-estacionária

Considerem-se agora os gráficos da Figura 38, que representam um experimento similar em que a carga não é mantida estacionária. Nesse experimento, no instante $1500tu$ o tempo entre chegadas de requisições é aumentado abruptamente de $1/\lambda = 5$ para $1/\lambda = 7.5$, significando um potencial aumento da utilização em 50%. Dado que o mecanismo de alocação adaptativo está em funcionamento, esse valor não é alcançado pois o número de VMs é aumentado.

(a) Taxa de utilização.



(b) Tempo de resposta.



(c) Máquinas virtuais.

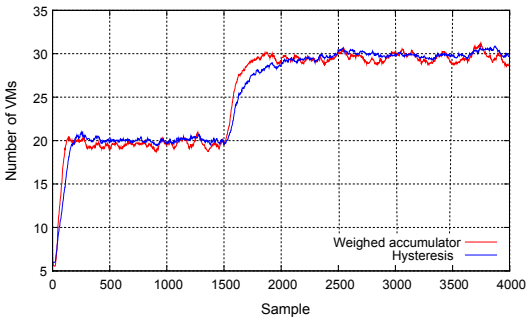


Figura 38 – Dados do regime transiente.

Os gráficos mostram ambos os regimes permanente e transiente alcançados durante o experimento. O período k = 0 a k = 500 corresponde ao *warm-up*: iniciando da condição de repouso sem carga residente, o sistema recebe requisições, que leva algum tempo para que o controlador perceba que o número de vezes não é suficiente e passe a alocar novas instâncias; leva também algum tempo para que as filas dessas novas VMs sejam preenchidas. A partir daí o

sistema está no ponto de operação estacionário. No instante k = 1500 quando a perturbação é propositadamente aplicada, a utilização sobe temporariamente (figura 37a) e então é forçada de volta à referência por força dos controladores, durante o período transiente. O efeito transitório é visível na escala do gráfico. Durante o transiente o controlador acumulador foi capaz de reduzir o máximo tempo de resposta e também de causar seu retorno ao estacionário mais rapidamente. Note-se também que o número médio de VMs utilizadas durante a simulação foi aproximadamente o mesmo, significando que o custo de oferecer uma melhor qualidade de serviço não foi significativamente maior, o que implica um resultado melhor para o operador do sistema.

O gráfico da Figura 39 apresenta uma avaliação quantitativa do mesmo experimento computando o número de requisições penalizadas em cada caso. Nota-se visível superioridade do acumulador.

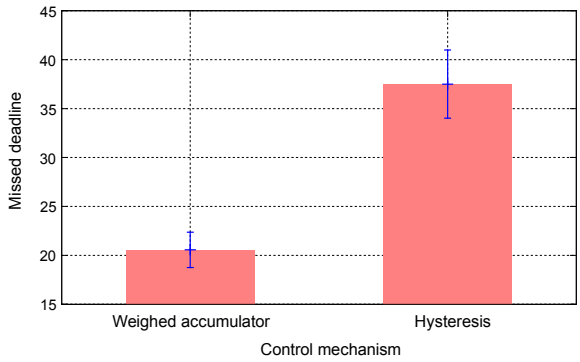


Figura 39 – Número de requisições penalizadas durante o regime transiente.

5.2.4 Aplicando o modelo dinâmico de desempenho

Os resultados dos experimentos descritos na seção anterior revelam que a diferença entre ambos os controladores é mais devido às diferenças no seu comportamento transiente do que na sua capacidade estacionária. Isso tende ser verdade para sistemas que exibem propriedades dinâmicas apreciáveis e sujeitos a altas variações de carga.

Até esse ponto a discussão concentrou-se na relevância da avaliação de desempenho transiente, exemplificado no ambiente de computação em nuvem. Contudo, um ponto a ser observado é que a abordagem para modelagem dinâmica de desempenho investigada no estudo é capaz de prever esses resultados. Os dois diferentes controladores apresentados foram introduzidos como soluções *ad hoc* e analisados por meio de simulação. Entretanto, em uma abordagem de modelagem dinâmica seria possível chegar às conclusões obtidas com menos esforço e em adição revelar mais do comportamento esperado do sistema sob carga variável. Ainda, a abordagem provê ferramentas não apenas para análise, mas também para o projeto de estratégias para atender aos requisitos de desempenho dinâmico, tais como máxima degradação de desempenho após perturbações de carga e máximo tempo de recuperação até o nível de desempenho desejado.

Como exemplo, a próxima seção reanalisa o controlador de acumulador ponderado por meio da abordagem proposta.

O primeiro mecanismo de adaptação apresentado anteriormente é denominado controle proporcional com histerese e pode ser modelado analiticamente por ferramental matemático existentes. Contudo, em virtude da sua importância em aplicações reais, esta seção focaliza o segundo controlador. No jargão de Engenharia de Controle, mecanismo acumulador ponderado desempenha a função de um controlador integral. O diagrama de bloco da Figura 40 representa o caminho entrada-saída através dos componentes do sistema em uma malha de controle realimentada.

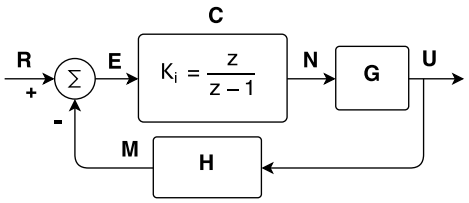


Figura 40 – Malha fechada de um controlador integral.

G é o sistema de interesse, cuja saída deve ser controlada. No exemplo, representa o *datacenter* com o sinal de entrada medindo o número de VMs a serem ativadas e a saída, de desempenho, medindo a taxa de utilização. Note que N é a ordem enviada para o gerenciador de recursos alocar ou desalocar VMs, e não o número de VMs ativas, o qual é um estado interno influindo máquinas ativas, em *bleeding* e em *booting* (a ordem N não tem efeito imediato, constituindo uma inércia que contribui para a dinâmica do sistema). O objetivo de controle é manter a taxa de utilização em torno da referência R. O mecanismo de controle realimentado pulsiona subtraindo a saída efetiva da referência. Se a diferença E é positiva, então N deve ser elevado; se for negativa, deve ser diminuído. O bloco C é o controlador e sua função é decidir o quanto aumentar ou diminuir N em função de E. Em um controlador meramente proporcional, C é apenas uma constante multiplicando E. Em um controlador dinâmico, C é um sistema dinâmico. No exemplo, C é um integrador

$$n(k) = n(k-1) + K_i e(k-1) \tag{5.1}$$

K_i, sendo a constante integradora. O modelo ARX no domínio do tempo pode ser convertido para sua função de transferência no domínio da frequência pela transformada Z.

$$U(z) = z^{-1}U(z) + K_i z^{-1} N(z) U(1-z^{-1}) = z^{-1} K_i N(z) C = \frac{U(z)}{N(z)} = K_i \frac{1}{z-1} \tag{5.2}$$

Enquanto C pode ser composto dedutivamente, G é um sistema d caixa-preta a ser modelado experimentalmente. Utilizando a abordagem proposta, para a obtenção do modelo

analítico de G, o seguinte experimento foi projetado utilizando as capacidades do framework OnlineClounSim introduzido no Capítulo 3. O ponto de operação do sistema foi fixado em $1/\lambda = 5$ e $1/\mu = 70$, e o número de VMs em 26. Nessas condições o sistema apresenta uma ocupação de 54%. No tempo de simulação $k = 50$, seguindo o protocolo proposto, uma perturbação integral foi aplicada à entrada reduzindo-se o número de VMs de 26 para 14.

A fim de aplicar o método de identificação, as séries de dados foram normalizadas no tempo e amplitude. Nesse procedimento, a perturbação foi transladada para o tempo k=0 e os sinais foram deslocados no eixo das ordenadas para um offset igual aos seus valores médios. Utilizando o princípio de parcimônia, a sistema mais simples candidato a ajustar a resposta ao degrau foi determinado como sendo um LTI de primeira ordem, posto que não foram observadas oscilações. Utilizando o método dos mínimos quadrados, os parâmetros do ARX foram encontrados como

$$u(k) = 0,5761u(k-1) - 0,016263n(k-1) \tag{5.3}$$

o que se traduz na função de transferência

$$G(z) = \frac{U(z)}{N(z)} = \frac{-0,016263}{z-0,5761}. \tag{5.4}$$

Essa é a função de transferência que aproxima a dinâmica do *datacenter* por um modelo linear e de baixa ordem. A partir da expressão analítica várias conclusões são possíveis. Uma delas é que o modelo estático do sistema, o qual teria sido obtido pelo protocolo de avaliação de desempenho estacionário é

$$U = -0.0387N \tag{5.5}$$

isto é, a informação de que o acréscimo de uma máquina virtual, no regime de operação, decresce ta taxa de utilização em 0.0387. Esse resultado é derivado diretamente do teorema do valor final

$$\lim_{k\to\infty} x(k) = \lim_{z\to 1}(z-1)Y(z) \tag{5.6}$$

É também possível prever o tempo de duração do transiente a partir das equações vistas no Capítulo 3 Além disso, a representação no domínio do tempo pelas funções de transferência permite a manipulação algébrica, de modo simples, da malha fechada a fim de obter seu modelo analítico. Aqui, deseja-se saber a relação entre R e U, que informa o quão rapidamente a taxa de

utilização efetiva é corrigida quando uma ordem para tal é emitida pelo operador. A partir da Figura 40, tem-se

$$F_{cl}(z) = \frac{C(z)G(z)}{1 - H(z)C(z)G(z)} \tag{5.7}$$

A equação 5.7 é uma representação matemática da dinâmica da malha fechada do sistema de controle. Por métodos analíticos ou numéricos é possível predizer a resposta temporal do sistema como um todo. Por exemplo, seria possível obter os valores de sobrepassagem e temppo de assentamento utilizando um software como Matlab ou GNU Octave, sem a necessidade de repetir as simulações para cada nova entrada — desde que respeitadas as hipóteses da faixa de operação. Como exemplo, os gráficos da Figura 38 mostram a saída do sistema frente a uma perturbação abrupta de 50% no ponto de operação. É possível observar a ação do sistema de controle, evidenciando a robustez do mecanismo adaptativo.

CAPÍTULO

6

ANÁLISE DE RESPOSTA EM FREQUÊNCIA

Este capítulo apresenta um dos resultados importantes do desenvolvimento da abordagem de avaliação de desempenho baseada em análise de resposta em frequência. O método consiste na utilização da caracterização de atenuação dependente da frequência, conhecido como Diagrama de Bode. Essa relação, usualmente representada na forma gráfica, informa qual a atenuação sofrida por um sinal senoidal de entrada, em função da sua frequência. Lembrando que a representação no domínio da frequência é uma descrição de como o sistema responde à sinais periódicos, o significado intuitivo do Diagrama de Bode é o de que, se o sinal de entrada puder ser decomposto em senoides, então pelo princípio da superposição dos sistemas lineares, sabendo-se o que ocorre com cada componente de frequência na entrada, é possível prever o que ocorre com a saída do sinal, que é a soma de todas as componentes aplicadas ao sistema.

No gráfico de resposta em frequência, o eixo das abscissas indica as frequências; o eixo das ordenadas indica a atenuação sofrida naquela frequência. Geralmente, para sistemas físicos, um sinal de frequência zero (constante) não é atenuado (por frequência) e atenuação tende a ser grande para frequências acima de certo valor. No arcabouço conceitual de avaliação de desempenho, o gráfico é capaz de mostrar como as variações de carga de trabalho impactam o desempenho. Pode evidenciar, por exemplo, que perturbações bruscas, mesmo de pequena amplitude, podem produzir impactos no desempenho mais significativos que perturbações de grande amplitude, porém mais lentas.

O diagrama de resposta em frequência pode ser obtido experimentalmente aplicando-se entradas (cargas de trabalho) senoidais ao sistema e medindo a atenuação sofrida na saída (alteração desempenho). Esse procedimento pode ser experimentalmente complicado. Alternativamente, o diagrama de resposta em frequência pode ser obtido diretamente da função de transferência do sistema. O estudo de caso descrito neste capítulo utiliza essa técnica para ilustrar a utilidade desse tipo de análise de resposta em frequência para avaliação de desempenho.

O caso de uso utiliza uma aplicação de computação em nuvem implementada fisicamente

e medida com o auxílio da ferramenta OnlineBench4q. A aplicação é submetida a um regime carga de trabalho com rajadas.

Os resultados demonstram como a análise do diagrama de resposta em frequência foi capaz de prever, por exemplo, que para o ponto de operação escolhido, uma rajada com picos de duração de $15s$ produz um impacto não maior que 38% daquele que seria obtido se a duração dos picos tivesse duração muito prolongada. Além disso, que picos de duração maior que $35s$ não causam oscilação no desempenho (e são percebidos pelo sistema como cargas estacionárias); bem como prever que o sistema é praticamente imune a picos com duração menor que $5s$, os quais são praticamente absorvidos pela dinâmica do sistema e não modificam o desempenho.

6.1 Sistema em estudo

Como estudo de caso foi utilizado um sistema real para análise da resposta em frequência. A ideia foi a de utilizar a mesma proposta descrita em (MAMANI *et al.*, 2015), a qual fundamenta-se na implementação de um sistema de controle que regula a quantidade de máquinas virtuais (VMs) em função das condições operacionais do sistema. No referido artigo utilizou-se o CloudSim com a extensão MEDC. O presente estudo de caso faz a extensão do *benchmark* Bench4Q (ZHANG *et al.*, 2011), adequando-o à proposta MEDC.

Muito embora ele possua funcionalidades bem definidas, o Bench4Q foi projetado basicamente para a análise e produção de resultados estacionários. Basicamente, a ferramenta é orientada a qualidade de serviço (QoS) e trata-se da extensão do TPC-W (*Transaction Processing Performance Council — transational web benchmark*) (MENASCE, 2002). As principais melhorias do Bench4Q incluem o suporte a métricas baseadas em sessões (não apenas em requisições) e geração de carga de trabalho sensível a parâmetros de QoS para fins de planejamento de capacidade (ZHANG *et al.*, 2011).

A configuração básica do ambiente inclui o *console*, uma interface gráfica que permite ao analista configurar como o experimento será executado; os *emulated browsers* (EBs) que atuam como os clientes do sistema em estudo; o sistema em estudo (*System Under Test* (SUT)), que, no caso corresponde a uma aplicação de *e-commerce*, mas que pode ser substituída caso seja de interesse da análise; a gerência dos EBs é feita por agentes (*agents*) que monitoram a utilização de recursos dos *hosts* que hospedam os EBs.

O esquema de implementação do lado do servidor contempla as camadas de aplicação e persistência de dados. A camada de aplicação possui um balanceador de carga que recebe as requisições e as encaminha para o servidor de aplicação mais conveniente. Cada servidor de aplicação possui instalada uma cópia do SUT, que, neste caso, implementa a lógica de uma loja de *e-commerce*. As transações SQL são enviadas à outra camada que corresponde a um servidor de banco de dados tradicional (MySQL, PostgreSQL, DB2 etc.). Portanto, uma requisição chega ao balanceador de carga, que a repassa para o servidor de aplicação, que gera requisições no

banco de dados. Os resultados do banco de dados são enviados para o servidor de aplicações, que retorna para o balanceador de carga que encaminha a resposta ao cliente.

Do lado do cliente, em cada máquina destinada a geração de carga, há a instância do *agent*. Ele irá instanciar um número n de EBs. As sessões serão criadas pelos EBs e cada sessão possui r requisições. É importante observar a característica de geração de carga distribuída pela ferramenta, uma vez que os *agents* podem estar em qualquer lugar desde que haja conectividade. Durante o ciclo de vida de requisições as métricas são armazenadas. Ao final do experimento elas são enviadas ao *console* para serem representadas graficamente ao analista.

O mecanismo de monitoramento disponibilizado pelo Bench4Q atua do lado do cliente, nos EBs. As métricas básicas disponíveis são: tempo de resposta, número de requisições e erros por sessão. As métricas de QoS são: sessões válidas por segundo, número médio de sessões válidas, função densidade de probabilidade (PDF) da duração das sessões e interações web por segundo. Do lado do servidor não há mecanismo disponibilizado pelo *benchmark*, mas utiliza-se ferramentas tradicionais dos sistemas operacionais para obtenção de métricas como utilização de CPU, memória etc., bem como aquelas disponibilizadas pelo balanceador de carga: quantidade de conexões TCP abertas por segundo, quantidade de servidores disponíveis etc.

Uma funcionalidade importante do TPC-W e, portanto, também do Bench4Q é a geração de carga realística baseada no comportamento dos clientes. Isso é descrito pelo *customer behavioral model graph* (CBMG) (grafo representativo do comportamento do cliente) (MENASCÉ *et al.*, 1999). Trata-se de um grafo direcionado com pesos, esses pessos correspondem à probabilidade de mudar de um nó para outro; cada nó representa uma página do SUT. Ao configurar esse grafo consegue-se obter diferentes cargas de trabalho que impactam diferentemente no SUT. Por exemplo, pode-se aumentar a probabilidade de os clientes finalizarem compras. Ou seja, haverá mais condição de disputa e operações de escrita no banco de dados, portanto, uma carga de trabalho com essas características seria mais severa. Analogamente, pode-se configurar uma probabilidade de maior acesso às páginas de produtos e pesquisa — o resultado é uma carga em que as operações de leitura são mais frequentes, caracterizando-a como uma carga menos severa. Há muitas possibilidades de configuração e isso torna o *benchmark* mais flexível, pois pode-se observar uma aplicação real e reproduzir o comportamento típico (estatisticamente) do cliente em experimentos controlados.

Dessa forma, há meios para que sejam estudadas configurações que exponham o SUT a condições operacionais extremas em que o resultado seja tornar evidente os gargalos do sistema (pontos de estrangulamento de desempenho). Por outro lado, os clientes (implementados pelos EBs) finalizarão sessões cujo o tempo de resposta ultrapassem determinado limite de tempo (*timeout*). Portanto, o Bench4Q possibilita a especificação, execução e análise detalhada para avaliação de desempenho do SUT e como isso impacta na QoS.

Como já mencionado, o Bench4Q é orientado a métricas estacionárias. Por exemplo, não possui meios para descrever alteração no número médio de requisições que chegam ao

SUT. O trabalho de doutorado de Mamani (2016) tratou do projeto e desenvolvimento de *benchmarks* para avaliação de desempenho não estacionária com estudo de caso o Bench4Q, e o trabalho de mestrado de Souza (2016) correspondeu ao módulo gerador de carga de trabalho (responsabilidade *Demand* do modelo MEDC) como extensão ao Bench4Q para que fosse possível a geração de cargas variantes no tempo. O presente estudo de caso faz uso dessas implementações. Foram conduzidos experimentos específicos para produção de resultados e análise da resposta em frequência.

A Fig. 41 apresenta um diagrama de blocos que ilustra as modificações e a forma com que os componentes da arquitetura são organizados logicamente. Os EBs foram *modificados* para que seu padrão de requisições pudesse ser alterado em tempo de execução, o módulo *Demand* permite que uma função seja utilizada para gerar as configurações dos EBs. Essa função poderia utilizar dados coletados pelo *Monitor* (por exemplo: diminuir a quantidade de requisições caso o tempo de resposta fosse alto), porém essa funcionalidade não foi utilizada no presente estudo de caso. O *Monitor* coleta dados de taxa de utilização de CPU do servidores de aplicação e quantidade de conexões TCP concorrentes do LB. O módulo *Capacity* processa a utilização de CPU para adicionar ou remover Máquinas Virtuais (VMs) da camada de aplicação; é utilizado um controlador Proporcional-Integral que segue o mesmo projeto de (NOBILE, 2013)— é importante dizer que o projeto do controlador é o mesmo, porém a aplicação é diferente. O *Effector* faz chamadas à biblioteca `libvirt`[1] para a gerência das VMs operacionalizadas pelo virtualizador KVM.

Outro papel do *Effector* é a gerência do ciclo de vida das VMs. Uma VM subutilizada é indicada para ser desligada, enquanto houver requisições na fila de serviços da aplicação, a VM continua ativa e seu estado é marcado como *desligando*. Caso seja necessário, em um instante posterior, adicionar outra VM e houver VM em estado de *desligando*, esta será colocada em estado de *execução*. Analogamente, máquinas em estado de *inicialização* são fortes candidatas a serem desligadas. As VMs que estavam em *execução* e foram para *desligando*, eventualmente ficarão em estado de *pausa* (passarão de *desligando* para *pausa*). Quando uma nova VM for requisitada terão preferência as VMs em estado de *pausa*, pois elas tornam-se úteis em um intervalo de tempo mais curto.

6.2 Análise de linearidade

Esse é o contexto em que este estudo de caso está inserido. O modelo para análise relaciona entrada e saída como ilustrado na Fig. 42, em que F representa o OnlineBench4Q, W a carga de trabalho em requisições por segundo, U e VMs a média de utilização de CPU e a quantidade de VMs que compõem a camada de aplicação do SUT, respectivamente. A utilização do sistema é estimada por uma média móvel exponencial para evitar que o controlador atue em

[1] Libvirt: <http://libvirt.org/>

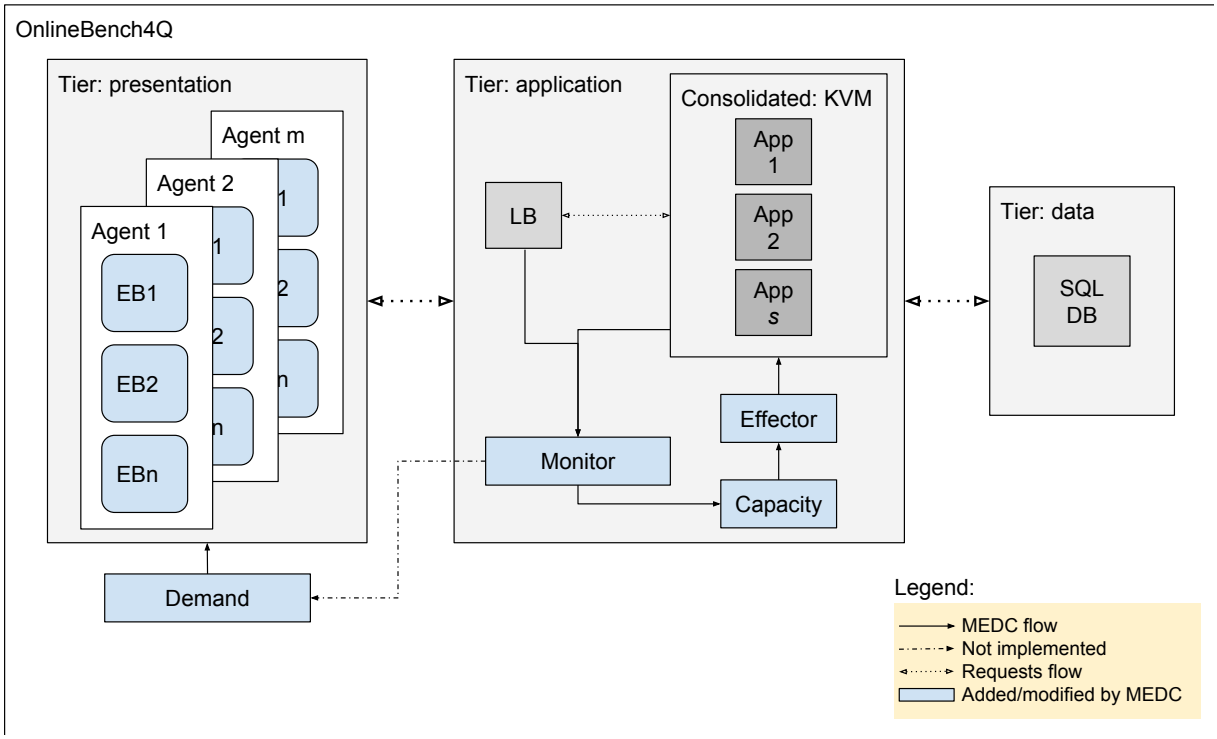


Figura 41 – OnlineBench4Q: Bench4Q + MEDC. Esta ilustração representa o ambiente no qual um experimento é executado. A configuração do experimento é feita através do *console*, não incluso aqui.

surtos requisições de baixa duração, mas que geram alta utilização.

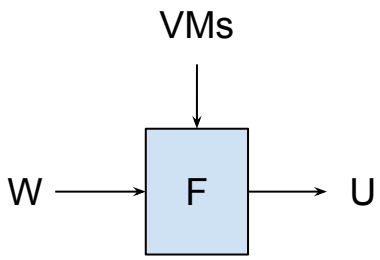


Figura 42 – Modelo a ser estudado. Entrada (W): requisições por segundo. Saída (U): utilização de CPU. Perturbação: número de VMs.

Dado o modelo que relaciona entrada e saída, o primeiro passo para modelagem é verificar a linearidade do sistema em uma determinada faixa de operação. Ao realizar esse experimento pode-se inspecionar a aproximação do sistema por um modelo linear, o ponto de operação do sistema e o ganho estacionário. O procedimento experimental é dado pelo Alg. 1.

Para a execução dos experimentos foi utilizada a infraestrutura disponível no Laboratório de Sistemas Distribuídos e Programação Concorrentes (LaSDPC), onde esta tese foi desenvolvida, e o hardware utilizados está listado na Tab. 8. No balanceador de carga foi utilizado o software HAProxy[2]. O SUT foi hospedado com o servidor de aplicação IBM WebSphere Application

[2] HAProxy: <http://www.haproxy.org/>

Algoritmo 1: Passos para avaliar a linearidade do sistema representado pela Fig. 42. Valores médios obtidos por experimento com 6 réplicas.

```
1 forall n in VMs=6,8,12 do
2     forall u in U=30,40,50,60 do
3         forall r=1 in replica=1-6 do
4             Degrau em W para aumentar U em 15%;
5             Medir W e U periodicamente;
6             Considerar dados estacionários apenas;
```

Server[3], que é uma versão do Apache Tomcat[4] com melhorias de ferramentas para conexões com banco de dados e configurações. O banco de dados utilizado foi o PostgreSQL[5] configurado em modo *cluster*. O tipo da carga de trabalho adotada nos experimentos foi de somente leitura, ou seja, diminuiu-se o gargalho típico banco de dados apresentado neste tipo de arquitetura (3 camadas). O sistema de amostragem permite a coleta com resolução de segundos, isto é, o tempo de amostragem foi 1 s. A rede de interconexão utilizada era do tipo Ethernet com largura de banda 1 Gbps, interligada por meio de comutadores (*switches*).

A distribuição física dos elementos foi dada da seguinte forma. Foi configurado três redes distintas, uma para cada camada: clientes, aplicação e banco de dados; cada rede possuía seu *switch*. Cada máquina cliente possuía uma interface de rede em que direcionavam suas requisições ao balanceador de carga. O balanceador de carga possuía duas interfaces de rede, uma externa para atender as requisições dos clientes e outra direcionada aos servidores de aplicação. Os servidores de aplicação possuíam duas interfaces de rede, uma para atender as requisições enviadas pelo balanceador de carga e outra para direcionar requisições SQL à camada de banco de dados. O servidor mestre de banco de dados possuiu duas interfaces de rede, uma para atender as requisições dos servidores de aplicação e outra para repassar as requisições ao respectivo *pool* de servidores. Cada servidor do *pool* possuía uma interface de rede que respondia às requisições SQL. O serviço de monitoramento era um processo que executava em cada um desses elemento e os dados eram persistidos em arquivos de *log* locais. É importante mencionar que apenas parte desses *logs* foi utilizado neste estudo de caso, os demais dados foram utilizados em outros estudos.

A Tab. 9 contém a configuração necessária que satisfaz o requisito de degrau para análise de linearidade apresentado no Alg. 1. É apresentado para cada quantidade VMs na camada de aplicação, o número necessário de requisições a serem geradas pelos EBs. Ganho é a razão entre a quantidade de requisições e a taxa de utilização, em outras palavras o quanto cada requisição ocupa de taxa de utilização de CPU. Cada célula da tabela apresenta a média da execução de um experimento com 6 réplicas. O ganho médio apresentado na última linha da tabela, corresponde a

[3] WebSphere: <https://www.ibm.com/software/websphere>
[4] Tomcat: <http://tomcat.apache.org/>
[5] PostgreSQL: <https://www.postgresql.org>

Tabela 8 – Configuração do hardware utilizado nos experimentos. O símbolo # representa a palavra *quantidade*.

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	-

média da coluna, e, para 6, 8 e 12 VMs, é $0,0346$, $0,0283$ e $0,0217$, respectivamente. Conforme pode ser observado na Fig. 43, esse sistema pode ser aproximado por um sistema linear. Portanto, a abordagem de pequenos sinais pode ser adotada e considerar que o ponto de operação é linear em torno em torno da quantidade de requisições utilizada.

Tabela 9 – Configurações necessárias para obter o degrau estipulado para a análise de linearidade. Os valores em destaque foram os utilizados para a Identificação do Sistema (ver subseção 6.3)

	VM: 6		**VM: 8**		VM: 12	
CPU	Req/s	Ganho	Req/s	Ganho	Req/s	Ganho
30	801	0,0374	996	0,0301	1366	0,0220
40	1145	0,0349	**1375**	**0.0291**	1851	0,0216
45	1358	0,0331	1612	0,0279	2156	0,0209
50	1412	0,0354	1760	0,0284	2287	0,0219
55	1648	0,0334	2024	0,0272	2589	0,0212
60	1686	0,0356	**2153**	**0.0278**	2726	0,0220
65	1968	0,0330	2358	0,0276	3021	0,0215
75	2238	0,0335	2681	0,0280	3379	0,0222
Ganho	média	0,0346	média	0,0283	média	0,0217

6.3 Identificação do sistema

Para a Identificação do Sistema foi utilizada uma perturbação em formato de degrau. Para este caso, utilizou-se uma quantidade fixa de VMs na camada de aplicação, número de VMs igual a 8. A faixa de operação a ser analisada corresponde a taxa de utilização de 40% a 60%, de forma que a quantidade de requisições por segundo fosse de uma média de 1371 a 2158. O experimento foi replicado 6 vezes e os dados obtidos são apresentados na Fig. 44. Os dados de identificação possuem 112 segundos de duração, com a variável independente tempo indo de -50 a 62. E entrada (*workload* — W) possui valor médio 1371 de -50 a -1 e 2158 de 0 a 62. Os valores anteriores a -50 foram descartados por correspondem ao desempenho do sistema fora da região de operação, ou seja, no instante de tempo -50 o sistema encontrava-se em regime operacional estacionário. A saída corresponde à taxa média de utilização de CPU das VMs da

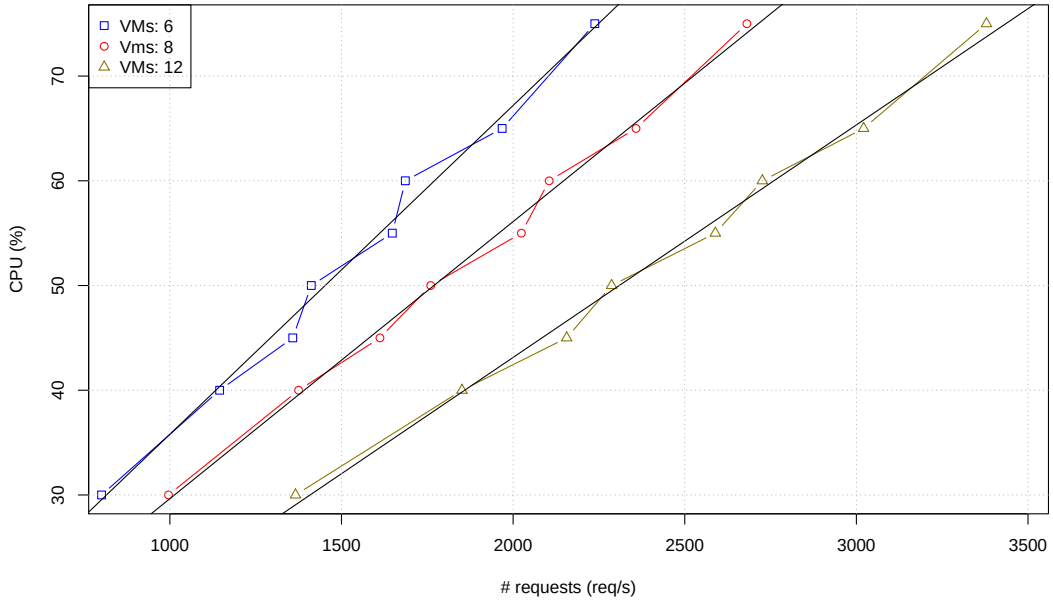


Figura 43 – Análise de linearidade entre número de requisições por segundo e taxa de utilização de CPU.

camada de aplicação passada por um filtro de média móvel exponencial. É possível observar que o relacionamento de causa-efeito entre as duas variáveis pode ser representado por um sistema de primeira ordem.

Os dados empíricos foram passados para a função `tfest` do Matlab ©. Essa função faz a estimativa de parâmetros para o modelo Função de Transferência através método dos mínimos quadrados. O modelo obtido possui *fit* de $92,24\%$ (com foco em simulação) em relação aos dados utilizados para identificação e Erro Final de Predição (*Final Prediction Error* (FPE)) de $0,4407$ e Erro Quadrado Médio (*Mean Square Error* (MSE)) de $0,4277$. O tempo de assentamento foi de 39 s. e o ganho DC foi de 0.0284. Na Fig. 45 é apresentada o resultado simulado pela função de transferência identificada, passando-se como entrada o mesmo degrau. Os valores para RMSE e R^2 foram $0,6478$, $0,988$.

Para a validação do modelo executou-se um experimento no sistema real de modo que o sinal de entrada ficasse fixo em dois patamares em intervalo de 6 segundos. O patamar inferior teve média 1371 e o superior 2158. Para ambas, entrada e saída reais, subtraiu-se o valor médio do sinal, ou seja, eles tiveram média 0 (zero), e oscilaram entre este valor. Isso faz com que os efeitos do transiente na simulação sejam minimizados. O sinal de entrada com média zero foi passado à função de transferência e a saída foi comparada com a saída real. A Fig. 46 apresenta o resultado. O valor de RMSE foi $0,7924$ e R^2 foi $0,8661$. A Fig 47 apresenta o espalhamento dos resíduos em relação aos valores reais obtidos do sistema.

O modelo de Função de Transferência identificado pode ser representado por um ARX

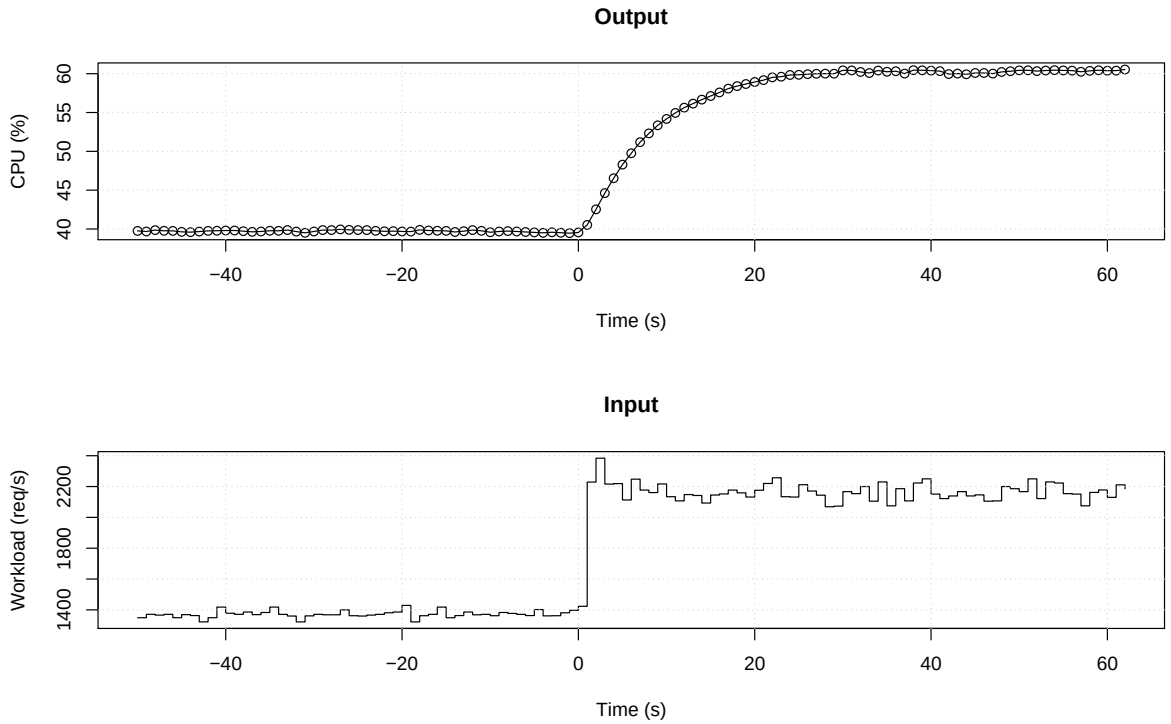


Figura 44 – Dados utilizados para a Identificação da Função de Transferência. Na parte superior os dados de saída (utilização de CPU) e inferior a quantidade média de requisições por segundo. A linha do tempo vai de -50 a 62. O degrau acontece no instante 0.

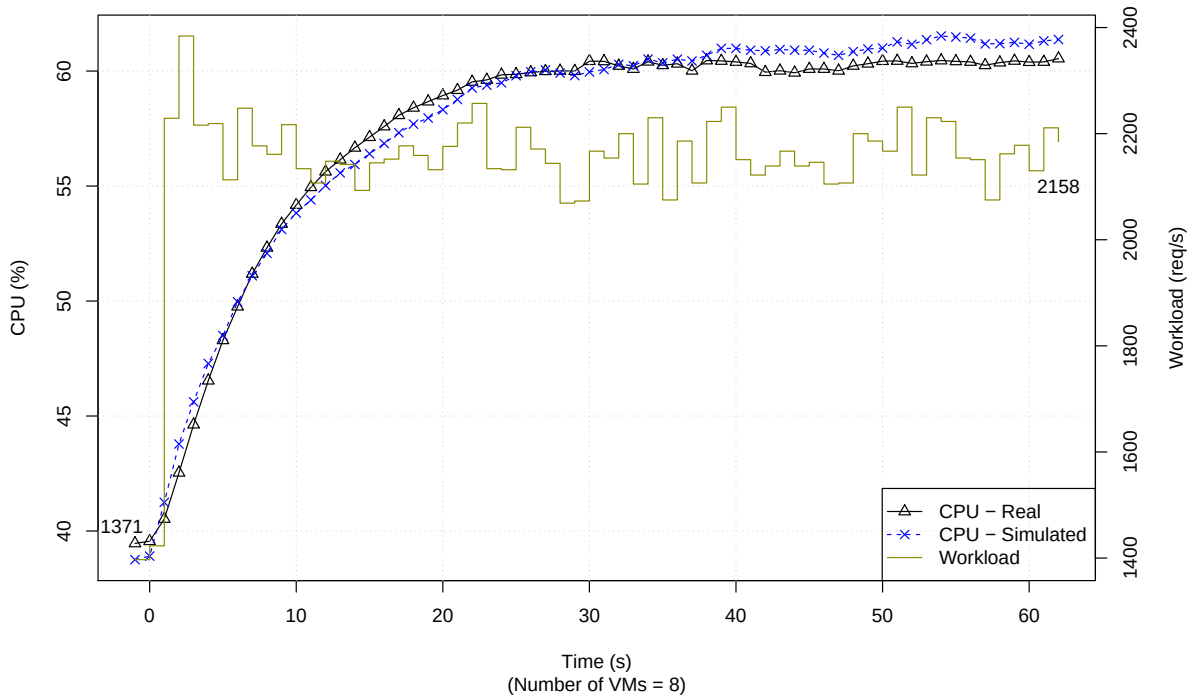


Figura 45 – Dados de identificação (entrada e saída) e simulado pela função de transferência. Os números anotados dentro da área de dados deste gráfico estãos os valores médios de requisições por segundo antes (1371) e depois (2158) do degrau.

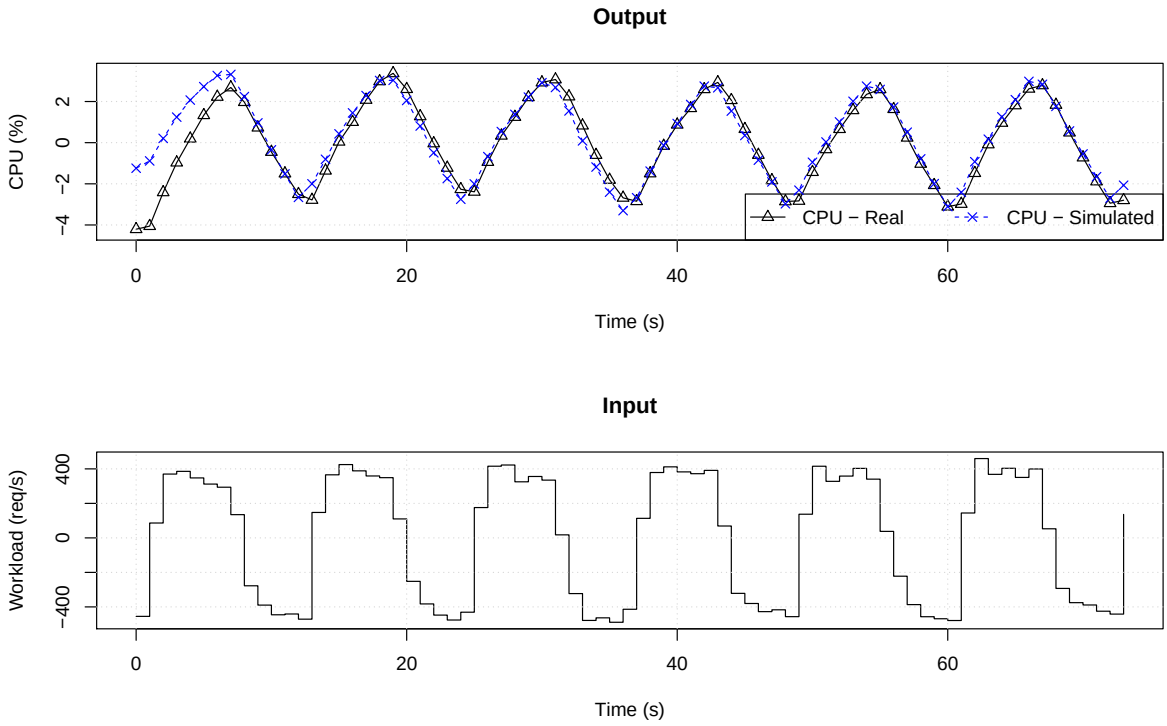


Figura 46 – Validação do modelo identificado. .

com os parâmetros estimados $a_1 = -0.9037$ e $b_1 = 0.00273$, ou seja,

$$y(k) = 0.9037y(k-1) + 0.00273u(k-1). \tag{6.1}$$

A Função de Transferência do sistema é

$$G(z) = \frac{b}{1+az^{-1}} = \frac{0.00273}{1-0.9037z^{-1}}. \tag{6.2}$$

6.4 Análise da Resposta em Frequência de um trem de pulso

De posse da Função de Transferência (Eq. 6.2) e, portanto, fazendo a aproximação linear e invariante no tempo (*Linear time-invariant* (LTI)), a resposta do sistema quando há variações na taxa de requisições no balanceador de carga pode ser analisada conforme a diagrama de Bode (Fig. 49). O diagrama pode ser obtido analiticamente considerando-se uma excitação senoidal (por meio de sua expressão no domínio Z, multiplicada à função de transferência), ou numericamente com o auxílio de ferramenta de software como Matlab ou Octave. Os experimentos descritos nesse capítulo basearam-se nessa segunda forma.

Naturalmente, um sistema LTI excitado por um sinal de entrada senoidal de frequência ω e amplitude A irá apresentar sua saída com mesma frequência (ω), porém com amplitude

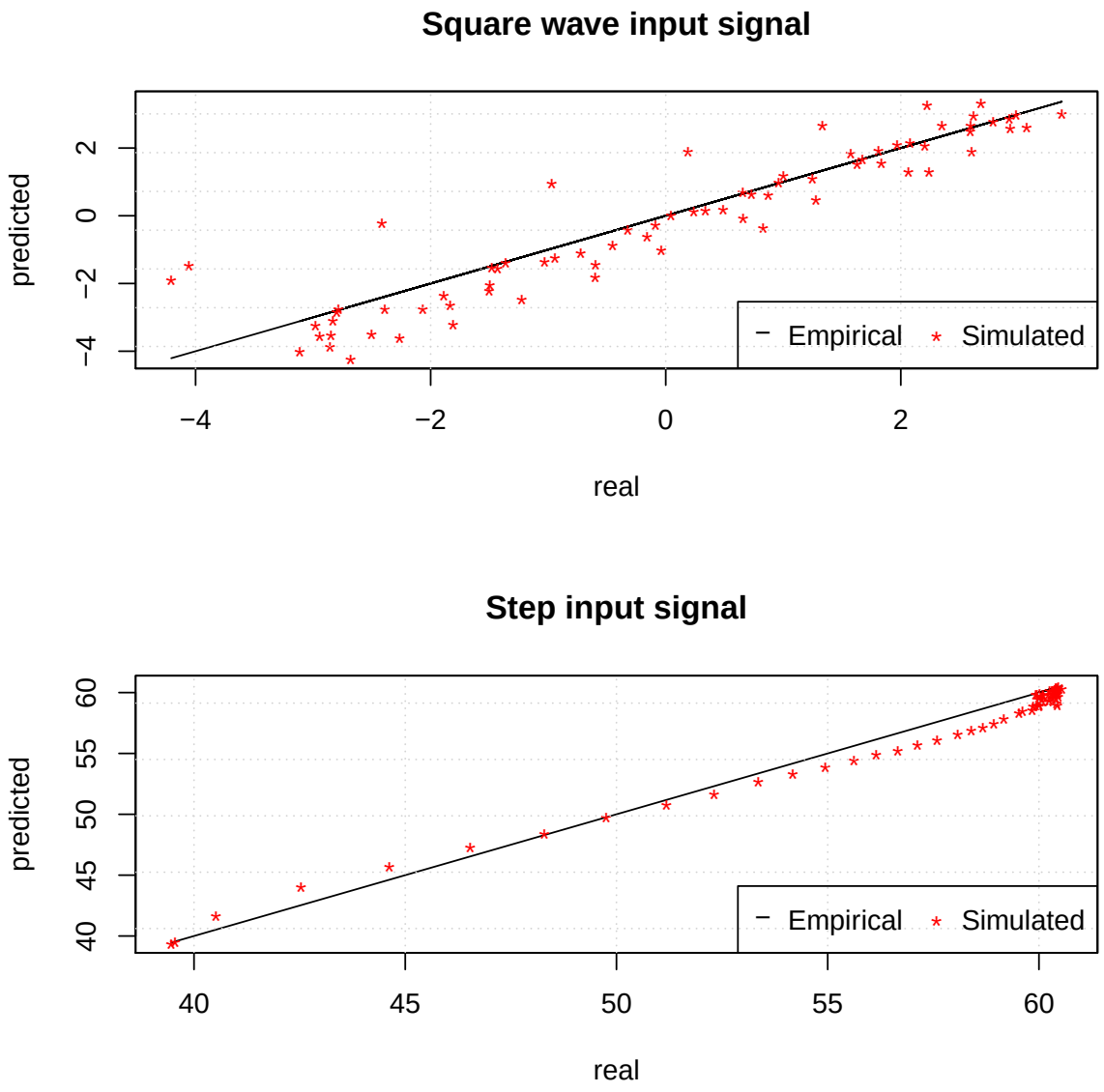


Figura 47 – Resíduo do modelo identificado.

alterada (αA, tal que $\alpha \in (0,1)$). Por inspeção, três frequências serão escolhidas: a frequência de corte (frequência em que o sinal atenuado corresponde a $\sqrt{2}/2 \approx 70\%$ do ganho DC), e as frequências em que o sinal atenuado corresponde a 50% e 25%. A Fig. 48a apresenta os valores relativos que auxiliam na escolha das frequências, e a Fig. 48b apresenta os valores absolutos com a anotação das três frequências escolhidas. Com esses valores escolhidos, é possível inferir quão longo uma perturbação do tipo pulso deve ser para alcançar a respectiva atenuação. Um pulso é caracterizado por duas partes temporais distintas, possuindo suas respectivas amplitudes (A_1 e A_2) e um tempo de duração fixo ($D = \Delta t$). Ele é formado por um sinal com amplitude A_1 alternada para A_2 em determinado instante t e no instante de tempo $t + \Delta t$ a amplitude retorna ao patamar A_1. Um trem de pulsos ou onda quadrada é formada por uma série de pulsos (a Fig. 48 ilustra esse comportamento). Assim, dada uma determinada frequência (f_i), pode-se obter o

comprimento da onda (em segundos) através de sua inversa e a máxima amplitude acontecerá no exata metade desse comprimento (na metade de seu período). Portanto, o degrau do pulso (que alterará da amplitude A_1 para A_2) terá duração de D_i obtido pelo cálculo

$$D_i = \frac{1}{f_i}\frac{1}{2} = \frac{1}{2f_i}. \tag{6.3}$$

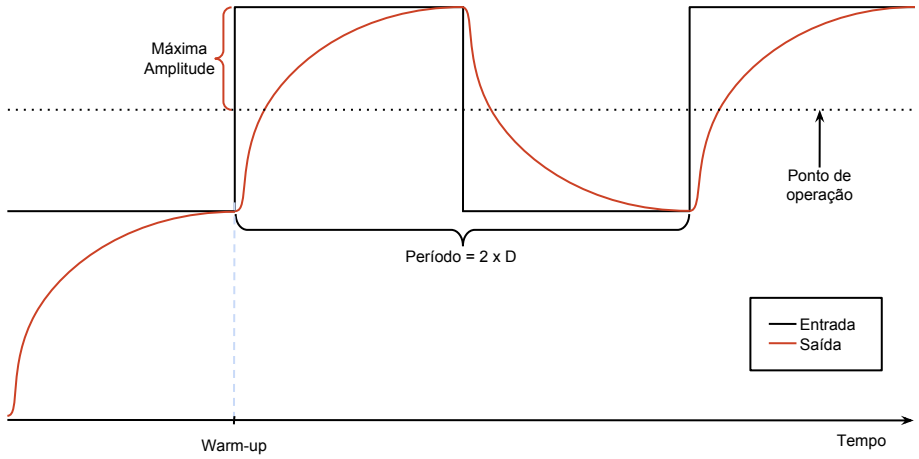


Figura 48 – Ilustração de uma onda quadrada. Na primeira parte do sinal ocorre o *warm-up* do sistema. A onda oscilará em torno do ponto de operação, alternando em duas amplitudes (A_1 e A_2) com duração fixa D. O período da onda corresponde a duas vezes essa D. Pelo diagrama de Bode é possível saber qual a amplitude máxima a ser atingida pela saída com base na magnitude da saída e pela duração fixa.

Considerando o ganho DC do sistema (Eq. 6.2), $0,0284$, a CPU terá média de utilização ($\overline{U} = 50\%$), e os valores de baixa e alta utilização de CPU a serem alcançados pelo experimento são computados como

$$\begin{aligned} U_{min} &= \overline{U} - A/2 \\ U_{max} &= \overline{U} + A/2, \end{aligned} \tag{6.4}$$

em que A é a amplitude computada como

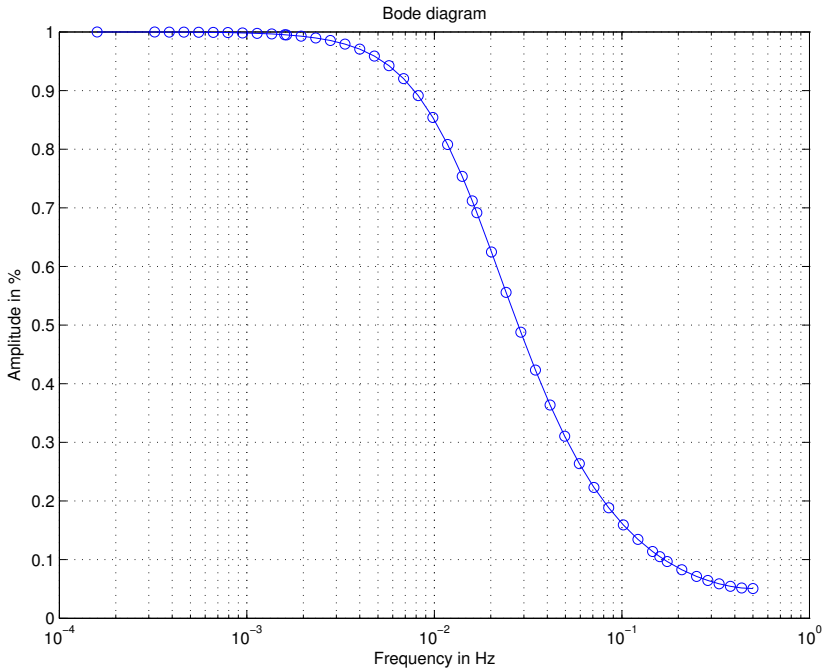
$$A = 2 \cdot Attenuation. \tag{6.5}$$

A atenuação (*Attenuation*) é a razão entre a magnitude do sinal a um determinada frequência e o ganho DC do sistema, ou seja, quanto do ganho final (quando em estado estacionário) será alcançado pelo sistema,

$$Attenuation = Magnitude/DCGain. \tag{6.6}$$

A magnitude é dada em função da frequência e pode ser obtida por inspeção no diagrama de Bode (ver anotações em Fig. 48b). Para obter os valores equacionados nas Eqs. de 6.4 a 6.6, o primeiro passo é obter a frequência de interesse e encontrar a magnitude e assim calcular

(a) Diagram de Bode com atenuação relativa.



(b) Diagrama de Bode com atenuação com valores absolutos.

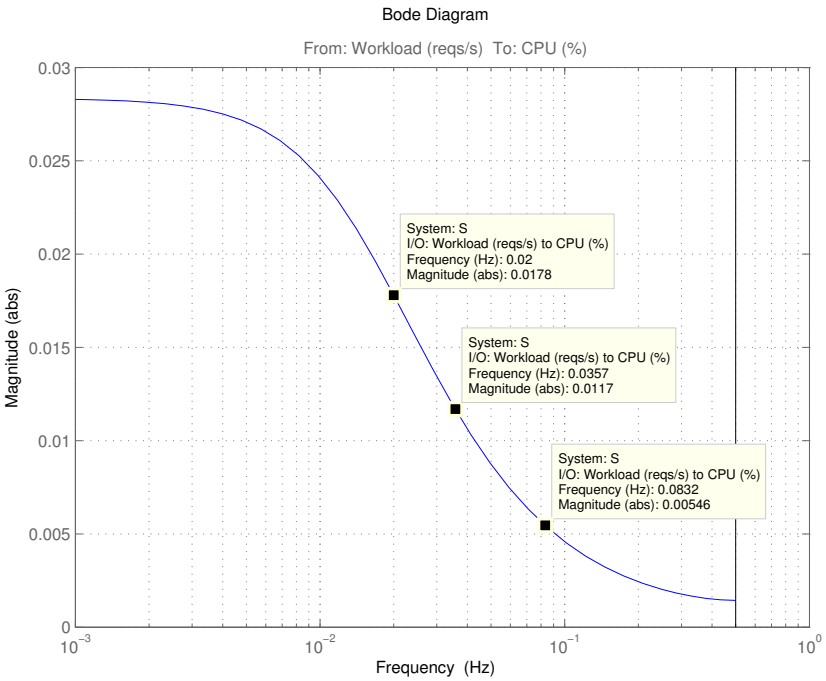


Figura 49 – Diagrama de Bode da Função de Transferência identificada (Eq. 6.2).

a atenuação. A amplitude será computada e, portanto, U_{max} and U_{min} também. No sistema em estudo a média será $\overline{U} = 50$, pois o sistema oscilará por uma onda quadrada que excursionará de 40% a 60% de taxa de utilização de CPU. Desse modo, o protocolo do experimento consistirá em fazer um determinado número de requisições durante um tempo de duração D tal que a utilização da CPU alterne entre baixa (1371 req/s $\rightarrow$ 40%) e alta (2158 req/s $\rightarrow$ 60%).

Para verificar o resultado predito pelo modelo (Tab. 10), experimentos foram conduzidos utilizando-se o OnlineBench4Q. Cada experimento foi replicado 6 vezes. A Fig. 50 apresenta os resultados obtidos. Cada gráfico possui um eixo x (tempo) e dois eixo y, utilização de CPU à esquerda e requisições por segundo à direita. Os resultados foram *plotados* (desenhados) em forma de série temporal, de forma que a média das execução é a linha principal, envolvido por uma sombra que representa o intervalo de confiança de 95%. A Fig. 49a apresenta o resultado que possui o pulso de mais longa duração (25 s.), o que significa a maior amplitude obtida desses experimentos. A CPU oscilou em torno de 50 % de utilização com amplitude de $\pm$6,27 %. A Fig. 49b apresenta o pulso de duração média (14 s.) destes experimentos. A CPU oscilou em torno da média com amplitude $\pm$4,12 %. E a Fig. 49c com a menor duração do pulso (6 s.) oscilou com amplitude $\pm$1,93 %.

Tabela 10 – Predição da utilização de CPU para os valores baixo e alto a serem atingidos pelos sistema em estado estacionário, quando excitado por ondas quadradas de diferentes frequências. (para $\overline{U} = 50$)

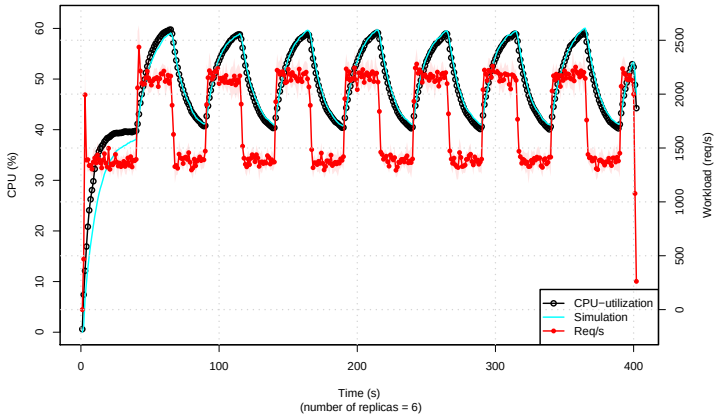
Duração Degrau (s)	Frequência (Hz)	Magnitude (CPU %)	Atenuação (%)	Amplitude (CPU %)	Baixo (CPU %)	Alto (CPU %)
25	0,02	0,0178	62,68	12,54	43,73	56,27
14	0,0357	0,0117	41,20	8,24	45,88	54,12
6	0,0832	0,00546	19,23	3,85	48,08	51,93

Estes resultados evidenciam que, muito embora o sistema tenha características estocásticas em sua natureza, dentro de sua região de operação o comportamento de sua resposta pode ser predito. A Tab. 11 apresenta os resultados obtidos e os respectivos resíduos. Pode-se obter a porcentagem de CPU de diferença entre a predição e os valores obtidos da execução pelo cálculo

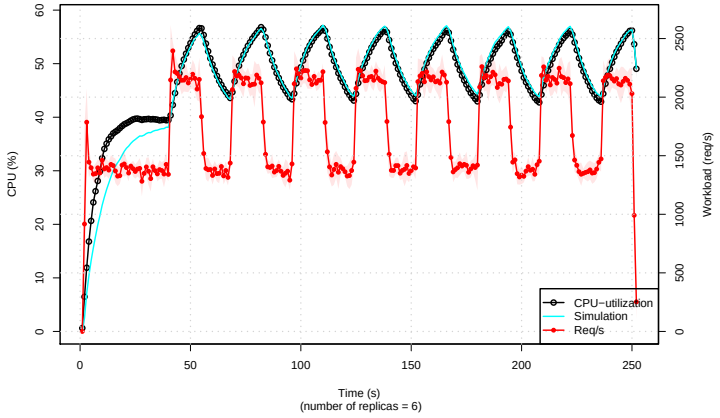
$$Diff = \frac{A_p - A_r}{2}, \tag{6.7}$$

em que A_p é a amplitude prevista (Tab. 10) e A_r a amplitude real (obtida pela subtração dos valores altos e baixos de cada duração do degrau, vide coluna Em da Tab. 11); a divisão por dois significa que metade do erro estará na parte superior (valores de pico) e a outra metada na parte inferior. Para os pulsos de duração 25 s., 14 s. e 6 s. a diferença ($Diff$) entre os valores preditos e os esperados foram de 3.12%, 2.52% e 0,8%, respectivamente. Deve-se mencionar que na Fig. 50 há uma linha adicional referente à simulação do modelo, isto é, a mesma entrada utilizada nos experimentos reais foi passada para a função de transferência identificada, e sua saída foi adicionada ao gráfico. Percebe-se que o modelo foi capaz de representar a dinâmica do sistema. Com isso, esses experimentos reforçam o argumento de que o modelo matemático ARX

(a) Trem de pulso com duração de 25 s.



(b) Trem de pulso com duração de 14 s.



(c) Trem de pulso com duração de 6 s.

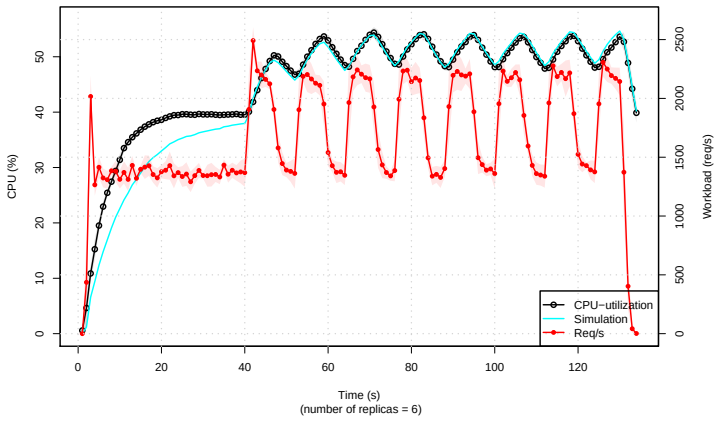


Figura 50 – Resposta em frequência do sistema. A magnitude da excursão do sinal de utilização de CPU diminui quando a frequência do trem de pulso aumenta, como predito pelo diagrama de Bode. A duração do experimento é proporcional à duração do pulso.

para representação da Função de Transferência é útil para predizer a dinâmica apresentada por um sistema computacional. Uma vez que o sistema esteja identificado, os resultados podem ser assumidos como o do sistema real. Em outras palavras, os experimentos do sistema real podem ser substituídos, observada a acurácia do modelo identificado, por simulações com a própria função de transferência. Isso é útil pois permite um menor tempo de obtenção dos resultados de diferentes cenários operacionais para estudo com o sistema em teste. O modelo em si é uma ferramenta para entender como o sistema reage a alterações na capacidade e demanda, caso típico operacional de computação em nuvem.

Tabela 11 – Acurácia dos valores baixo e alto preditos pelo modelo.

Duração Degrau	Predito (Pr) (CPU %)	Empírico (Em) (CPU %)	Resíduo (Pr - Em)
25 alto	56,27	59,095	-2,825
25 baixo	43,73	40,305	3,425
14 alto	54,12	56,372	-2,252
14 baixo	45,88	43,099	2,781
6 alto	51,92	53,406	-1,486
6 baixo	48,08	47,960	0,12

6.5 Análise da Resposta em Frequência de diferentes harmônicos

Como foi evidenciado até o momento, há um comportamento característico do sistema ao ser excitado por um sinal periódico, no caso uma onda quadrada (Fig. 50). Muito embora haja pequenas variações nos patamares altos e baixos daquele sinal (por exemplo, observar Fig. 49c), essas variações não aparecem na saída devido às propriedades dinâmicas do sistema. Nesta seção é descrito um estudo em que são selecionadas estrategicamente 6 frequências e 5 sinais são produzidos de forma a combiná-las diferentemente. As combinações foram adicionadas a um sinal estacionário obtido a partir do sistema real.

Com as mesmas configurações mencionadas no início deste capítulo, foi realizado um experimento de 1 hora de duração, em que retirou-se os dados de *warm-up* no início e no final. Desses resultados, utilizou-se 512 s da variável de entrada, requisições por segundo. Esse experimento foi executado apenas uma vez. A intenção foi produzir dados reais obtidos do sistema e que servissem como sinal base para o presente estudo. A quantidade de requisições ficou estacionada em 1378 req/s. Os 5 sinais foram estrategicamente produzidos, adicionando-se frequências ao sinal estacionário de modo a posicioná-las à direita ou à esquerda de uma certa frequência f_c. Ao escolher uma frequência mais à direita de f_c a tendencia é que haja uma atenuação, ao passo que mais à esquerda, menor a atenuação. Essa atenuação é inspecionada conforme o diagrama de Bode do sistema. Neste caso, f_c é a frequência de corte, isto é, a frequência que, quando utilizada para excitar um sistema, sua amplitude na saída cai para

$\sqrt{(2)}/2$ da amplitude na entrada. Assim, essas frequências têm um impacto maior ou menor no sistema, dependendo da sua posição em relação a f_c. A Fig. 51 apresenta as frequências escolhidas.

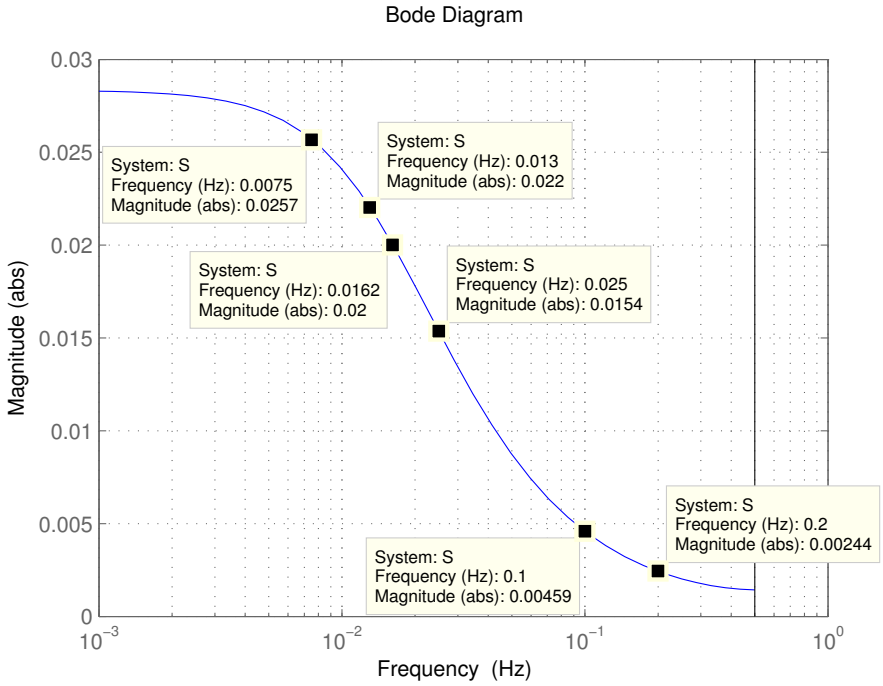


Figura 51 – Diagram de Bode com as frequências escolhidas em destaque.

A ideia é que as frequências altas sejam filtradas pelo sistema, diminuindo drasticamente a amplitude de sua presença na saída. Por outro lado, a frequência baixa será apreciada na variável de saída. A Tab. 12 apresenta a tabulação das frequências escolhidas. A coluna *Amplificação* significa em quanto o valor de um harmônico será aumentado. Por exemplo, seja um sinal s e sua FFT j; cada frequência escolhida tem um harmônico correspondente em j; o valor presente em j e correspondente a uma frequência f_i (conforme estipulado na tabela) é multiplicado pelo respectivo valor da coluna *Amplificação*. Os sinais foram gerados da seguinte forma:

- Sinal 1: frequências f_1 e f_3 adicionadas;
- Sinal 2: frequências f_1, f_4 e f_5 adicionadas;
- Sinal 3: frequências f_1 e f_2 adicionadas;
- Sinal 4: frequências f_1, f_2 e f_4 adicionadas;
- Sinal 5: frequências f_1 e f_c adicionadas.

Os sinais gerados foram aplicados ao sistema através da função `lsim` do Matlab e a saída correspondente foi analisada. O sistema é o mesmo da seção anterior (Eq. 6.2).

Tabela 12 – Frequências selecionadas e respectiva amplificação. A coluna *Amplificação* significa que, após extraído a FFT de um sinal, o valor encontrado naquela frequência será multiplicado pela constante apresentada neste tabela.

Frequência	Valor	Amplificação
f_1	$0,0075$	20
f_2	$0,0130$	35
f_c	$0,0162$	20
f_3	$0,0250$	40
f_4	$0,0100$	40
f_5	$0,0200$	60

O **Sinal 1** possui dois harmônicos cujas frequências são menores que a frequência de corte (f_c). Isso significa que eles são observados na saída do sistema, pois, como possuem uma frequência menor, eles são capazes de excitar o sistema e contribuir com uma parcela significativa quando há a transformação de requisições por segundo em taxa de utilização. A Fig 52 apresenta o resultado obtido. Os harmônicos adicionados artificialmente no sinal estão presentes na saída. Há a diminuição do harmônico de frequência maior, conforme pode-se observar no diagrama de Bode (Fig. 51). Em contraste o **Sinal 2** possui, das frequências escolhidas, a mais baixa e as duas mais altas. A previsão é o harmônico mais baixo seja conservado, porém os dois harmônicos mais altos sejam substancialmente reduzidos. Como pode ser observado na Fig. 53, essa previsão está correta, a frequência f_5 ainda está presente no sinal de saída, pois sua magnitude no sinal de entrada é grande.

(a) Entrada. (b) Saída.

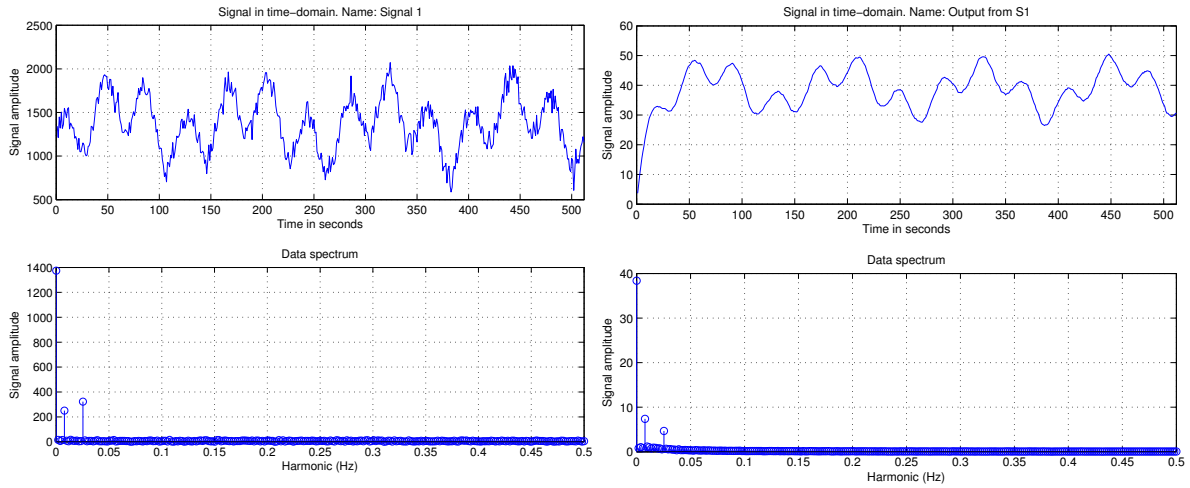


Figura 52 – Sinal 1.

O **Sinal 3** é semelhante ao primeiro sinal. As frequências mais baixas aparecem no sinal de saída, no entanto, dessas duas frequências mais baixas, a maior tem uma atenuação também maior, conforme pode ser visto na Fig. 54. O quarto sinal (**Sinal 4**) a frequência f_4 é adicionada ao terceiro sinal. Conforme pode ser observado na Fig. 55, o comportamento principal

(a) Entrada. (b) Saída.

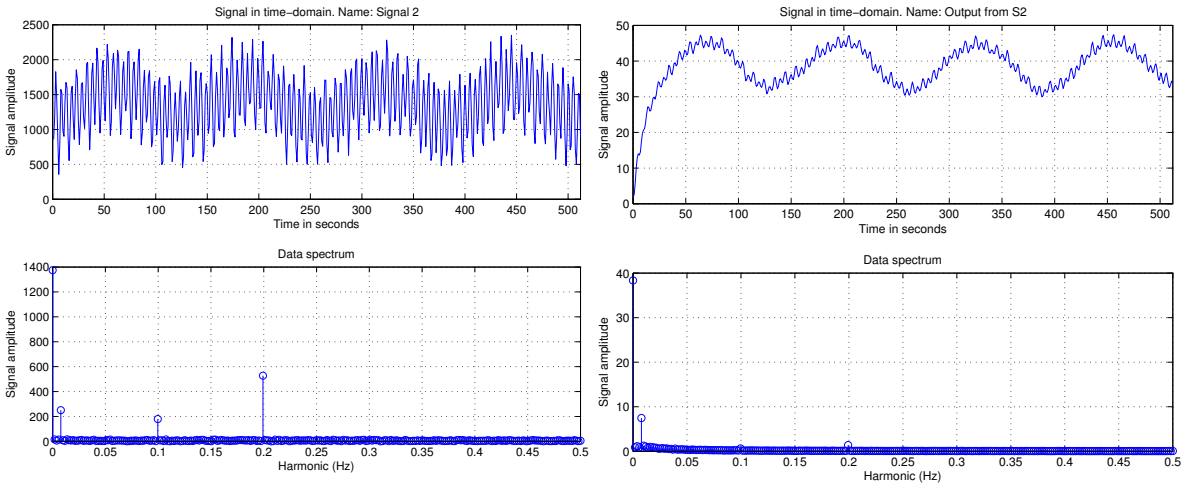


Figura 53 – Sinal 2.

do sistema quando observada sua saída é o mesmo do terceiro sinal, adicionado do harmônico correspondente a f_4, com baixa magnitude.

(a) Entrada. (b) Saída.

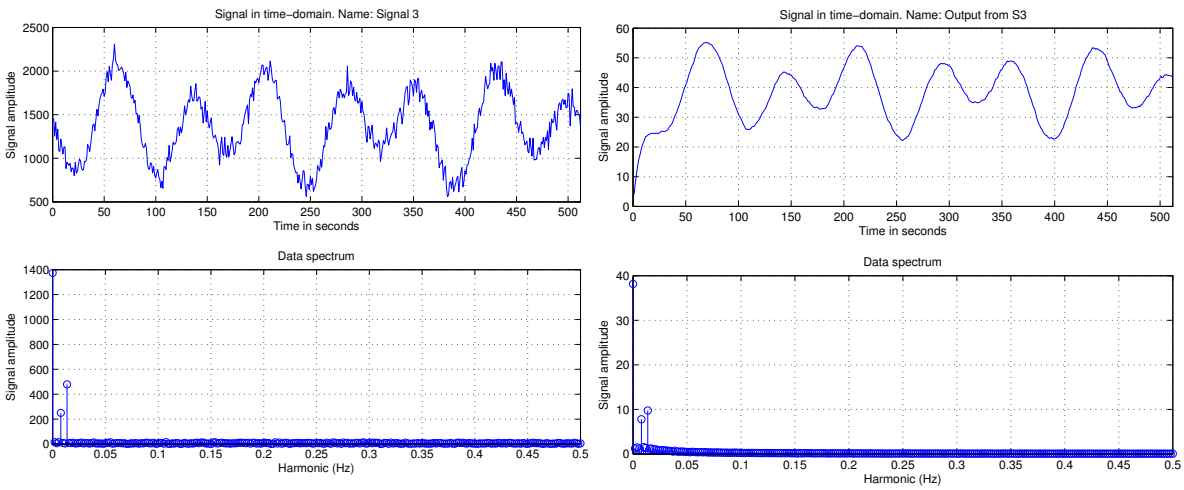


Figura 54 – Sinal 3.

A Fig. 56 possui o resultado correspondente ao **Sinal 5**. Como os dois harmônicos apresentam baixas frequências para o sistema, elas ficaram presentes na saída. Há de se mencionar que boa parte do ruído é retirado pelo sistema, justamente por essa propriedade. As frequências altas são mais filtradas do que aquelas baixas, conforme é inspecionado no diagrama de Bode. Neste estudo evidenciou-se algumas frequências com o intuito de ressaltar esse comportamento.

(a) Entrada. (b) Saída.

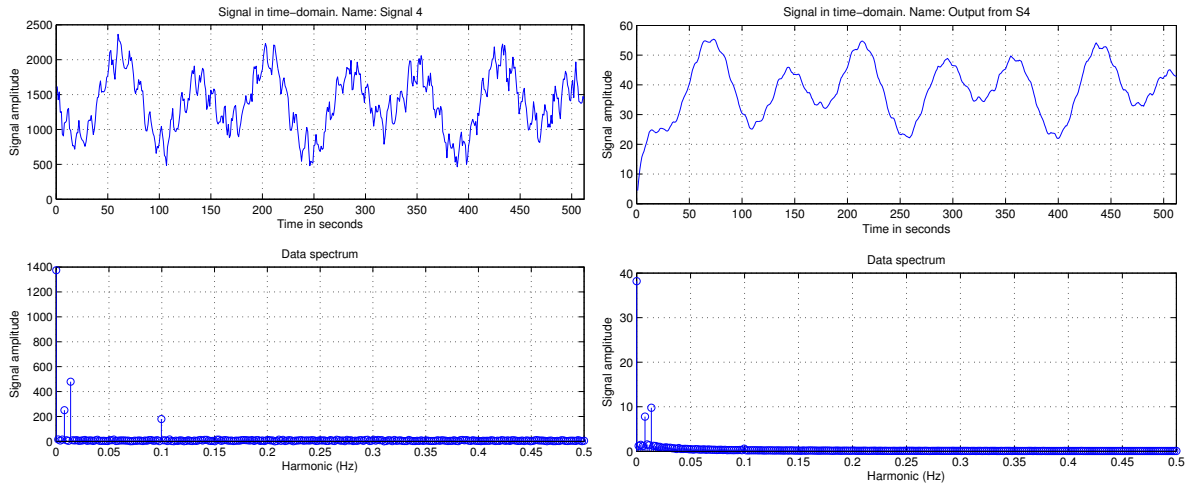


Figura 55 – Sinal 4.

(a) Entrada. (b) Saída.

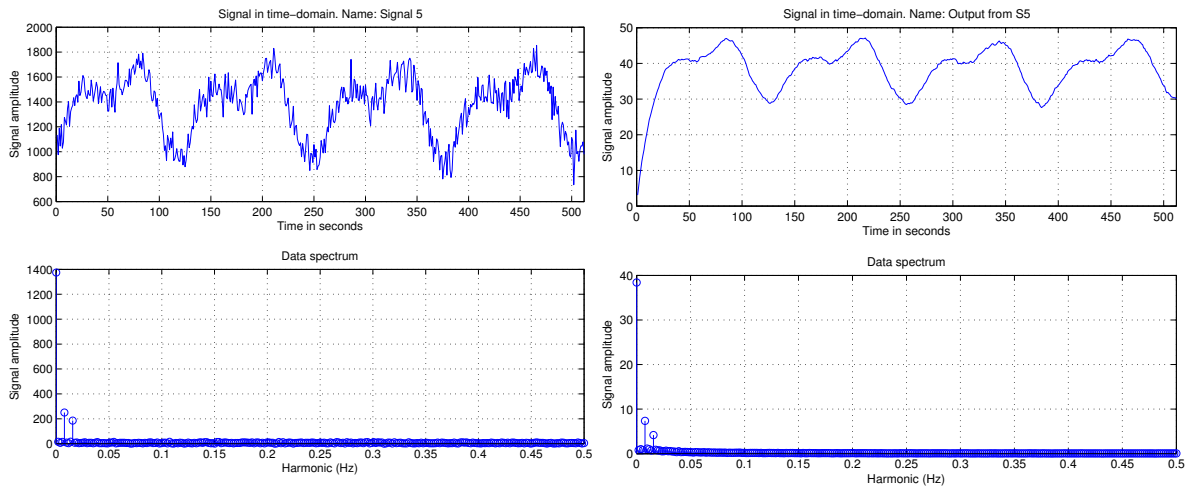


Figura 56 – Sinal 5.

6.6 Geração de um sinal com rajada

Analisado o comportamento da resposta do sistema quando excitado por um trem de pulso e também como a atenuação de frequências ocorre ao observar o diagrama de Bode, é possível estipular como uma carga de trabalho com características de rajada pode ser especificada. A rajada não pode ser tão curta de modo que ela seja absorvida pela inércia do sistema e também não tão longa de modo que ela se apresente como um degrau. Um ponto importante a ser mencionado é que nesta abordagem é considerado a dinâmica do sistema ao produzir a carga de trabalho.

Como já foi ilustrado pela Fig. 48, o trem de pulso possui duas variáveis características: a amplitude (A) e a duração (D). Para este estudo essas variáveis terão valores aleatórios dentro de faixas de valores significantes, tendo como parâmetro o sistema em estudo. O tempo de

assentamento (*settling time*) do sistema é 39 s (Eq. 6.2). Portanto, D deve possuir pelo menos tamanho 40 s de forma que o sistema atinja diferentes patamares de estacionariedade. Também é importante que a rajada ocorra dentro da região de operação ($U \in [40\%, 60\%]$), ou seja, A deve estar entre 1371 e 2158.

Um sinal do tipo *pseudo-random binary sequency* (PRBS) apresenta uma série que se assemelha com o trem de pulso utilizado nesta tese, porém com duração (D) variável. O comprimento do sinal e a quantidade de mudanças de magnitude dependem do parâmetro n passado para a função que gera a sequência. No caso do PRBS esse valor corresponderá a

$$l = 2^n - 1 \tag{6.8}$$

e

$$m = \frac{2^n}{2}, \tag{6.9}$$

em que l é o comprimento do sinal, m é a quantidade de variações que ocorrerá no sinal e n a quantidade de *bits* utilizados para gerar a sequência. Isso significa que, caso n seja 4 (quatro), o sinal produzirá uma entrada que corresponderá a uma excitação de duração 31 s e a quantidade de vezes em que o sinal alternará será 16.

A amplitude pode ser alterada após o sinal PRBS ter sido gerado. Uma estratégia é gerar m valores dentro da faixa de operação e para cada mudança presente no sinal, atribuir determinada amplitude. Com essa alteração o sinal resultante teria as características aleatórias de mudanças presentes no PRBS, porém não limitado aos valores extremos. Uma segunda alteração foi feita de forma a garantir a duração mínima de cada patamar. A solução foi realizada ao replicar os valores contidos em cada posição. Por exemplo, seja o fragmento de um vetor hipotético $v = \{10, 10, 20, 15, 15\}$, cujo o tamanho é 5; caso seja necessário dobrar os valores desse vetor, o resultado seria $v_2 = \{10, 10, 10, 10, 20, 20, 15, 15, 15, 15\}$, de tamanho 10; ao triplicar, $v_3 = \{10, 10, 10, 10, 10, 10, 20, 20, 20, 15, 15, 15, 15, 15, 15\}$, de tamanho 15. O Código 10 apresenta a implementação.

Código-fonte 10: Função que gera um sinal com rajada, codificado em Matlab.

```
1 function sig = randomsig(nbits,low,high,nrepeat,rseed)
2   sig=idinput(2^nbits-1, 'prbs',[0 1], [low high]);
3   rng(rseed);
4   amps=randi([low high],1,(2^nbits)/2);
5
6   j=1;
7   for i = 1:length(sig)
8     oldsig=sig(i);
9     sig(i)=amps(j);
```

```
    if i < length(sig)
      if oldsig ~= sig(i+1)
        j=j+1;
      end
    end
  end
  sig=[sig;sig(end)];

  sig=reshape(repmat(sig',nrepeat,1),1,[]);
end
```

O primeiro passo é gerar a sinal PRBS (`idinput`), seguido das amplitudes (`amps`). Para cada valor no sinal altera-se sua amplitude, caso o próximo sinal tenha amplitude diferente, altera-se o índice do vetor `amps`. Como o sinal gerado possui tamanho $2^n - 1$, o último valor é adicionado uma segunda vez para que o tamanho seja múltiplo de 2. Por fim, replica-se cada valor do vetor por `nrepeat` vezes. Deve ser mencionado que a quantidade de valores gerados deve ser em potência de 2 de forma que seja mais adequado ao algoritmo FFT. A Fig. 57 apresenta um sinal que foi utilizado neste estudo. A geração desse sinal foi realizada invocando a função desta forma: `randomsig(5,1350,2150,2^6,101233)`, o que resultada em um sinal PRBS de tamanho 2^5, repetido 2^6, portanto o sinal resultante possui 16 patamares diferentes de tamanho $2^{11} = 2048$ s, em que os patamares então na faixa de valores entre 1350 e 2150. O sinal pode ser reproduzido, utilizando-se a mesma semente (101233).

Ao sinal de rajada gerado foi adicionado um ruído oriundo do próprio sistema. Com as mesmas configurações mencionadas no início deste capítulo, foi realizado um experimento de 1 hora de duração, em que retirou-se os dados de *warm-up* no início e no final. Esse experimento foi executado apenas uma vez. Um trecho central de mesmo tamanho do sinal com rajada gerado foi extraído. A intenção foi produzir dados reais obtidos do sistema e que servissem como base para extração do ruído. A quantidade de requisições ficou estacionada em 1378 req/s. A Fig. 58 apresenta esse trecho.

A partir da FFT dos sinais da Fig. 57 e 58, foi mantida o frequência fundamental do sinal de rajada e as demais frequências foram somadas. O Código 11 apresenta essa operação. O resultado produzido é um sinal com rajadas e ruidoso, mostrado na Fig. 59. O sinal com rajadas e ruidoso foi passado ao sistema, e a saída correspondente é mostrada na Fig. 60.

Código-fonte 11: Função que combina a frequência de dois sinais.

```
function newsig = mixsig(a,b)
  ffta=fft(a);
  fftb=fft(b);
  newsig=ifft([ffta(1) (ffta(2:end)+fftb(2:end))]);
end
```

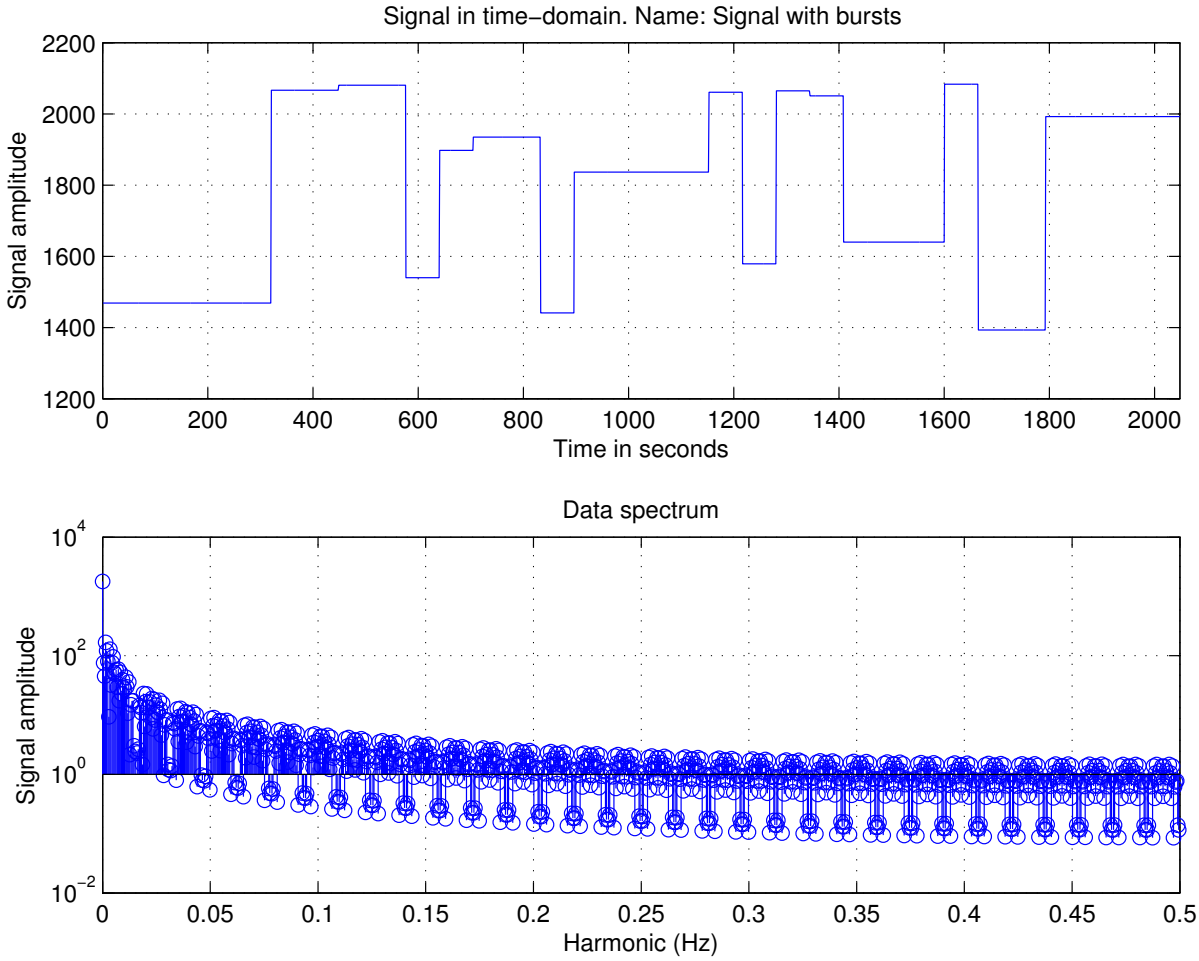


Figura 57 – Exemplo de sinal que representa uma rajada para o sistema.

Dos resultados obtidos destaca-se:

- **Generalização do modelo**: pode-se simular o comportamento do sistema em diferentes cargas de trabalho;
- **Parâmetro para geração de rajadas**: ao observar o modelo dinâmico que representa o sistema é possível especificar critérios quantitativos sobre como a carga de trabalho será gerada — neste caso observou-se o tempo de assentamento e o diagrama de Bode;
- **O sistema atua como um filtro**: como pode ser observado na Fig. 60 os harmônicos de menor frequências são mantidos e ao aumentar a frequência a potência dos harmônicos é diminuída, conforme previsto pelo diagrama de Bode.

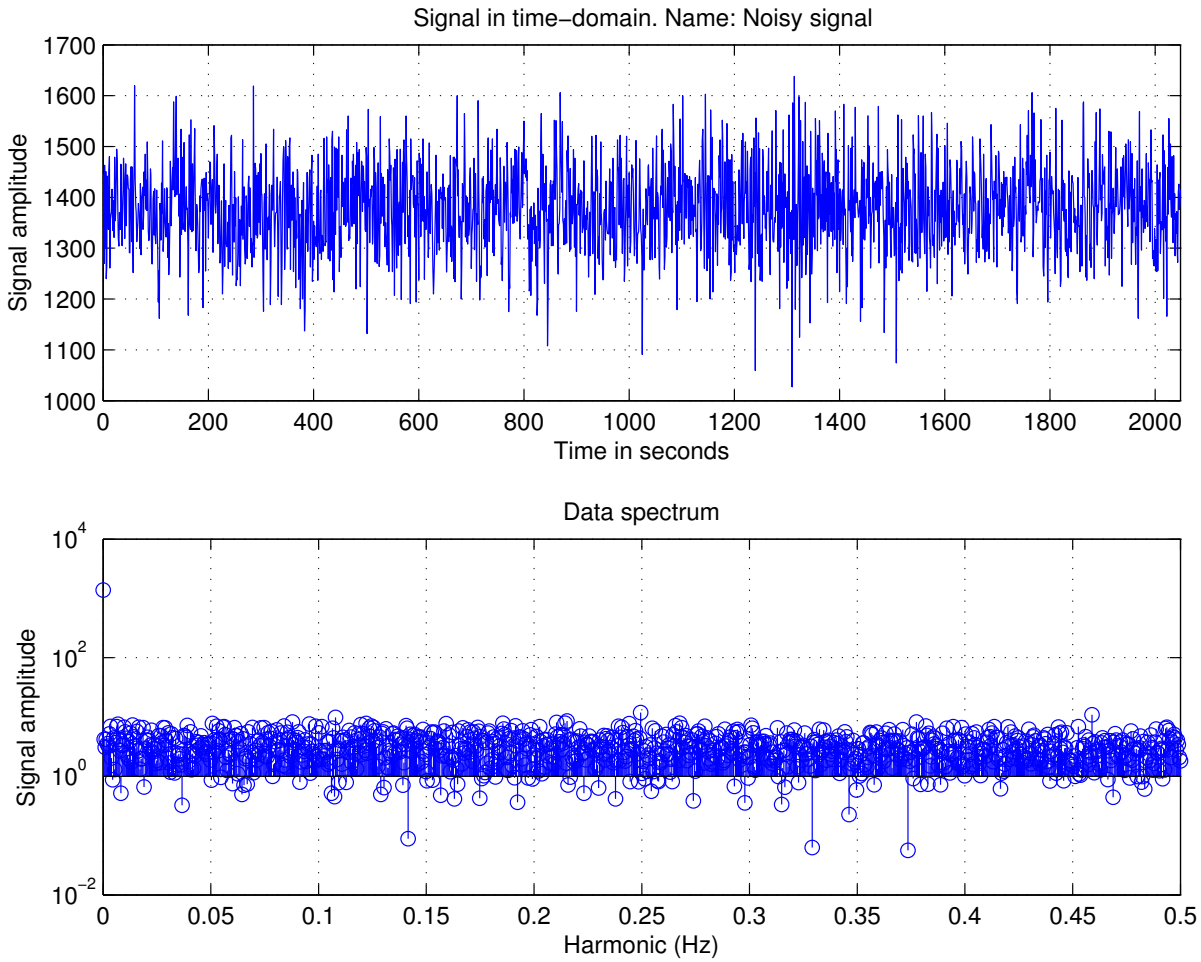


Figura 58 – Trecho de execução que contém o ruído do sinal. A amplitude do sinal corresponde à quantidade de requisições por segundo.

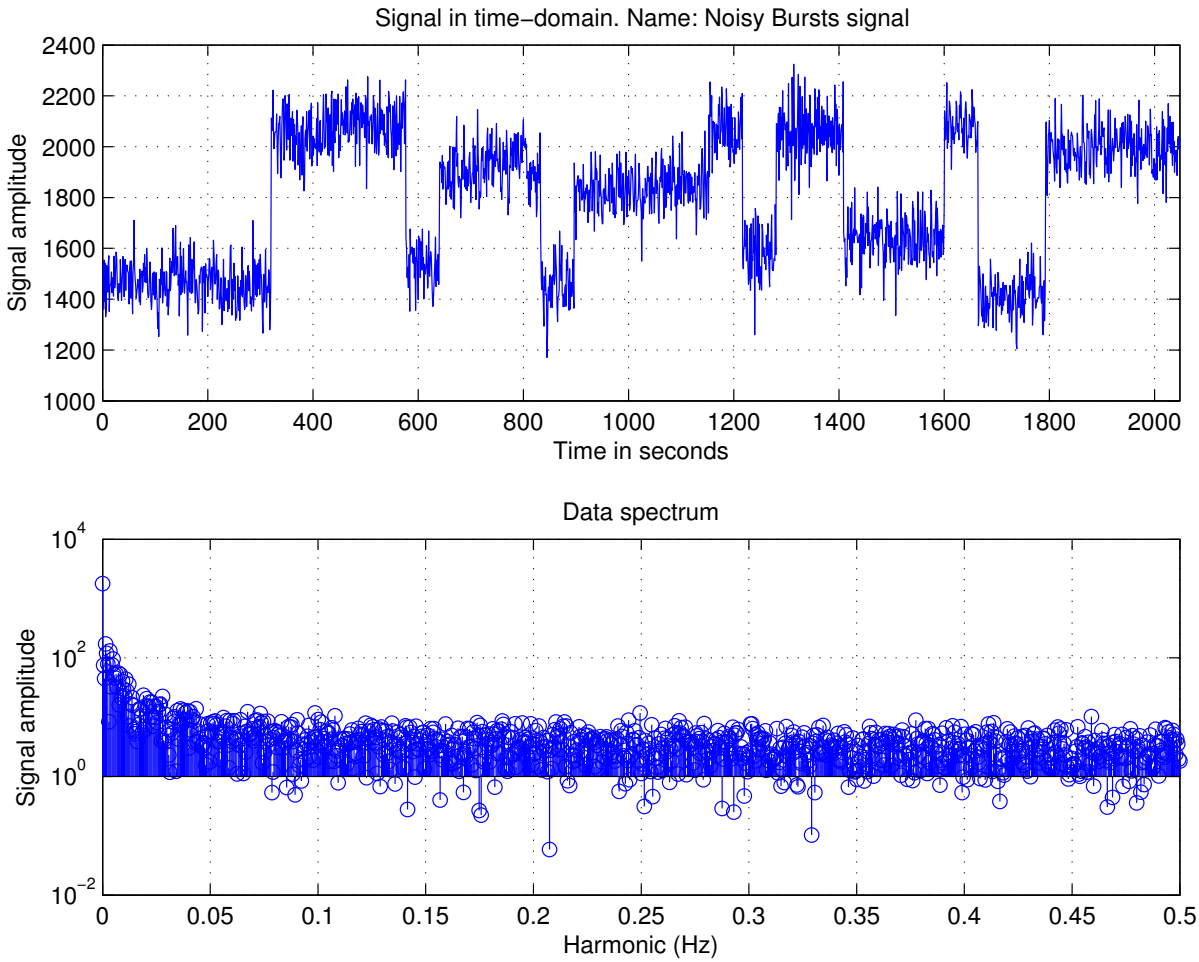


Figura 59 – Combinação entre a rajada e o ruído do sistema.

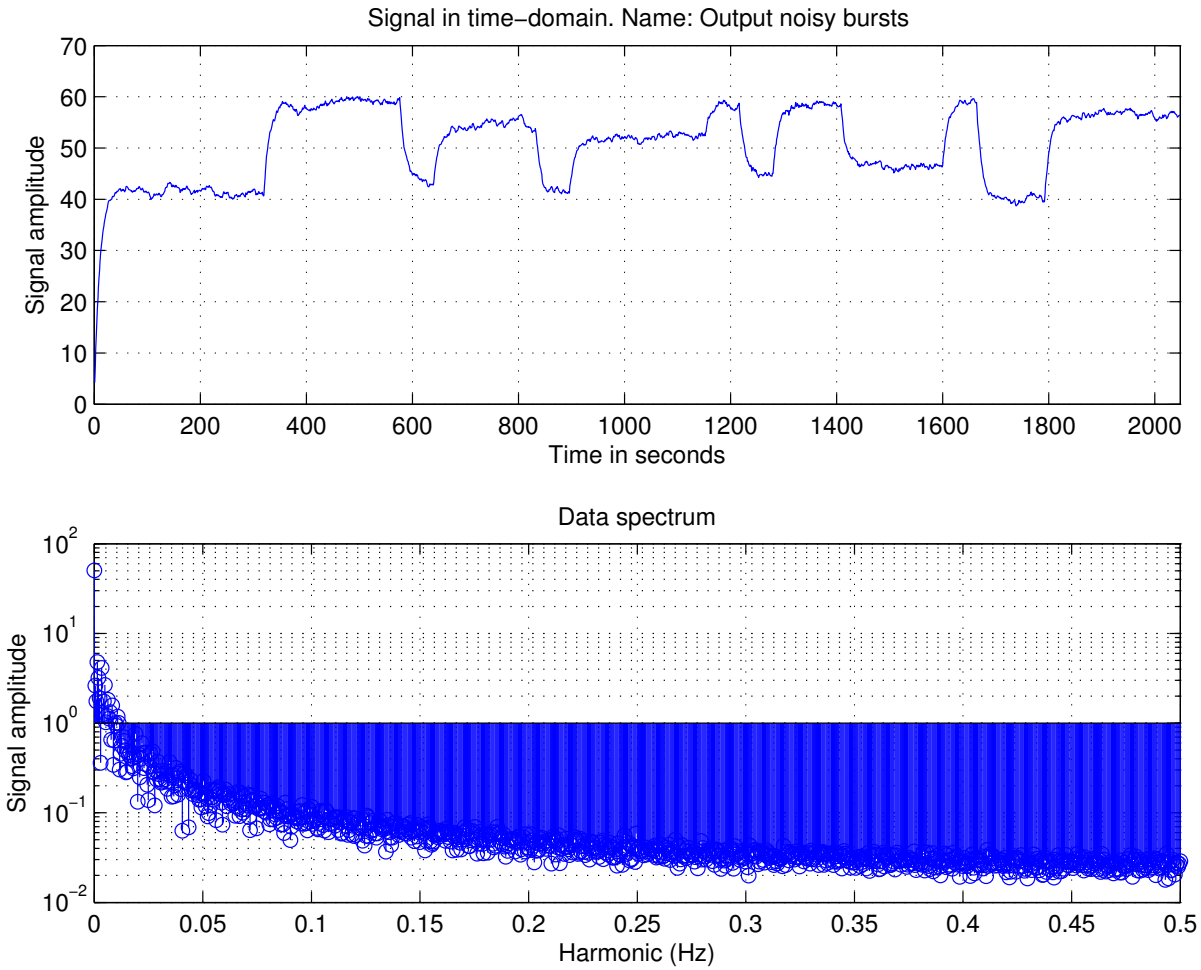


Figura 60 – Saída produzida pelo sistema. Observa-se que o sistema atua como um filtro.

CAPÍTULO

7

CARACTERIZAÇÃO DE CARGA

Este capítulo apresenta alguns resultados complementares aos experimentos realizados. São incluídos no relato para demonstrar que a análise de resposta em frequência pode ser uma ferramenta para caracterização de carga de trabalho do ponto de vista de despenho dinâmico. A noção explorada nessa abordagem é que, se duas cargas de trabalho diferem apenas nas componentes de frequências que são muito atenuadas, então, para efeitos práticos, elas podem ser consideradas similares. Para isso, o estudo de caso faz uso do diagrama espectral dos sinais e os compara com o diagrama de bode. O diagrama espectral possui no eixo das abscissas as frequências presentes no sinal, e no eixo das coordenadas, as amplitudes das respectivas componentes. O diagrama de Bode auxilia a avaliar qual região do espectro do sinal de carga de trabalho impacta o desempenho, e qual região é absorvida pela sua dinâmica.

7.1 Estudo de caso

A Função de Transferência é do sistema foi identificada com os parâmetros conforme indicados na Eq. 7.1. O ganho DC é $0,0283$ e a reposta em frequência é ilustrada conforme Diagrama de Bode da Fig. 61.

$$G(z) = \frac{b}{1+az^{-1}} = \frac{0.00273}{1-0.9037z^{-1}}. \tag{7.1}$$

A finalidade deste estudo é verificar como uma carga de trabalho influencia no desempenho do sistema, dependendo da sua característica espectral e da resposta em frequência do sistema. Como foi observado no capítulo de Resposta em Frequência (Cap. 6), a Função de Transferência é capaz de representar o comportamento do sistema em estudo. Isso significa que pode-se utilizar o Matlab para gerar experimentos cujos os resultados são tão significativos como aqueles obtidos a partir do sistema real. Nesse sentido, será gerado um sinal oriundo de um gerador de números pseudo-aleatório; o tamanho desse sinal será de 1 milhão de segundos, ou

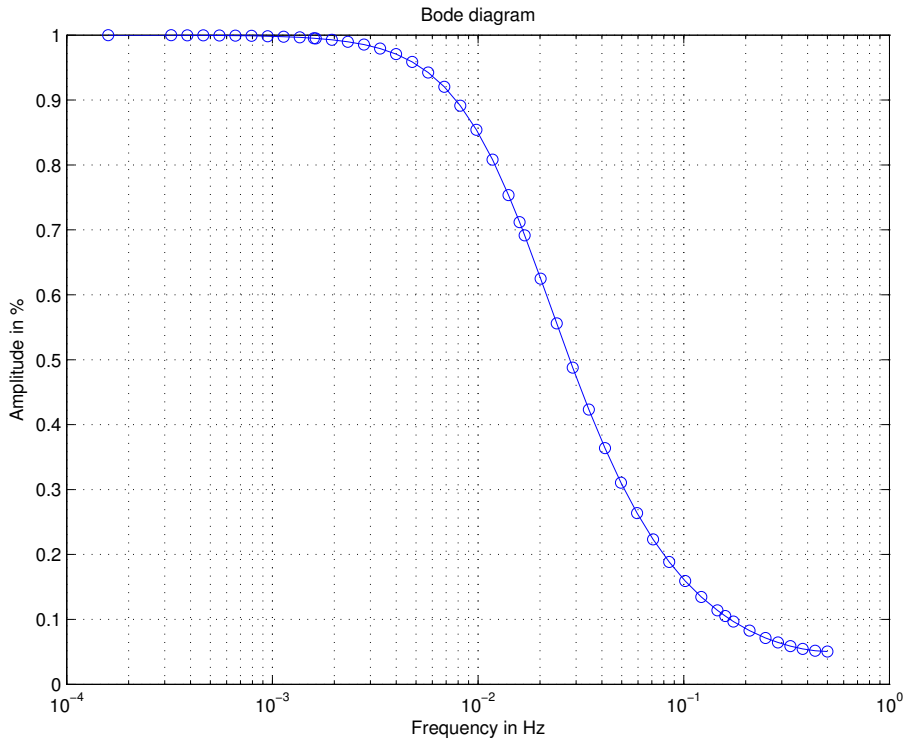


Figura 61 – Diagrama de Bode do sistema em estudo.

seja, uma simulação de 291 horas. Esse sinal será passado para a função de transferência e seu resultado será observado. Uma característica desse sinal é que ele é estocástico e estacionário.

Antes de iniciar o estudo é possível observar que determinadas frequências contidas no sinal de entrada simplesmente serão reduzidas (observável no Diagrama de Bode — Fig. 61). Isto é, ao filtrar o sinal de entrada por um filtro com as mesmas propriedades dinâmicas do sistema, o resultado produzido tende a ser semelhante ou se aproximar bastante. A Função de Transferência transforma taxa de requisições por segundo em taxa de utilização de CPU. Para transformar a Função de Transferência em estudo (Eq. 7.1), basta equacionar seu ganho para 1.

Demonstração. [Transformar uma Função de Transferência em um filtro.] Suponha um sistema Linear e Invariante no Tempo, s, cujo o ganho DC é $0,1$, e um sinal de entrada $f(t)$, tem-se que

$$f(t) = \{100, 100, 100, 100\} \tag{7.2}$$

$$s(f(t)) = \{10, 10, 10, 10\} = h(t). \tag{7.3}$$

Assume-se que o sistema está em regime operacional estacionário. Pode-se escrever, portanto, $S(f(t)) = h(t)$. Ao passar o sinal $h(t)$ pelo sistema, há a transformação novamente, ou seja, haverá um ganho de $0,1$ aplicado ao sinal de entrada. Assumindo o sistema em regime estacionário, tem-se que

$$s(h(t)) = \{1,1,1,1\} = g(t). \tag{7.4}$$

Percebe-se notoriamente que $h(t) \neq g(t)$. No entanto, ao multiplicar $g(t)$ pelo ganho, obtém-se o sinal original, $h(t)$. Dessa forma, deseja-se estabelecer a equação

$$dc \cdot \alpha = 1 \tag{7.5}$$

$$\alpha = \frac{1}{dc}. \tag{7.6}$$

Em que dc é o ganho DC da Função de Transferência e α a incógnita a ser encontrada. Ou seja, o filtro será realizado por

$$S_{filtro}(z) = \alpha S(z) = \frac{S(z)}{dc}, \tag{7.7}$$

Observa-se que as funções estão no domínio $\mathscr{Z}$.

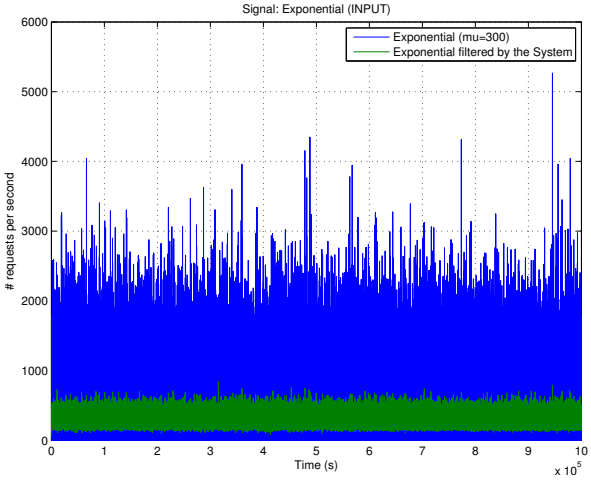
Isso faria com que $g(t)$, no domínio do tempo, fosse multiplicado pelo inverso do ganho, ou seja, 10. Portanto, resultaria em $h(t) = g(t)$. □

A conclusão da demonstração é que ao dividir a função de transferência por seu ganho DC, obtém-se uma função com ganho DC 1. Isso é importante pois não haverá alterações na magnitude do sinal, porém as frequências contidas no sinal antes de ser passado pelo filtro serão removidas, conforme estabelecido pelo Diagrama de Bode do sistema. Considerando a Eq. 7.1 como o sistema e $0,0283$ como ganho DC, tem-se

$$G_{filtro}(z) = \frac{G(z)}{0,0283} = \frac{0.00273}{0.02835 - 0.02562z^{-1}}. \tag{7.8}$$

Feita a especificação do filtro, geraram-se dois sinais, um exponencial com média 300 e outro filtrado, doravante referenciados convenientemente como exponencial e filtrado. A Fig. 62 apresenta os dois sinal de entrada (Fig. 61a) e os respectivos sinais de saída (Fig. 61b). O sinal exponencial possui média $299,82$ e desvio padrão $299,90$, enquanto o filtrado $299,81$ e $67,33$ para média e desvio padrão respectivamente. Os sinais de saída para o exponencial possui média $8,49$ e desvio padrão de $1,90$, ao passo que para a saída do filtrado $8,49$ de média e $1,35$ para desvio padrão. Ao dividir a média dos sinais de saída pela média dos sinais de entrada, obtém-se o ganho de $0,02835$, o que é condizente com o sistema. Ao dividir o desvio padrão dos sinais de entrada, obtém-se $0,22452$, fazendo a mesma operação para os sinais de saída, tem-se como resultado $0,70756$. Isso significa que há o sinal filtrado possui uma dispersão de $22,45\%$ do sinal exponencial original, porém na saída o resultado dessa razão aumenta para $70,75\%$, ou seja, boa parte da dispersão é filtrada pelo sistema o que faz com que os sinais de saída sejam semelhantes.

(a) Sinais de entrada: requisições por segundo.



(b) Sinais de saída: taxa de utilização.

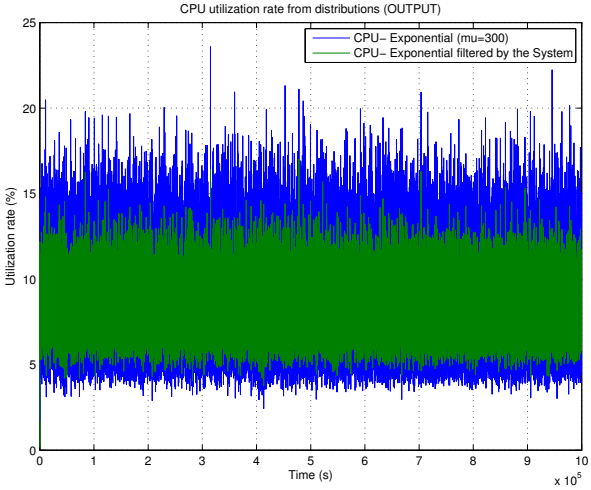


Figura 62 – Exponencial com média 300 e a mesma exponencial filtrada pelo sistema.

É possível perceber que, é apresentado um caso em que duas cargas de trabalho diferentes apresentam resultados semelhantes. Pode-se aproximar o sinal de entrada filtrado por uma normal. Ou seja, caso ao passar dois sinais com propriedades estatísticas semelhantes, qual resultado seria obtido? A Figura 63 apresenta essa comparação. Foi gerado um sinal com 1 milhão de amostras oriundo de um gerador de números pseudo-aleatórios com distribuição normal com média 300 e dispersão 67; esse sinal será referenciado doravante como normal. Os sinais de entrada filtrado e normal (Fig. 62a) são passados pelo sistema o que gera o sinal de saída (Fig. 62b). O sinal normal possui média 299.96 e desvio padrão 66.97. Para o sinal normal, ao dividir a média de sua saída pela média do respectivo sinal de entrada, obtém-se o ganho de $0,02835$, o que é condizente com o sistema. Ao dividir a dispersão dos sinais de entrada (dispersão do normal pelo filtrado), obtém-se $0,99361$, enquanto que, para a mesma operação considerando os sinais de saída, tem-se $0,31719$. O que significa que há uma semelhança estatística entre os sinais de entrada, porém os de saída são diferentes. O que leva à evidência de que duas cargas de trabalho semelhante apresentam resultados diferentes. O que é contraditório ao resultados obtidos até o momento.

Para observar a semelhança dos sinais filtrado e normal e a diferença apresentada nos resultados, calculou-se a Função Densiddade de Probabilidade (PDF) com a função `ksdensity` disponível no Matlab. Essa função recebe uma série de números e estima como seria a PDF da série. A Fig. 64 apresenta essa estimativa. Pode-se observar uma boa aproximação para os sinais de entrada (Fig. 63a), porém uma distribuição diferente para a saída (Fig. 63b), muito embora a média seja a aproximadamente a mesma. Até o momento, tem-se que para duas distribuições diferentes (Fig. 62) os resultados produzidos foram semelhantes, e que para duas distribuições semelhantes (Fig. 63) os resultados foram diferentes. Ou seja, o resultado sugere que as estatísticas de média e dispersão da distribuição, nesse caso, influencia pouco nesse sistema. A análise da resposta em frequência que considera tanto o sinal de entrada quanto o

(a) Sinais de entrada: requisições por segundo. (b) Sinais de saída: taxa de utilização.

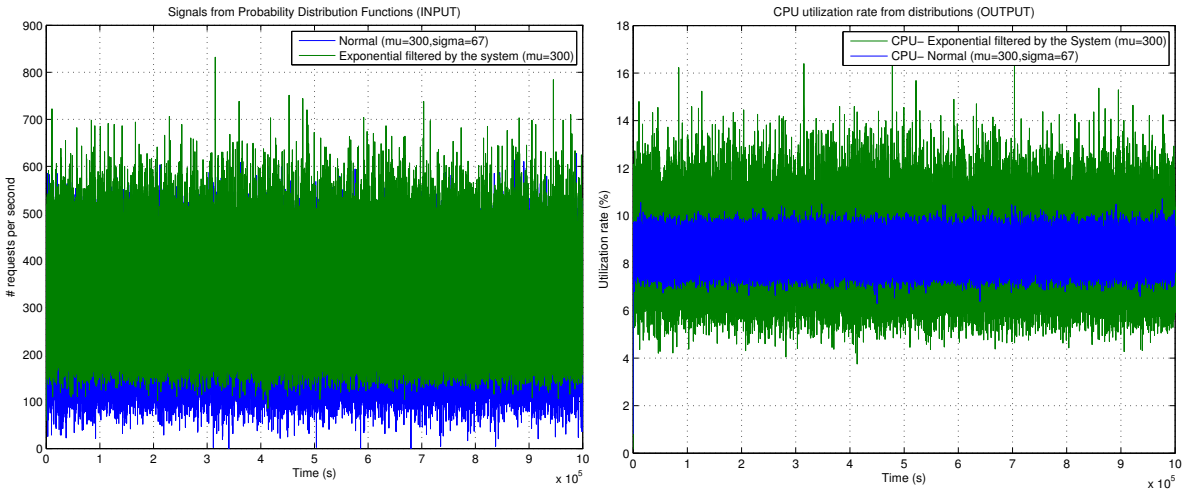


Figura 63 – Exponencial filtrada pelo sistema e normal com média 300 e dispersão 65.

comportamento do sistema pode revelar o ponto que explica tais resultados.

(a) Sinais de entrada: requisições por segundo. (b) Sinais de saída: taxa de utilização.

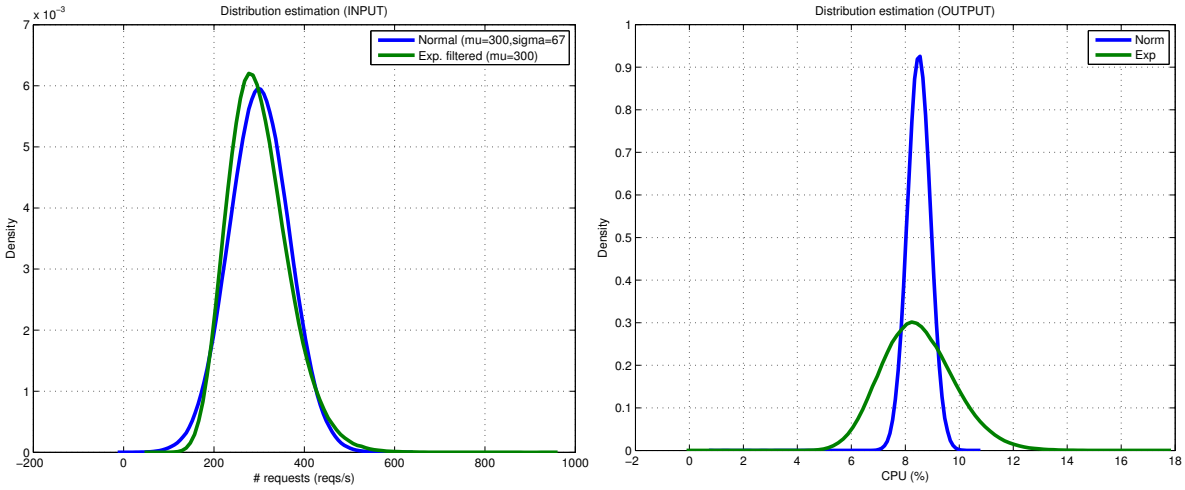


Figura 64 – Função densidade de probabilidade estimadas a partir dos sinais gerados.

Os três sinais considerados até o momento (exponencial, filtrado e normal) foram passados para o algoritmo FFT (implementado pela função `fft` do Matlab). O resultado é um vetor de números complexos de mesmo tamanho do sinal. O tamanho do sinal é de aproximadamente 1 milhão de valores (mais precisamente 2^{20}). Desse vetor resultante, obteve-se o valor correspondente aos harmônicos que geram o sinal no domínio do tempo. A Fig. 65 apresenta a FFT do sinal exponencial, a Fig. 66 a FFT do sinal filtrado, e a Fig. 67 a FFT do sinal normal. Pode-se observar (Fig. 65) que o sinal exponencial é composto pela média (300), correspondente ao harmônico fundamental e um ruído branco, isto é, uma grande quantidade de harmônicos distribuídos aproximadamente de forma isonômica na faixa de frequência. Ao passar o sinal exponencial pelo filtro especificado na Eq. 7.8, parte das das frequências são atenuadas (Fig. 66),

de modo que as frequências mais altas (próximas a $0,5$ Hz) tendem a zero, conforme previsto pelo diagrama de bode (Fig. 61). Há uma concentração de harmônico de baixa ordem. A FFT do sinal normal apresenta o harmônico fundamental e um ruído branco (Fig. 67). A FFT do sinal normal difere dos outros dois sinais. É diferente da FFT do sinal exponencial pois as frequências do sinal normal possuem amplitude menor, ao passo que essas são maiores no sinal exponencial. Essas diferenças fazem com que os dois sinais tenham a mesma média, porém com probabilidade de distribuição diferentes. A exponencial com harmônicos de maior amplitude propiciam picos maiores no domínio do tempo, o que é característico de sua calda. A FFT do sinal normal difere da FFT do sinal filtrado pois possui uma concentração de harmônicos maiores e de maior amplitude em baixas frequências. Esse fato impacta na saída do sistema, pois baixas frequências significa uma maior excursão do sinal resultante, e isso foi observado nos resultados, uma vez que a dispersão do sinal de saída do filtrado foi maior que o respectivo do normal.

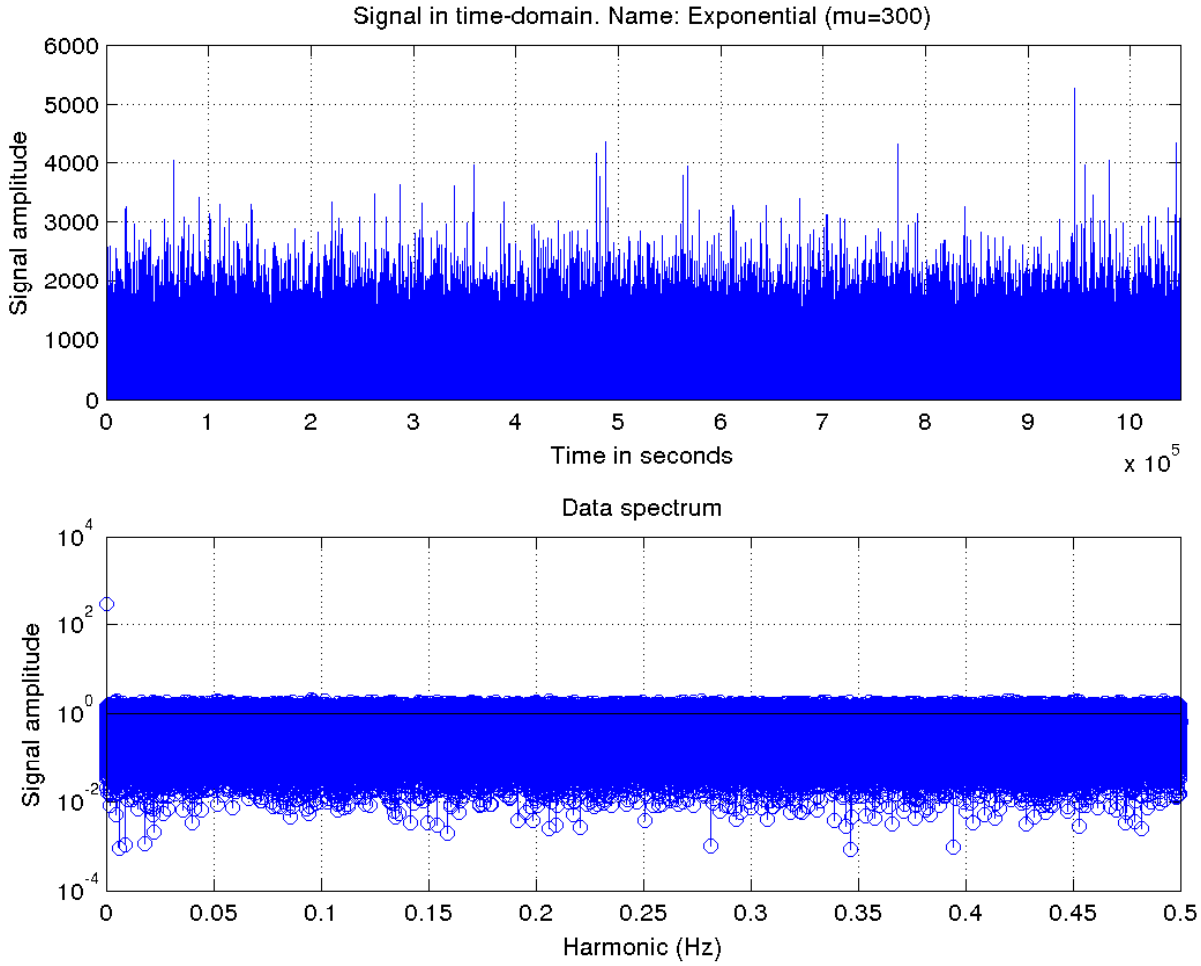


Figura 65 – Função densidade de probabilidade estimadas a partir dos sinais gerados para a exponencial com média 300.

É interessante observar como é a distribuição estatística dos harmônicos resultantes da FFT. A Fig. 68 apresenta o resultado obtido pela função `ksdensity`. Observa-se que os sinais exponencial e filtrado possuem uma calda longa, ou seja, há a presença de amplitudes cuja os valores são maiores. O sinal exponencial tem uma boa distribuição na faixa de 0 a

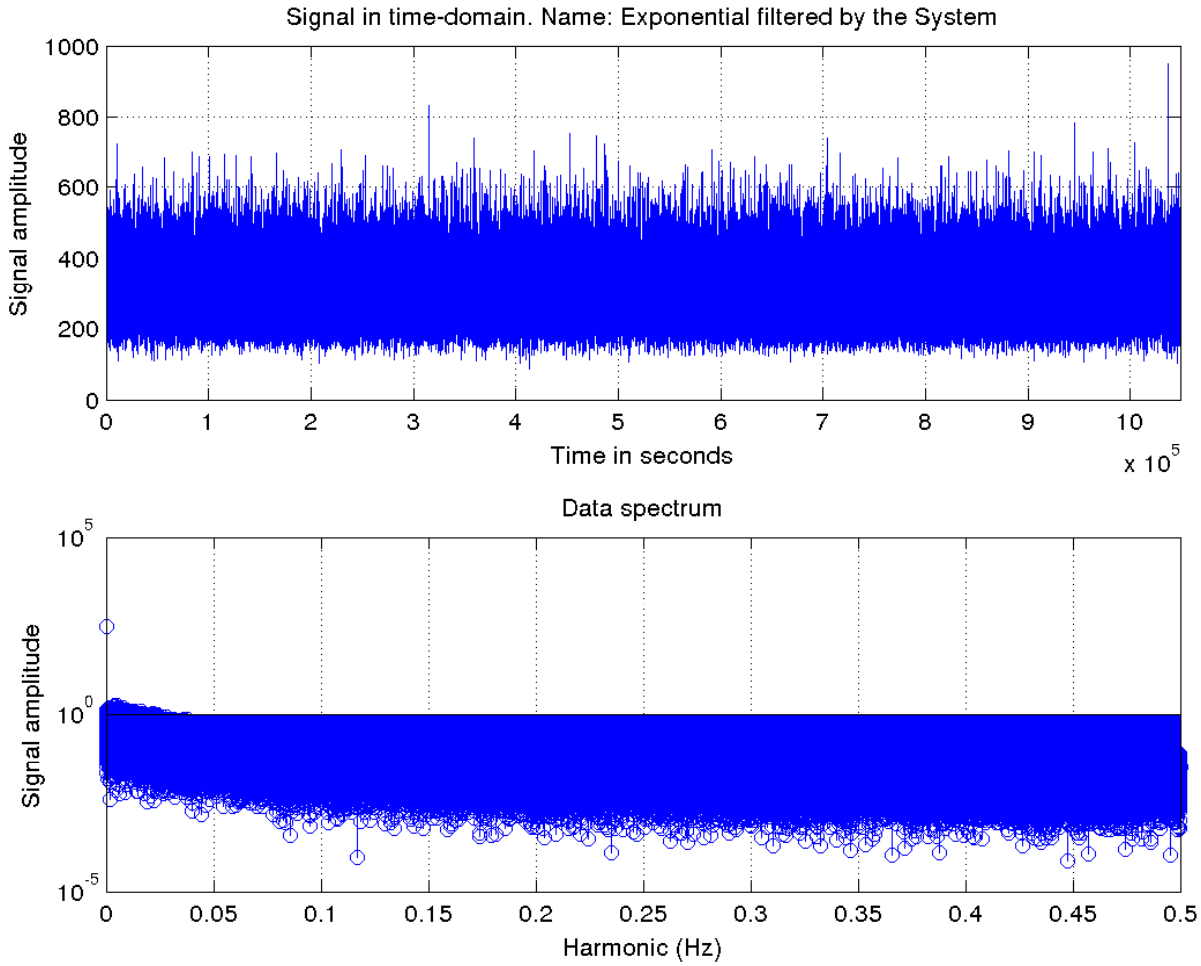


Figura 66 – Função densidade de probabilidade estimadas a partir dos sinais gerados para a exponencial filtrada pelo sistema.

aproximadamente $2,1$. Por outro lado, o sinal filtrado boa parte dos valores tendem a zero. Para o sinal normal, a calda é menor e há um agrupamento de magnitudes com valores menores. A distribuição dos valores de vai de 0 a aproximadamente $0,5$.

A conclusão desse estudo é que o fato de haver casos em que o sistema é excitado por (1) distribuições diferentes que geram resultados semelhantes e (2) distribuições semelhantes que geram resultados diferentes, depende não somente da distribuição, mas como o sistema absorve a carga de trabalho imposta a ele. Observou-se que a carga de trabalho exponencial possui ruído branco com valores significantes para amplitude, o que propicia uma variância maior; a carga de trabalho filtrada, mantém as frequências que impactam na saída do sistema. Se comparada à exponencial, as altas frequências, que no domínio do tempo traduzem-se a seus valores de picos, são absorvidos pela dinâmica do sistema. Em outras palavras, os picos têm magnitude considerável, porém duração despresível. O sinal normal, muito embora, semelhante estatisticamente ao filtrado, possui ruído branco com valores menores, o que faz com que a dispersão seja menor no domínio do tempo. Esses resultados evidenciam a importância da abordagem da modelagem de sistemas computacionais através de modelos dinâmicos e destaca

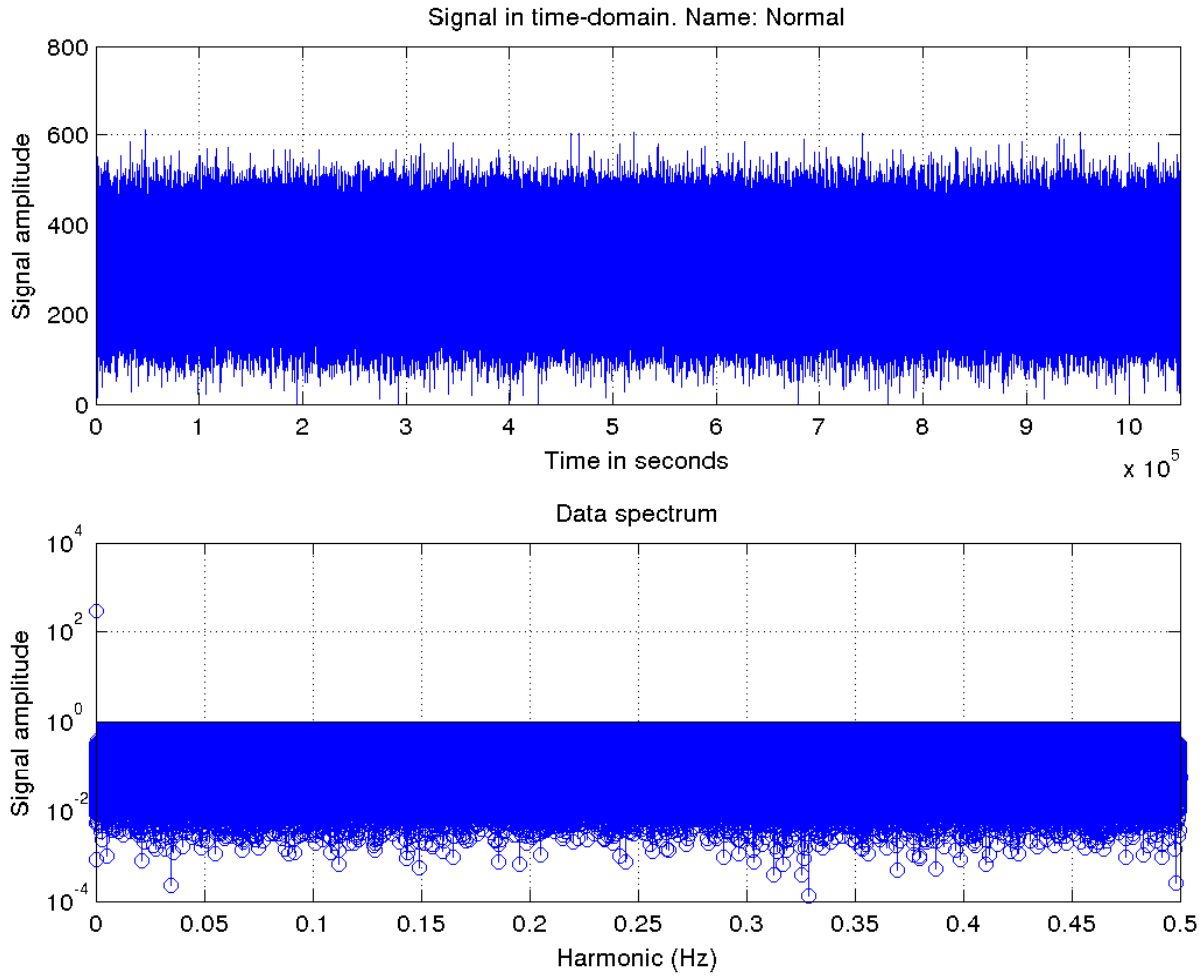


Figura 67 – Função densidade de probabilidade estimadas a partir dos sinais gerados para a normal com média 300 e dispoersão 67.

o uso da análise em frequência, pelos diagrama de Bode e FFT, para obter comportamentos importantes a respeito do desempenho do sistema.

7.2 Traços de sistemas reais

Esta seção apresenta alguns estudos complementares, semelhantes aos anteriores, realizados sobre cargas reais obtidos de *logs* publicamente acessíveis.

7.2.1 Recuperação da informação

Os traços de execução utilizados neste estudo foram obtidos da lista disponível neste site *The Internet Traffic Archive*[1].

Escolheu-se três traços de interesse:

[1] ITA: <http://ita.ee.lbl.gov/html/traces.html>

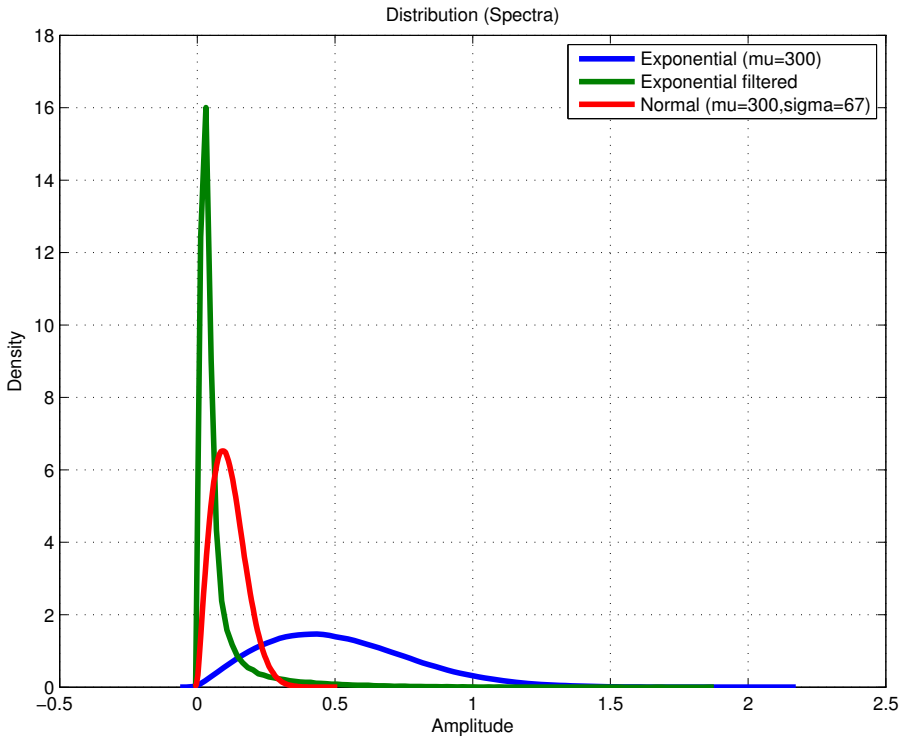


Figura 68 – Função densidade de probabilidade estimadas a partir dos harmônicos presentes nos *spectra* dos sinais de entrada exponencial (média 300), exponencial filtrada pelo sistema e normal (média 300 e dispersão 65).

- Clarknet[2];
- NASA[3];
- Copa 98[4].

Todos esses arquivos estão disponibilizados em arquivos compactados (extensão `.gz`).

7.2.1.1 *Clarknet e NASA*

São compostos de dois arquivos texto cada. Um arquivo é formado por um conjunto de registro de requisição. O código-fonte 12 apresenta uma linha de exemplo.

Código-fonte 12: Uma entrada do arquivo de traço.

```
1 204.249.225.59 - - [28/Aug/1995:00:00:34 -0400] "GET /pub/rmharris/
    catalogs/dawsocat/intro.html HTTP/1.0" 200 3542
```

O formato da entrada do arquivo de traço é:

```
Host - - [timestamp] ‘‘comando http’’ <código retorno> <n.o de bytes na resposta>
```

[2] Clarknet: <http://ita.ee.lbl.gov/html/contrib/ClarkNet-HTTP.html>
[3] NASA: <http://ita.ee.lbl.gov/html/contrib/NASA-HTTP.html>
[4] Copa 98: <http://ita.ee.lbl.gov/html/contrib/WorldCup.html>

É de interesse apenas o *timestamp*[5] (registro de tempo) de cada requisição. A resolução de amostragem é de segundos. Ou seja, é possível obter a informação que em determinado segundo 5 requisições chegaram ao servidor HTTP, mas não o exato momento em que cada uma chegou.

Para extrair as informação desejada utilizou-se o *shell* Bash do Linux e foi feito através dos seguintes passos:

- Extrair arquivos com timestamps:
  - Descompactar o arquivo (`zcat arquivo.gz`);
  - Trocar o caractere `]` do timestamp por `[` (`sed 's/]/[/'`);
  - Obter o segundo campo do arquivo, considerando caractere `[` como separador de campo (`cut -d [ -f 2`);
  - Como a data está no formato `28/Aug/1995:00:00:34 -0400`, foi necessário trocar `/` e `:` (apenas a primeira incidência) por espaço, de forma que fosse possível utilizar a porção de texto resultante nas funções `strptime(3)` e `mktime(3)`. A intenção é a de obter um número inteiro correspondente ao `epoch` do Unix. (`sed 's// /g;s/:/ /'`);
  - Salvar o conteúdo intermediário em um arquivo. (`> intermediate.ts`)
- Processar o arquivo intermediário:
  - Implementado um programa em C que lê da entrada padrão uma `string` no formato `28 Aug 1995 00:00:34 -0400` e produz como saída o número correspondente `809578834`. (`cat intermediate.ts | ./d2u`);
  - A saída produzida é ordenada numericamente. (`sort -g`)
  - O comando `uniq` é utilizazado com o parâmetro `-c` de forma que gere a quantidade de vezes que uma linha se repete, cada linha repetida corresponde a uma requisição. (`uniq -c`)
  - O comando `uniq` acrescenta muitos espaços em branco no início da linha, portanto é necessário removê-los. Cada saída possui dois campos separados por um espaço em branco, a quantidade em que uma linha se repetiu e a linha em si. O espaço em branco separador de campos é substituído por vírgula para ficar compatível com o formato CSV.

    ```
    sed "s/^ *\(.*\)/\1/;s/ /,/"
    ```

  - Os dados estarão no formato `<#requisições, timestamp>`. Para deixar mais lógica a organização do dados, a ordem desses dois campos foi invertida para `<timestamp, #requisições>`.

[5] O termo *timestamp* será utilizado sem itálico deste ponto em diante.

```
awk 'BEGIN {OFS=FS=","}{print $2,$1}'
```

– A saída foi compactada com o gzip e salva. (`gzip > output.csv.gz`)

7.2.1.2 Copa 98

É composto de 249 arquivos compactados. Em vez de arquivo texto, os traços foram convertidos em formato binário (Código-fonte 13).

Código-fonte 13: Estrutura de dados para representar uma requisição.

```
1  struct request
2    {
3      uint32_t timestamp;
4      uint32_t clientID;
5      uint32_t objectID;
6      uint32_t size;
7      uint8_t method;
8      uint8_t status;
9      uint8_t type;
10     uint8_t server;
11   };
```

Há alguns programas auxiliares que facilitam a leitura, diponibilizados no sítio do ITA/GOV[6]. Diferentemente do Clarknet e do NASA, o timestamp já está disponível diretamente, portanto, foi necessário alterar a linha 110 no arquivo `ita_public_tools/src/read.c` para o seguinte conteúdo

```
printf(``%u\n'', R->timestamp);
```

Todo os arquivos foram processados de uma só vez pelo comando

```
zcat wc_day* | ita_public_tools/bin/read > outputs/wc.ts
```

O processamento foi igual ao descrito no item 2 dos arquivos Clarknet e NASA, mas com diferença no comando sort, em que paralelizou-se a execução da seguinte forma: sort –parallel=8 -S 90

[6] Ferramentas auxiliares para o Copa 98: <ftp://ita.ee.lbl.gov/software/WorldCup_tools.tar.gz>

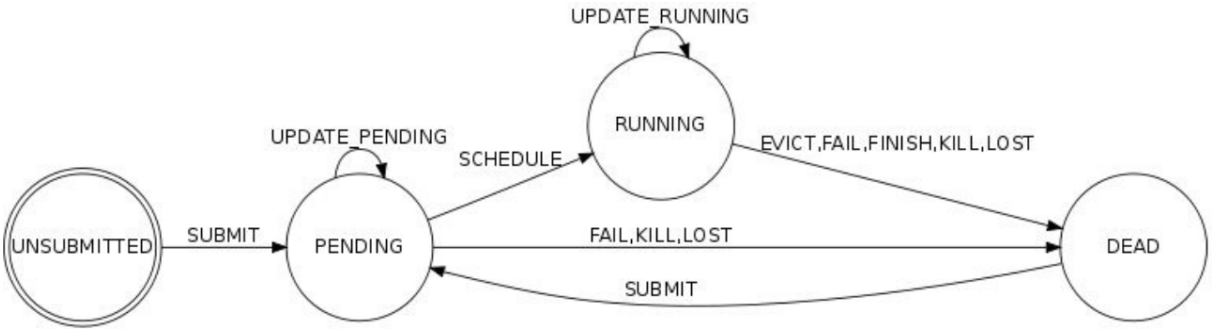


Figura 69 – Ciclo de vida das requisições (jobs ou tasks)

Fonte: Reiss, Wilkes e Hellerstein (2011).

7.2.1.3 Google

Outro conjunto de dados utilizado foi o Google Cluster Data (REISS; WILKES; HELLERSTEIN, 2011). Para obter seu conteúdo obtenha e instale o Google Cloud SDK[7]. A cópia local dos arquivo necessários é feita pelos comandos:

```
gsutil -m cp -R gs://clusterdata-2011-2/schema.csv/ .
gsutil -m cp -R gs://clusterdata-2011-2//job_events/ .
gsutil -m cp -R gs://clusterdata-2011-2/task_events/ .
```

Ele possui dados de chegadas de requisições a um `cluster` com aproximadamente 12,5 mil máquinas. Cada requisições recebe o nome de `job` e cada `job` possui um número n de tarefas (`tasks`). Como detalhado na documentação(REISS; WILKES; HELLERSTEIN, 2011) tanto `jobs` quanto tarefas possuem os estados conforme ilustrado na Fig. 69.

A coleta de dados tem resolução de microsegundos (10^{-6}). No início da coleta considerou-se as tarefas existentes como pertencentes ao instante 0. O dados foram coletados a partir do segundo 600. A Fig. 70 ilustra esse processo.

Os dados estão distribuídos em 500 arquivos (0-499) compactados, cuja estrutura está descrita no arquivo `eschema.csv`. O diretório `jobs_events` contém os dados sobre os `jobs` e o diretório `task_events` contem dados sobre as tarefas.

Os arquivos de `jobs` possuem a seguinte estrutura:

```
> grep job_events schema.csv
job_events/part-?????-of-?????.csv.gz,1,time,INTEGER,YES
job_events/part-?????-of-?????.csv.gz,2,missing info,INTEGER,NO
job_events/part-?????-of-?????.csv.gz,3,job ID,INTEGER,YES
** job_events/part-?????-of-?????.csv.gz,4,event type,INTEGER,YES **
```

[7] Sítio: <https://cloud.google.com/sdk/docs/> e <https://cloud.google.com/storage/docs/gsutil>

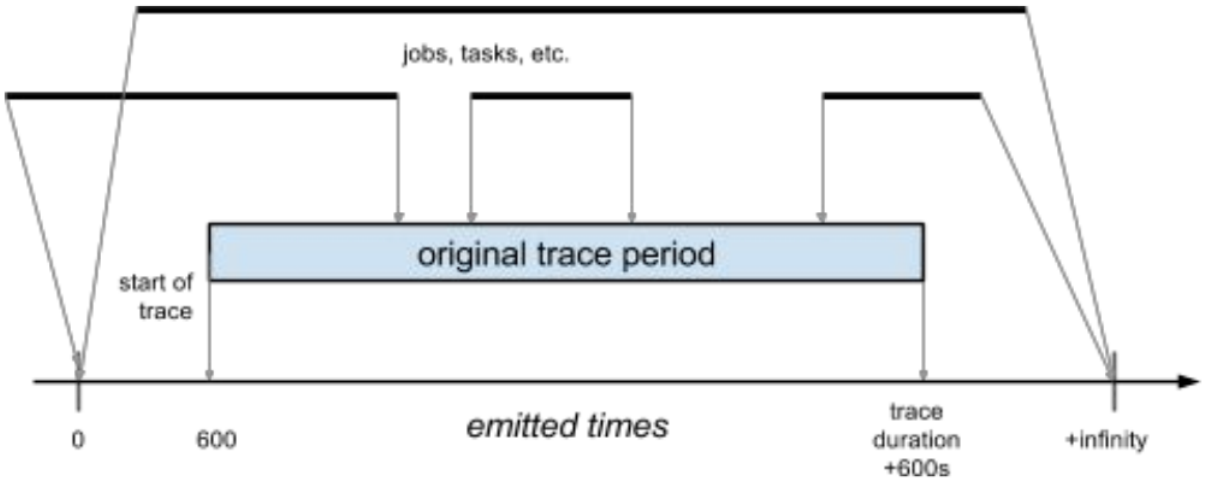


Figura 70 – Convenção utilizada para a coleta do tempo.

Fonte: Reiss, Wilkes e Hellerstein (2011).

```
job_events/part-?????-of-?????.csv.gz,5,user,STRING_HASH,NO
job_events/part-?????-of-?????.csv.gz,6,scheduling class,INTEGER,NO
job_events/part-?????-of-?????.csv.gz,7,job name,STRING_HASH,NO
job_events/part-?????-of-?????.csv.gz,8,logical job name,STRING_HASH,NO
```

Os arquivos de tasks possuem a seguinte estrutura:

```
> grep task_events schema.csv
task_events/part-?????-of-?????.csv.gz,1,time,INTEGER,YES
task_events/part-?????-of-?????.csv.gz,2,missing info,INTEGER,NO
task_events/part-?????-of-?????.csv.gz,3,job ID,INTEGER,YES
task_events/part-?????-of-?????.csv.gz,4,task index,INTEGER,YES
task_events/part-?????-of-?????.csv.gz,5,machine ID,INTEGER,NO
** task_events/part-?????-of-?????.csv.gz,6,event type,INTEGER,YES **
task_events/part-?????-of-?????.csv.gz,7,user,STRING_HASH,NO
task_events/part-?????-of-?????.csv.gz,8,scheduling class,INTEGER,NO
task_events/part-?????-of-?????.csv.gz,9,priority,INTEGER,YES
task_events/part-?????-of-?????.csv.gz,10,CPU request,FLOAT,NO
task_events/part-?????-of-?????.csv.gz,11,memory request,FLOAT,NO
task_events/part-?????-of-?????.csv.gz,12,disk space request,FLOAT,NO
task_events/part-?????-of-?????.csv.gz,13,diff. mach. restrict.,BOOL,NO
```

Estão em destaque as informações de interesse. `Time` corresponde ao tempo e `event type` é o estado de determinado `job` ou tarefa. Há 8 tipos de eventos: `submit(0)`, `schedule(1)`, `evict(2)`, `fail(3)`, `finish(4)`, `kill(5)`, `lost(6)`, `update_pending(7)`, e `update_running(8)`.

De todas as informações coletadas apenas os eventos de `submit` são necessário. Dessa forma, a extração da informação foi feita da seguinte forma:

- Descompactar os arquivos; (`zcat job_events/*.gz` e `zcat tasks_events/*.gz`, um de cada vez)
- Escolher apenas os campos 1 e {4,6} para `jobs` e tarefas repectivamente. (`cut -d, -f 1,4` e `cut -d, -f 1,6`)
- Escolher apenas as linhas que terminam com o evento 0 (`submit`). (`grep 0$`)
- Separar apenas o campo 1, correspondente à informação de tempo (`cut -d, -f 1`)
- Obter o tempo com resolução de segundo, ou seja, dividir o valor encontrado por 10^6.

```
awk '{print int($1/1e6)}'
```

- Seguir os demais passos de ordenar, linhas únicas e compactação.

O processamento foi executado com estas linhas de comando:

```
zcat job_events/*.gz | cut -d, -f 1,4 | grep 0$ | cut -d, -f 1
  | awk '{print int($1/1e6)}' | sort -g | uniq -c
  | sed 's/^ *\(.*\)/\1/;s/ /,/'
  | awk 'BEGIN {OFS=FS=","}{print $2,$1}'
  | gzip > ../gjobs.log.gz
zcat task_events/*.gz | cut -d, -f 1,6 | grep 0$ | cut -d, -f 1
  | awk '{print int($1/1e6)}' | sort -g | uniq -c
  | sed 's/^ *\(.*\)/\1/;s/ /,/'
  | awk 'BEGIN {OFS=FS=","}{print $2,$1}'
  | gzip > ../4-gtasks.log.gz
```

7.2.2 Pós-processamento

Os arquivos gerados desse processamento contêm a informação de que, em determinado instante de tempo, um número específico de requisições chegou ao servidor. Denomina-se que cada linha corresponde aos eventos acontecidos em um instante de tempo. Para que os traços de execução fossem utilizados como entrada para uma função de transferência, os eventos devem ser espalhados em uma série temporal igualmente espaçada. Por exemplo, seja o seguinte arquivo de eventos: (primeira coluna instante de tempo, segunda quantidade de requisições)

```
1234,2
1237,1
1239,2
```

Ele deve ser igualmente espaçado de forma que corresponda a amostras periódicas. O arquivo a ser utilizado na análise deve ser:

```
1234,2
1235,0
1236,0
1237,1
1238,0
1239,2
```

O arquivo para análise foi processado no Matlab. A seguinte função foi utilizada para essa finalidade (Código-fonte 14).

Código-fonte 14: Estrutura de dados para representar uma requisição.

```
1 function fromLogToCSV(logname,csvname,ftime,freqs)
2   dat=csvread(logname);
3   seq=dat(:,ftime);
4   req=dat(:,freqs);
5
6   fullseq=(min(seq):1:max(seq))';
7   fullreq=zeros(length(fullseq),1);
8   fullreq(ismember(fullseq,seq))=req;
9   dlmwrite(csvname,[fullseq fullreq],'precision',10)
10 end
```

Para o caso da série do Google o instante de tempo 0 foi removido, pois ele continha a quantidade de requisições no sistema antes do início da medição.

7.2.3 Problemas: Clarknet e NASA

O traço Clarknet possui um período sem amostragem. Ao unir os dois arquivos de `Aug28` e `Sep4`, obtem-se a série apresentada na Fig. 71.

Portanto, para a análise, a série Clarknet foi dividida em duas partes:

1. `clark-1`: série completa do arquivo `Aug28`;

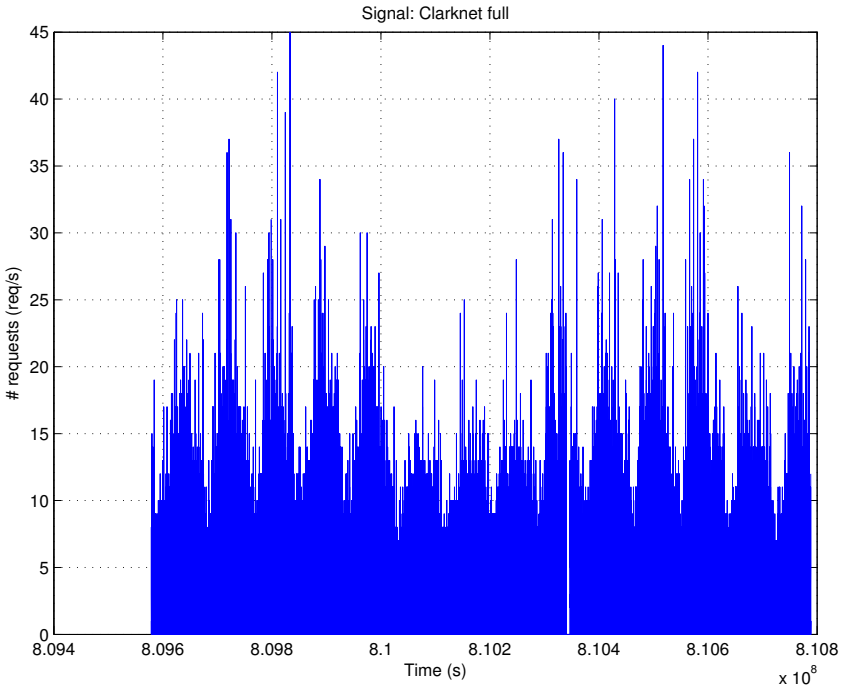


Figura 71 – Dados da série Clarknet versão completa.

2. `clark-2`: série contendo a última parte após a falha de coleta. O ponto de início (*offset*) utilizado no arquivo `Sep4` foi o instante de tempo $1,7 \times 10^5$.

Analogamente, o traço da NASA possui um hiato entre os arquivos e uma falha na coleta de dados. A última linha do arquivo `Jul95` possui data `28/Jul/1995:13:32:25` e a primeira linha do arquivo `Aug95` possui data `01/Aug/1995:00:00:01`. Um hiato de 85,5 horas. A falha ocorreu por causa de um furacão que atingiu o local onde o servidor estava hospedado. Ao unir os dois arquivos de traço, obtém-se a série dados conforme apresentada na Fig. 72.

Da mesma forma, para a análise, a série NASA foi dividida em duas partes:

1. `nasa-1`: série completa do arquivo `Jul95`;

2. `nasa-2`: série contendo a última parte após a falha da coleta. O ponto de início (*offset*) utilizado no arquivo `Aug95` foi o instante de tempo $10,02 \times 10^3$.

7.2.4 Fragmento da série Copa 98

O traço da Copa 98 possui um comportamento de requisições diferente quando comparado com os traços Clarknet e NASA. No início da coleta há um período com menos requisições, no entanto, durante os jogos o número de requisições aumentou de uma forma desproporcional e com características diferentes das iniciais. A Fig. 73 apresenta a série completa deste traço. Decidiu-se extrair apenas o fragmento central dessa série, correspondente aos dias de jogos

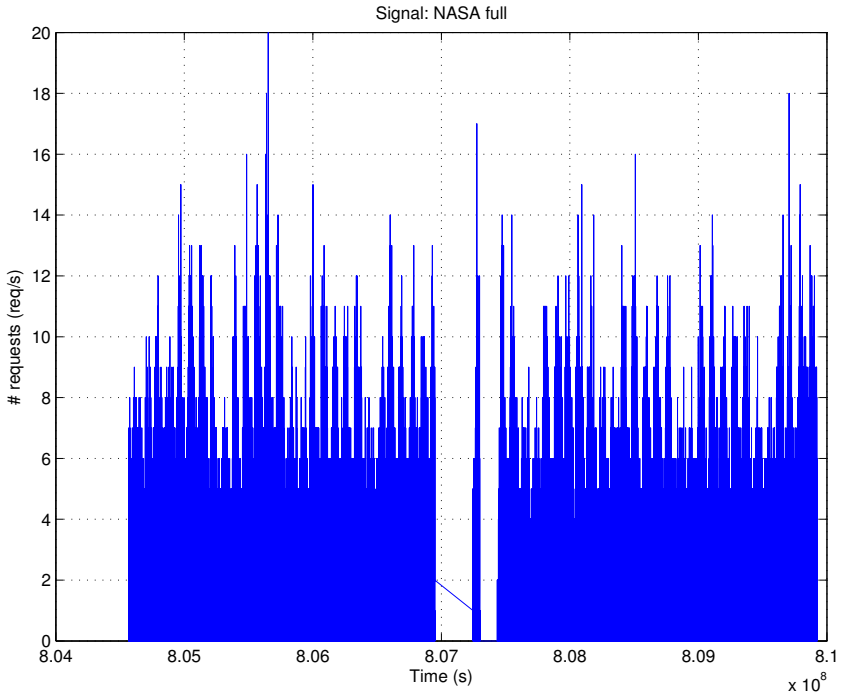


Figura 72 – Dados da série NASA versão completa.

com maior acesso. Os pontos de corte (*offsets*) de início e fim são $3,31 \times 10^6$ e $6,5 \times 10^6$, repectivamente.

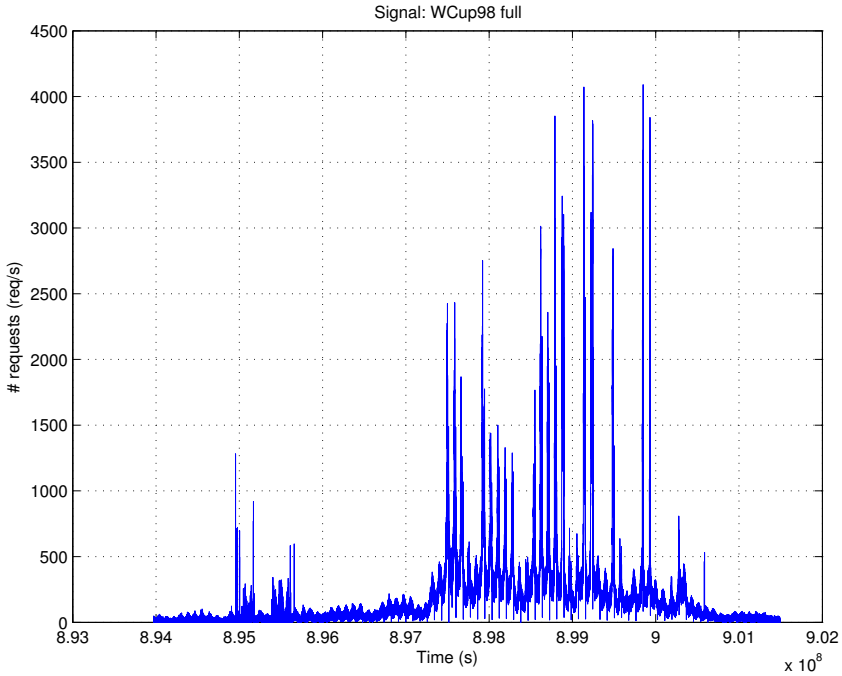


Figura 73 – Dados da série Copa 98 versão completa.

As séries resultantes podem ser observadas nas Figs. 74 a 80 e serão utilizadas para a análise dos dados.

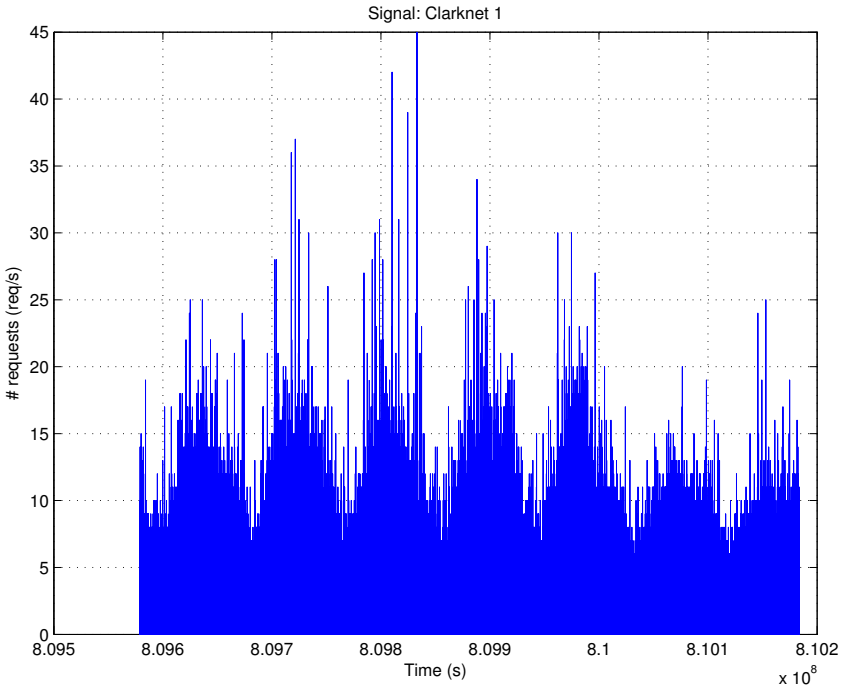


Figura 74 – Dados da série `clark-1`.

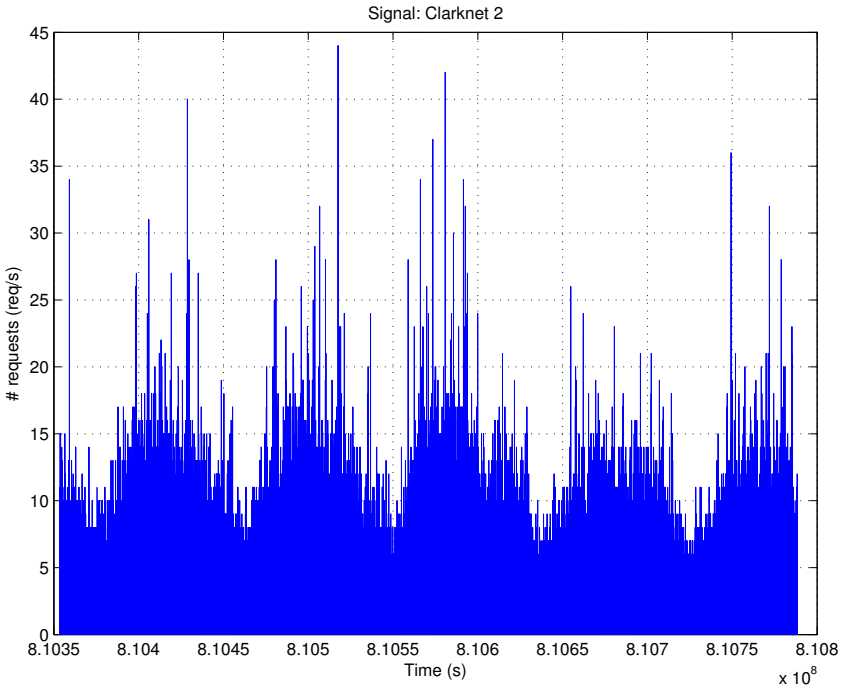


Figura 75 – Dados da série `clark-2`.

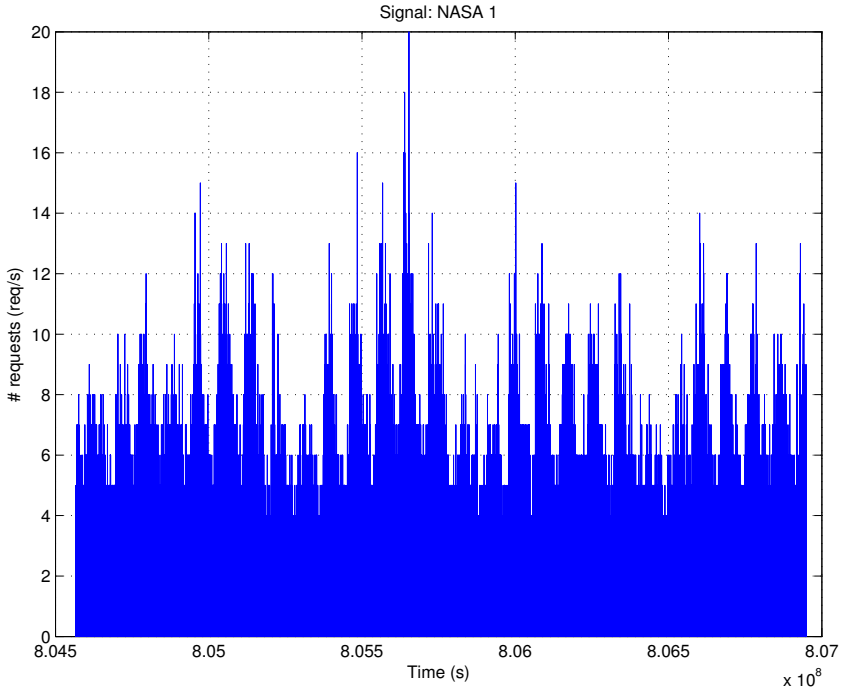


Figura 76 – Dados da série `nasa-1`.

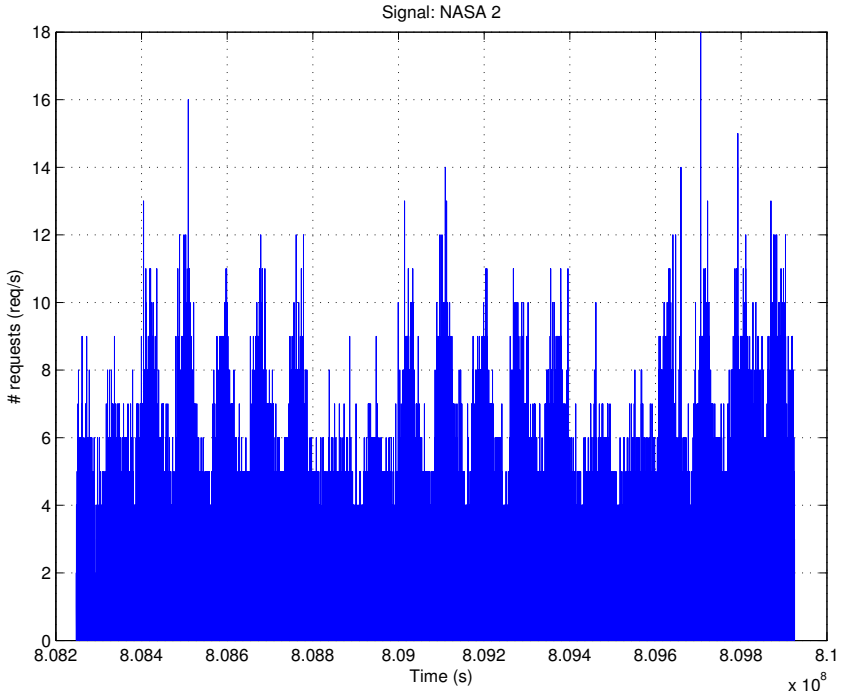


Figura 77 – Dados da série `nasa-2`.

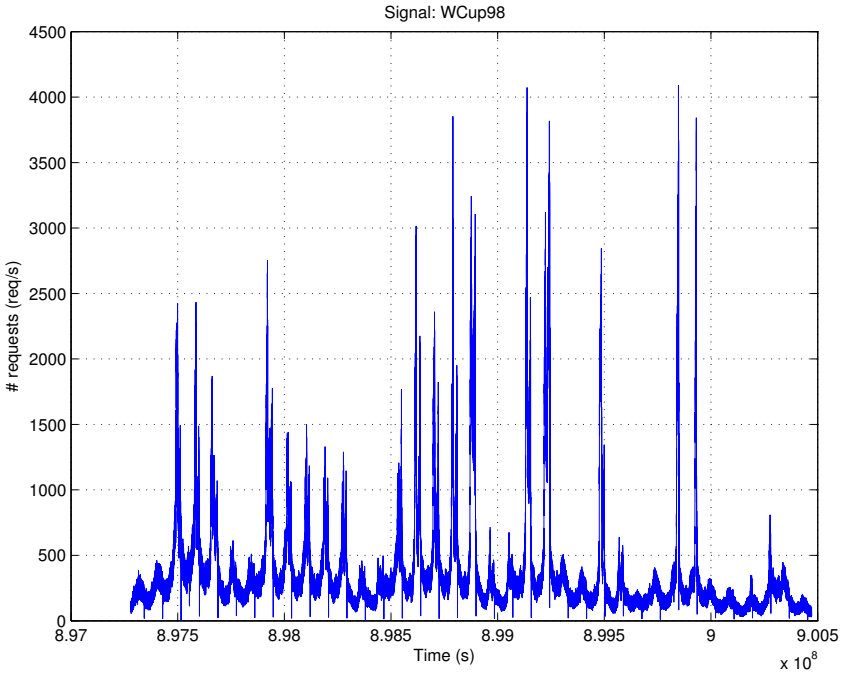


Figura 78 – Dados da série Copa 98.

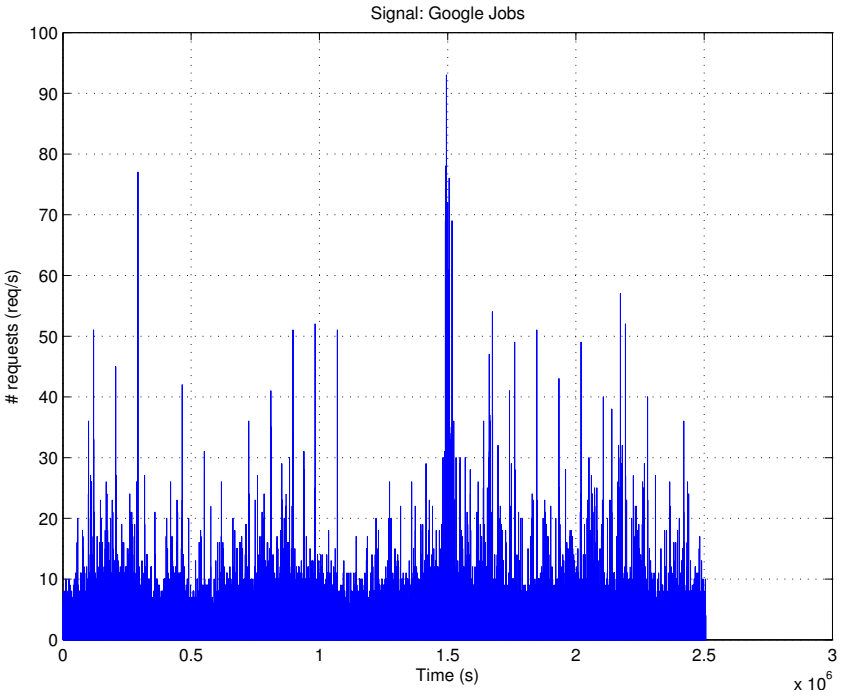


Figura 79 – Dados da série Google `jobs`.

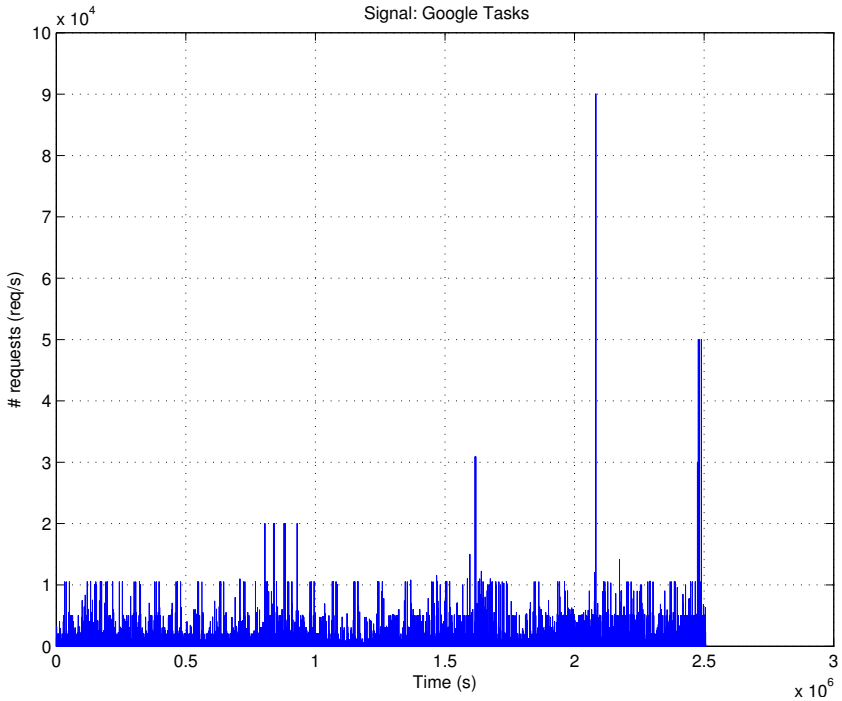


Figura 80 – Dados da série Google tarefas.

7.3 Análise das cargas

Nesta seção os traços são analisados conforme o impacto que eles produzem no sistema. O sinal de entrada corresponde à carga de trabalho, e a saída é a taxa de utilização de CPU gerada pela função de transferência. De posse dos sinais obtidos na seção anterior (7.2), eles foram passados pelo filtro especificado (Eq. 7.8) para gerar o sinal de entrada filtrado. Esses dois sinais de entrada foram passados para o sistema através da função `lsim` do Matlab. Os dois sinai de entrada foram subtraídos para observar a característica do ruído que foi removido pelo filtro. Desse modo, os sinais possuem a seguinte descrição:

- Entrada original: sinal obtido do traço após o pós processamento descrito Seção 7.2;
- Entrada filtrado: sinal original filtrado pelo filtro descrito na Eq. 7.8;
- Ruído da entrada: ruído que foi removido pelo filtro, ele é o resultado da subtração no domínio do tempo dos sinais de entrada original e filtrado;
- Saída original: resultado produzido quando o sinal de entrada original é passado pelo sistema;
- Saída filtrada: resultado produzido quando o sinal de entrada filtrado é passado pelo sistema.

As próximas seções descrevem os resultados obtidos.

7.3.1 Clarknet e NASA

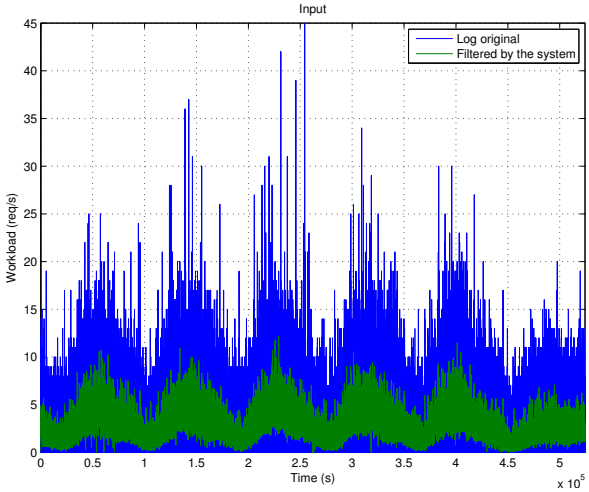
Os traços Clarknet e NASA possuem uma componente periódica muito evidente. Há um padrão evidente na forma com que as requisições chegam ao sistema. Esse padrão é repetido diariamente, podendo haver variações na magnitude do sinal. Por exemplo, finais de semana podem possuir uma menor quantidade de requisições, como é o caso da NASA Fig. 81a. Os resultados desses traço, Fig. 81 e 82, apresentam boa remoção de ruído do sinal de entrada original. Ao observar as saídas correspondentes, percebe-se uma grande semelhança entre elas. As FFTs dos ruídos, Fig. 80c e 81c, apresentam características de ruído branco pois há uma quantidade significante de harmônicos de mesma potência na maioria do espectro. No entanto, parte das frequências baixas possuem maior potência que as demais, o resultado é que o ruído possui a mesma característica periódica. As frequências mais baixas possuem menor potência ou potência próxima de zero, pois elas foram mantidas no sinal de entrada filtrado.

7.3.2 Copa 98

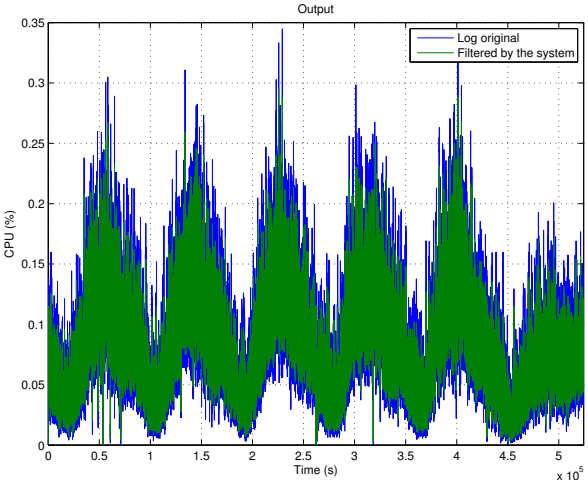
O traço da Copa 98 igualmente aos traços Clarknet e NASA possuem uma componente periódica muito evidente. Apresenta os mesmos padrões periódicos de acesso com frequência diária e com as variações na magnitude, em que os finais de semana possuem menor quantidade de acesso. A Fig. 83 apresenta os resultados. Os sinais de entrada original e filtrado são semelhantes e as saídas também semelhantes. O ruído observado na Fig. 82c, possui uma assinatura semelhante aos do Clarknet e NASA: ruído branco, com harmônicos de baixa frequência com mais potência, e pouca potência nos harmônicos de mais baixa frequência. Uma diferença com aqueles dois traços é que em geral os harmônicos tem uma potência menor em relação ao sinal original. Considere a Fig. 82a em que a magnitude do sinal é na casa de 500 req/s com picos na ordem de 1500 req/s, podendo a chegar a valores ainda maiores. Isso dá indícios que há a presença de altas frequências no sinal em questão, porém elas tendem a influenciar pouco no sinal em sua representação no tempo.

Para entender o motivo do resultado obtido — o filtro influenciou pouco no caso do traço Copa 98 —, a Fig. 84 apresenta a FFT dos sinais de entrada original e filtrado. Foi dado um foco maior nas baixas frequências na largura de banda de 0 Hz a $0,1$ Hz. Os demais harmônicos apresentam a mesma assinatura a partir de $0,1$ Hz, um ruído branco na faixa de $0,1$ Hz a $0,5$ Hz. O que se destaca é a presença de grande potência concentrada nos harmônicos de mais baixa frequência. O resultado é que o filtro atuará pouco nessas frequências, ou seja, não haverá atenuação nos harmônicos que mais influenciam no sinal em sua representação no domínio do tempo. É importante observar que no espectro do sinal de entrada filtrado (Fig. 83b) a atenuação começa a ser mais evidente após o harmônico de frequência $0,01$ Hz (aproximadamente) e, muito embora tenha diminuído a pontência daqueles harmônicos, a potência original já era baixa. Portanto, os sinais de entrada original e filtrado tendem a ser semelhantes, pois as frequências que mais influenciam no seu formato no domínio do tempo foi pouco alterado pelo filtro, e o que

(a) Entrada: Carga de trabalho.



(b) Saída: Taxa de utilização CPU.



(c) Ruído presente na entrada: carga de trabalho original subtraída da filtrada.

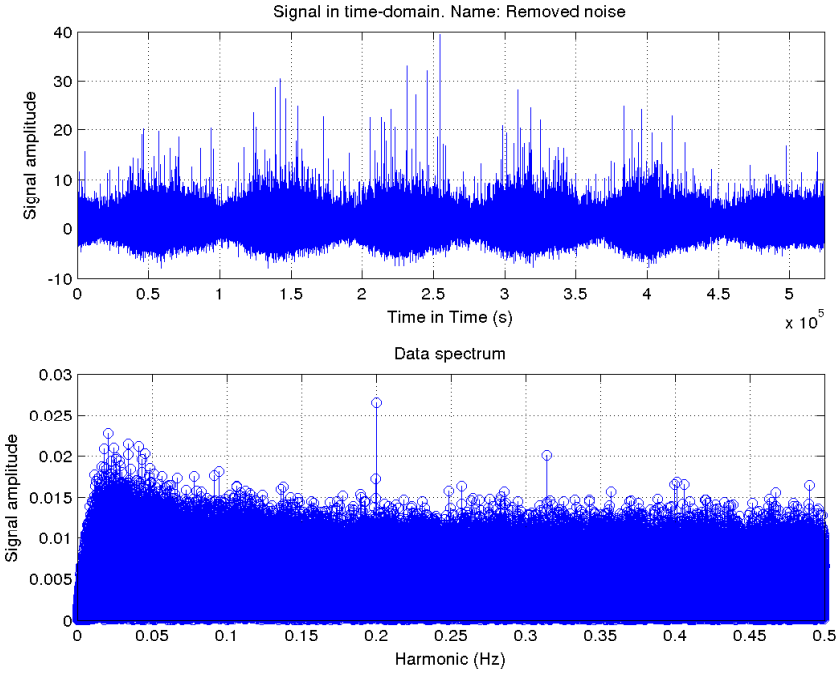
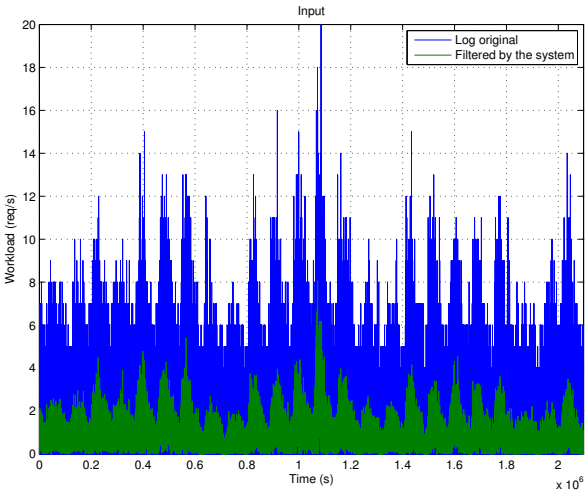
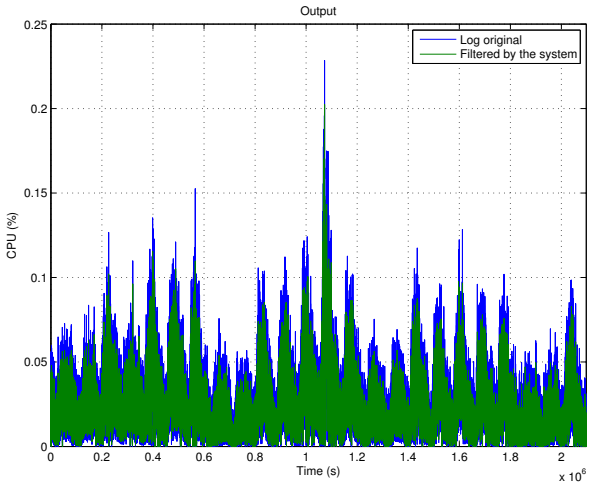


Figura 81 – Análise do traço Clarknet 1. a) apresenta as duas entradas que produzirão saída semelhantes; b) o resultado foi semelhante como previsto; c) a porção do sinal que foi retirada do sinal de entrada original e que foi absorvido pela dinâmica do sistema.

(a) Entrada: Carga de trabalho.



(b) Saída: Taxa de utilização CPU.



(c) Ruído presente na entrada: carga de trabalho original subtraída da filtrada.

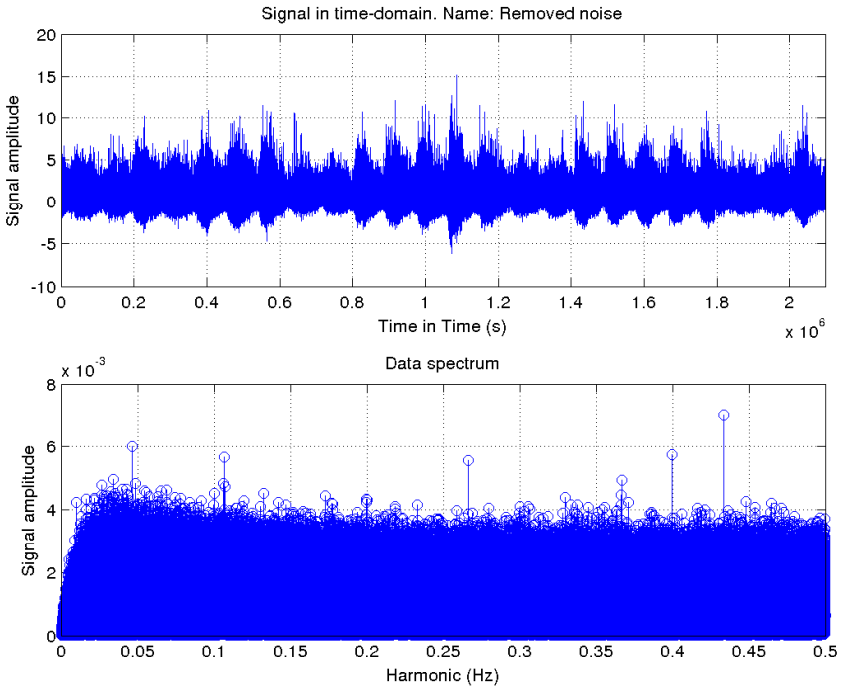
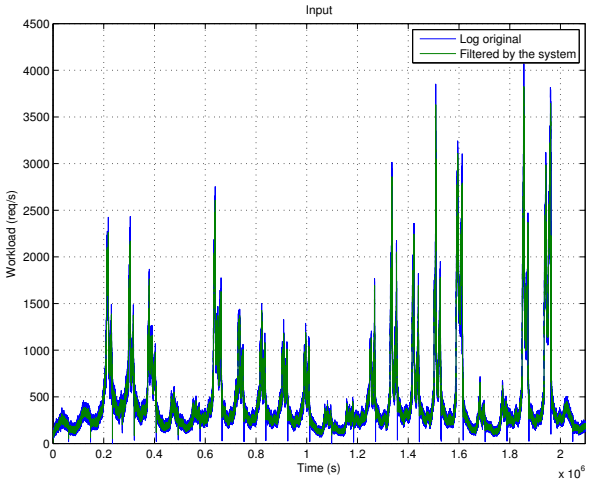
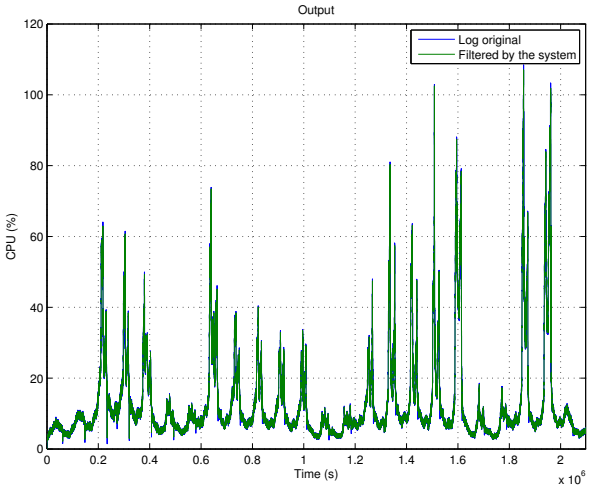


Figura 82 – Análise do traço NASA 1. a) apresenta as duas entradas que produzirão saída semelhantes; b) o resultado foi semelhante como previsto; c) a porção do sinal que foi retirada do sinal de entrada original e que foi absorvido pela dinâmica do sistema.

(a) Entrada: Carga de trabalho.



(b) Saída: Taxa de utilização CPU.



(c) Ruído presente na entrada: carga de trabalho original subtraída da filtrada.

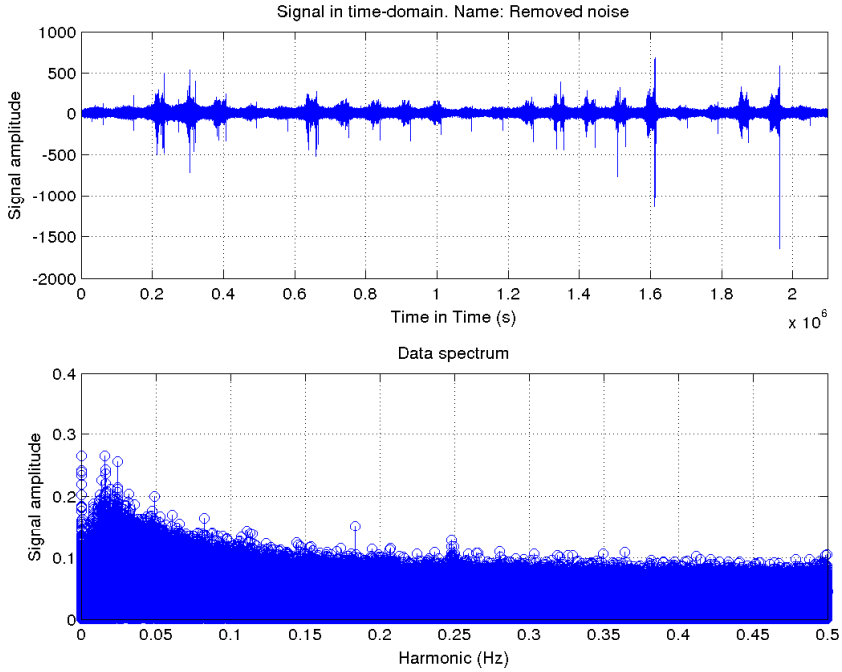


Figura 83 – Análise do traço Copa 98. a) apresenta as duas entradas que produzirão saída semelhantes; b) o resultado foi semelhante como previsto; c) a porção do sinal que foi retirada do sinal de entrada original e que foi absorvido pela dinâmica do sistema.

foi alterado pelo filtro tem pouca influência em seu formato.

7.3.3 Google

Em contraste com os sinais vistos até o momento, Clarknet, NASA e Copa 98, os sinais de entrada da Google possuem formato diferente. Há componentes periódicas pouco evidentes no caso do traço de *jobs* e mais evidente no caso do traço de tarefas. A componente periódica do sinal de tarefas lembra uma onda quadrada. Deve-se ser mencionado que esses dois sinais, *jobs* e tarefas, estão correlacionados, pois, para cada *job* que chega ao sistema da Google (neste caso célula de processamento), um conjunto de n tarefas é criado. Para esta análise os dois sinais foram tomados independentemente.

A Fig. 85 apresenta os resultados obtidos. O ruído da entrada tem uma característica diferente dos outros traços. Há a presença isolada de harmônicos de baixa frequência com potência maior (Fig. 84c). Uma possível explicação para a existência desses harmônicos foram o aumento na quantidade de *jobs* que chegaram ao sistema em torno do instante de tempo 150.000 s (vide Fig. 84a). Não há um pico isolado no aumento e sim uma série de eventos consecutivos que ocasionaram naquela sazonalidade, portanto um conjunto de baixas frequências pode ter sido combinado para que aquela ocorrência aparecesse no domínio do tempo. Outra informação observada na FFT do ruído é que há potência nos harmônicos de baixa frequência, mas com um padrão diferente dos sistemas web. O ruído do traço tarefas é predominantemente branco, com uma queda na potência dos harmônicos de baixa frequência (Fig. 86). Boa parte do sinal foi removida, e as saídas original e filtrada ficaram semelhantes.

7.3.4 Sumário da análise

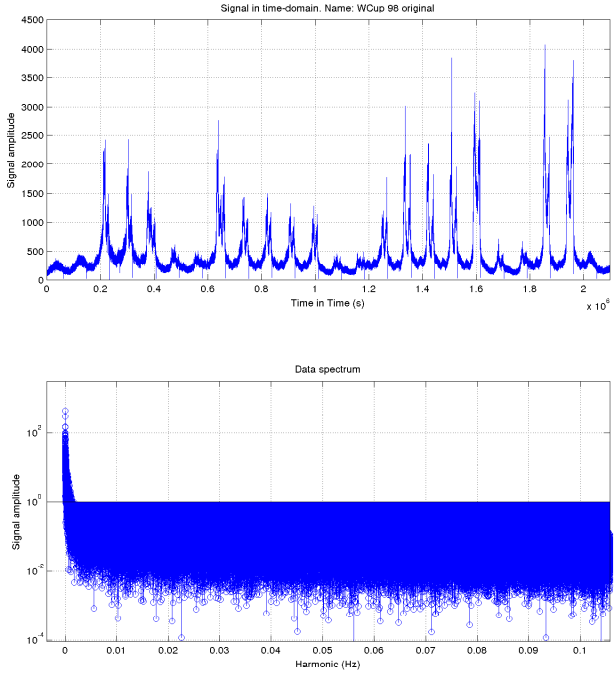
A Tab. 13 apresenta a correlação entre os sinais das entradas original e filtrada e das saídas original e filtrada. Observa-se que há uma correlação menor nos sinais de entrada, ao passo que essa correlação é maior nos sinais de saída, isso significa que as cargas filtradas formam uma síntese representativa da carga real. A exceção é o traço Copa 98 que, conforme já foi discutido, apresenta uma característica de baixas frequências maior, o que faz com que ambos sinais de entrada sejam muito semelhantes e sejam capazes de excitar o sistema de forma equivalente.

Tabela 13 – Correlações entre os sinais de entradas e saídas.

Nome	Entradas	Saídas
Clark	$0,6478$	$0,9509$
NASA	$0,5357$	$0,9198$
Copa 98	$0,9980$	$0,9998$
Google *jobs*	$0,4330$	$0,7931$
Google tarefas	$0,4395$	$0,8481$

Destacam-se como contribuições desta análise a compreensão de como uma carga obtida a partir de um sistema real impacta nas características de outro sistema sem a necessidade de

(a) Entrada original.



(b) Entrada filtrada.

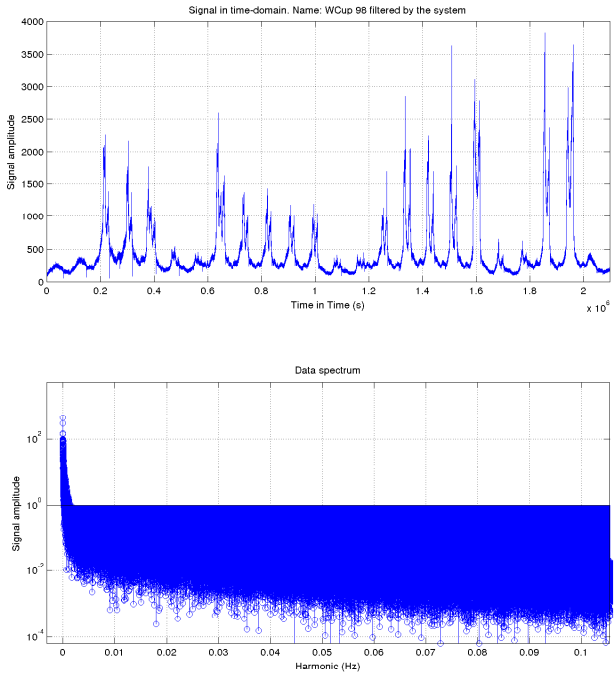
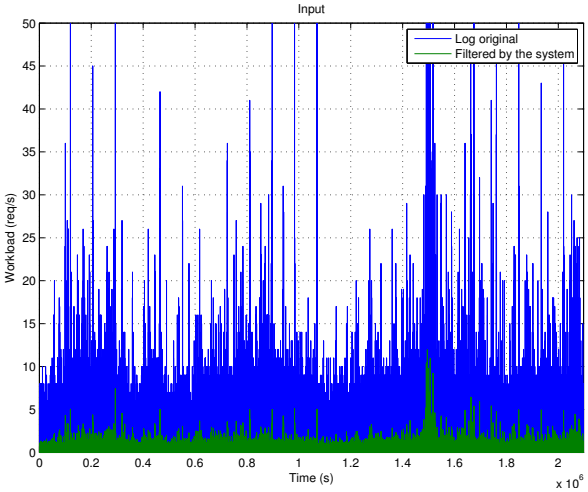
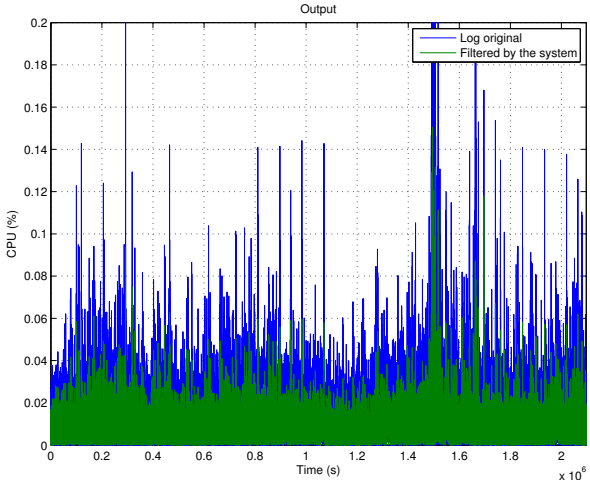


Figura 84 – Comparação entre os espectros do traço Copa 98.

(a) Entrada: Carga de trabalho.



(b) Saída: Taxa de utilização CPU.



(c) Ruído presente na entrada: carga de trabalho original subtraída da filtrada.

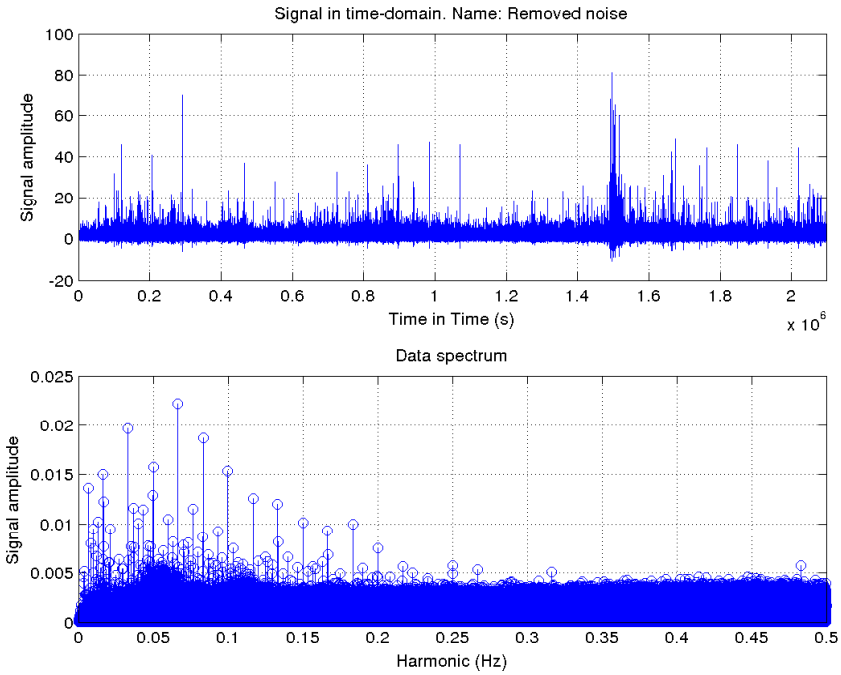
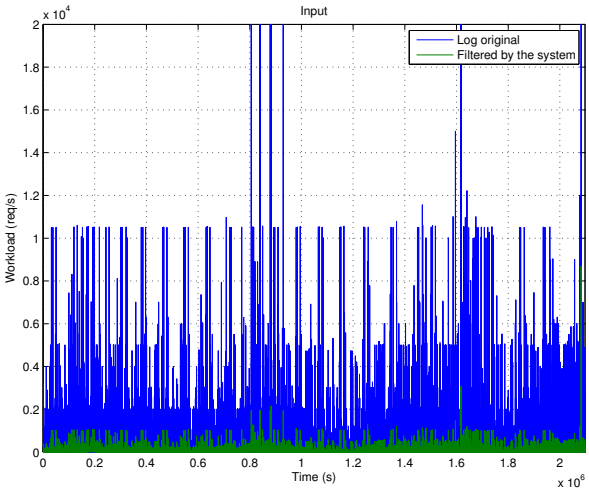
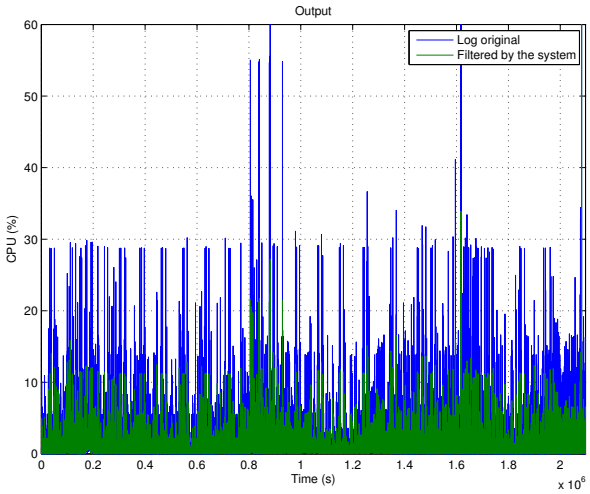


Figura 85 – Análise do Google *jobs*. a) apresenta as duas entradas que produzirão saída semelhantes; b) o resultado foi semelhante como previsto; c) a porção do sinal que foi retirada do sinal de entrada original e que foi absorvido pela dinâmica do sistema.

(a) Entrada: Carga de trabalho.



(b) Saída: Taxa de utilização CPU.



(c) Ruído presente na entrada: carga de trabalho original subtraída da filtrada.

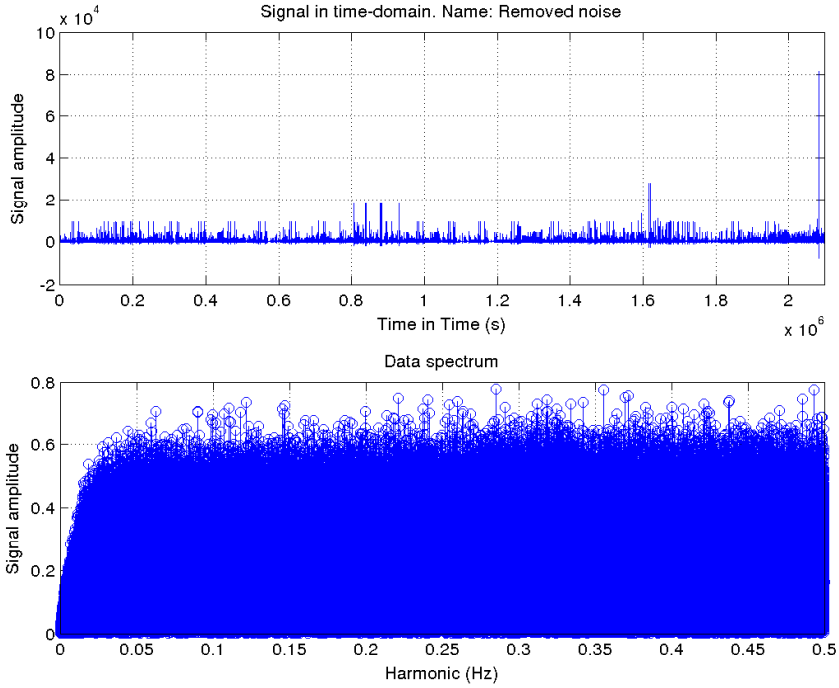


Figura 86 – Análise do Google tarefas. a) apresenta as duas entradas que produzirão saída semelhantes; b) o resultado foi semelhante como previsto; c) a porção do sinal que foi retirada do sinal de entrada original e que foi absorvido pela dinâmica do sistema.

realizar experimentos nesse outro sistema; parâmetros para escolha de uma carga de trabalho que seja significante para o sistema em estudo; e, embora fuja ao escopo do estudo, uma possibilidade de exporar a resposta espectral para geração de carga sintéica com as mesmas características, relativamente ao modelo dinâmico de desempenho do sistema.

CAPÍTULO

8

CONCLUSÕES

O estudo detalha a motivação para a avaliação de desempenho não estacionária de sistemas computacionais e introduz uma abordagem para a modelagem dinâmica de desempenho baseada na análise de resposta em frequência. A modelagem dinâmica é uma abordagem essencial que se desenvolveu como recurso importante na engenharia e algumas ciências naturais por muitas décadas e conta com uma coleção de ferramentas matemáticas para descrever o comportamento dinâmico de sistemas. Seja por tradição ou pela dificuldade de aplicação o uso de modelos dinâmicos em avaliação de desempenho de sistemas computacionais é recente e consideravelmente menos explorada que em outros domínios. Uma questão central tratada pelo presente estudo é enquanto o regime permanente caracteriza o desempenho sob uma determinada carga e seus requisitos de capacidade para lidar com dada demanda, a análise do estado transiente, por outro lado, avalia como o sistema reage a variações na carga. Pode revelar, por exemplo, que os recursos necessários para suportar determinada carga estacionária podem não ser suficientes para atender mudanças bruscas no regime de operação, mesmo que de intensidade menor do que a suportada pela capacidade estacionária do sistema. Ela também pode revelar que o atraso entre uma mudança de carga e o correspondente impacto no desempenho pode reduzir a eficiência de mecanismos adaptativos de alocação de recurso. Além disso, pode evidenciar que a inércia pode impactar o desempenho temporalmente de diversas formas, desde atrasos, evolução monotônica até oscilações, que ultimamente podem impactar a instabilidade do sistema.

Em adição, à discussão dessas questões a avaliação estacionária pode ser utilizada para prever esses efeitos. A contribuição proposta por esta pesquisa, é a formulação de um arcabouço para avaliação de desempenho não estacionário de sistemas computacionais. O propósito desse arcabouço é produzir um modelo analítico dinâmico experimentalmente construído, que represente a dinâmica que governa o desempenho do sistema. A abordagem é composta por: modelo conceitual para formulação de métricas de desempenho transientes; e um método empírico para a obtenção do modelo dinâmico; uma metodologia de análise baseada em resposta de frequência.

Enquanto a análise não estacionária não é um tópico completamente desconhecido, também é verdade que não é um problema amplamente explorado em sistemas computacionais. A esse respeito, de modo geral resultados reportados na literatura frequentemente referem-se a abordagens *ad-hoc* para resolução de problemas específicos. A presente pesquisa contribui com o estado da arte, formulando uma metodologia do ponto de vista da avaliação de desempenho. Uma característica diferencial é o uso de modelo de resposta em frequência como uma ferramenta de análise. Em particular a análise espectral é explorada para a caracterização de carga e observações que são difíceis ou impraticáveis no domínio do tempo. Para chegar a essas conclusões diversas hipóteses tiveram que ser consideradas e validadas experimentalmente. Elas demonstram, por exemplo, que sistemas computacionais complexos podem ser razoavelmente aproximados por modelos dinâmicos simples e que essas aproximações são úteis a despeito de muitas simplificações. Com relação à aplicação prática a pesquisa também contribui com uma arquitetura de requisitos para o projeto de experimentos destinados à obtenção do modelo dinâmico.

Usos práticos são ilustrados através de estudos de caso em que os resultados demonstram a aplicabilidade da abordagem. Em trabalhos futuros a pesquisa sugere a extensão da abordagem para sistemas com múltiplas entradas e o desenvolvimento de uma ferramenta específica que implemente o fluxo de trabalho da abordagem proposta para fácil aplicação em problemas da área.

REFERÊNCIAS

ABDELZAHER, T.; STANKOVIC, J. ControlWare: a middleware architecture for feedback control of software performance. In: **Proceedings 22nd International Conference on Distributed Computing Systems**. IEEE Comput. Soc, 2002. p. 301–310. ISBN 0-7695-1585-1. Disponível em: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1022267>. Citado 2 vezes nas páginas 15 e 52.

ABDELZAHER, T.; STANKOVIC, J.; LU, C.; ZHANG, R.; LU, Y. **Feedback Performance Control in Software Services**. 2003. Citado na página 48.

AMAZON. **Amazon EC2 FAQs**. 2016. Acessado em 08/07/2016. Citado na página 93.

AN, B.; LESSER, V. Automated negotiation with decommitment for dynamic resource allocation in cloud computing. **Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems**, v. 1, p. 981–988, 2010. Disponível em: <http://dl.acm.org/citation.cfm?id=1838338>. Citado na página 54.

ARMBRUSTER, D.; MARTHALER, D.; RINGHOFER, C.; KEMPF, K.; JO, T. A continuum model for a re-entrant factory. **Operations Research**, v. 54, n. 5, p. 933–950, 2006. Disponível em: <http://dx.doi.org/10.1287/opre.1060.0321>. Citado na página 43.

BARTON, R. R. Tutorial: Simulation metamodeling. In: **Proceedings of the 2015 Winter Simulation Conference**. Piscataway, NJ, USA: IEEE Press, 2015. (WSC '15), p. 1765–1779. ISBN 978-1-4673-9741-4. Disponível em: <http://dl.acm.org/citation.cfm?id=2888619.2888818>. Citado na página 46.

BASS, F. M. A new product growth for model consumer durables. **Management Science**, v. 15, n. 5, p. 215–227, 1969. Disponível em: <http://dx.doi.org/10.1287/mnsc.15.5.215>. Citado na página 43.

BOGGIA, G.; CAMARDA, P.; GRIECO, L. A.; MASCOLO, S. Feedback-Based Control for Providing Real-Time Services With the 802.11e MAC. **Networking, IEEE/ACM Transactions on**, v. 15, n. 2, p. 323–333, abr. 2007. ISSN 1063-6692. Citado na página 48.

BROWN, A. B.; HELLERSTEIN, J. L. An approach to benchmarking configuration complexity. In: **Proceedings of the 11th Workshop on ACM SIGOPS European Workshop**. New York, NY, USA: ACM, 2004. (EW 11). Disponível em: <http://doi.acm.org/10.1145/1133572.1133609>. Citado na página 50.

BUX, M.; LESER, U. Dynamiccloudsim: Simulating heterogeneity in computational clouds. **Future Generation Computer Systems**, v. 46, n. 0, p. 85 – 99, 2015. ISSN 0167-739X. Citado na página 80.

BUYYA, R.; RANJAN, R.; CALHEIROS, R. N. Modeling and simulation of scalable Cloud computing environments and the CloudSim toolkit: Challenges and opportunities. In: **2009 International Conference on High Performance Computing & Simulation**. [S.l.]: IEEE, 2009. p. 1–11. ISBN 978-1-4244-4906-4. Citado na página 54.

CALHEIROS, R. N.; RANJAN, R.; BELOGLAZOV, A.; ROSE, C. A. F. D.; BUYYA, R. Cloudsim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. **Softw. Pract. Exper.**, John Wiley & Sons, Inc., New York, NY, USA, v. 41, n. 1, p. 23–50, jan. 2011. ISSN 0038-0644. Disponível em: <http://dx.doi.org/10.1002/spe.995>. Citado 3 vezes nas páginas 39, 78 e 81.

CASOLARI, S.; TOSI, S.; PRESTI, F. L. An adaptive model for online detection of relevant state changes in internet-based systems. **Perform. Eval.**, Elsevier Science Publishers B. V., Amsterdam, The Netherlands, v. 69, n. 5, p. 206–226, maio 2012. ISSN 0166-5316. Disponível em: <http://dx.doi.org/10.1016/j.peva.2011.05.003>. Citado na página 51.

CHEN, H.; MANDELBAUM, A. Hierarchical modeling of stochastic networks, part i: Fluid models. In: ____. **Stochastic Modeling and Analysis of Manufacturing Systems**. New York, NY: Springer New York, 1994. p. 47–105. ISBN 978-1-4612-2670-3. Disponível em: <http://dx.doi.org/10.1007/978-1-4612-2670-3_2>. Citado na página 43.

CHEN, Y.; DAS, A.; QIN, W.; SIVASUBRAMANIAM, A.; WANG, Q.; GAUTAM, N. Managing server energy and operational costs in hosting centers. **ACM SIGMETRICS Performance Evaluation Review**, v. 33, n. 1, p. 303, jun. 2005. ISSN 01635999. Disponível em: <http://portal.acm.org/citation.cfm?doid=1071690.1064253>. Citado na página 48.

CLARK, G. M. Use of polya distributions in approximate solutions to nonstationary m/m/s queues. **Commun. ACM**, ACM, New York, NY, USA, v. 24, n. 4, p. 206–217, abr. 1981. ISSN 0001-0782. Disponível em: <http://doi.acm.org/10.1145/358598.358620>. Citado na página 42.

COADY, Y.; COX, R.; DETREVILLE, J.; DRUSCHEL, P.; HELLERSTEIN, J.; HUME, A.; KEETON, K.; NGUYEN, T.; SMALL, C.; STEIN, L.; WARFIELD, A. Green team paper: Falling off the cliff: when systems go nonlinear. 2005. Citado na página 50.

DIAO, Y.; HELLERSTEIN, J. L.; PAREKH, S. Control of large scale computing systems. **SIGBED Rev.**, ACM, New York, NY, USA, v. 3, n. 2, p. 17–22, abr. 2006. ISSN 1551-3688. Disponível em: <http://doi.acm.org/10.1145/1143489.1143494>. Citado na página 49.

DIAO, Y.; HELLERSTEIN, J. L.; PAREKH, S.; GRIFFITH, R.; KAISER, G. E.; PHUNG, D. A control theory foundation for self-managing computing systems. **IEEE Journal on Selected Areas in Communications**, v. 23, n. 12, p. 2213–2222, Dec 2005. ISSN 0733-8716. Citado na página 50.

DIAO, Y.; HELLERSTEIN, J. L.; PAREKH, S.; GRIFFITH, R.; KAISER, G.; PHUNG, D. Self-managing systems: a control theory foundation. In: **12th IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'05)**. [S.l.: s.n.], 2005. p. 441–448. Citado na página 50.

DIAO, Y.; HELLERSTEIN, J. L.; PAREKH, S.; SHAIKH, H.; SURENDRA, M. Controlling quality of service in multi-tier web applications. In: **26th IEEE International Conference on Distributed Computing Systems (ICDCS'06)**. [S.l.: s.n.], 2006. p. 25–25. ISSN 1063-6927. Citado na página 51.

DRAPER, N. R.; SMITH, H. **Applied Regression Analysis (Wiley Series in Probability and Statistics)**. Third. Wiley-Interscience, 1998. Hardcover. ISBN 0471170828. Disponível em: <http://www.worldcat.org/isbn/0471170828>. Citado na página 66.

EICK, S. G.; MASSEY, W. A.; WHITT, W. Mt/g/∞ queues with sinusoidal arrival rates. **Manage. Sci.**, INFORMS, v. 39, n. 2, p. 241–252, fev. 1993. ISSN 0025-1909. Disponível em: <http://dx.doi.org/10.1287/mnsc.39.2.241>. Citado na página 42.

____. The physics of the mt/g/ ∞ symbol queue. **Oper. Res.**, INFORMS, v. 41, n. 4, p. 731–742, jul. 1993. ISSN 0030-364X. Disponível em: <http://dx.doi.org/10.1287/opre.41.4.731>. Citado na página 42.

FILIERI, A.; HOFFMANN, H.; MAGGIO, M. Automated design of self-adaptive software with control-theoretical formal guarantees. In: **Proceedings of the 36th International Conference on Software Engineering**. New York, NY, USA: ACM, 2014. (ICSE 2014), p. 299–310. ISBN 978-1-4503-2756-5. Disponível em: <10.1145/2568225.2568272>. Citado na página 48.

FISHWICK, P. A. Simpack: Getting started with simulation programming in c and c++. In: **Proceedings of the 24th Conference on Winter Simulation**. New York, NY, USA: ACM, 1992. (WSC '92), p. 154–162. ISBN 0-7803-0798-4. Disponível em: <http://doi.acm.org/10.1145/167293.167322>. Citado 2 vezes nas páginas 43 e 61.

FITO, J. O.; GOIRI, I.; GUITART, J. SLA-driven Elastic Cloud Hosting Provider. In: **Parallel, Distributed and Network-Based Processing (PDP), 2010 18th Euromicro International Conference on**. [S.l.: s.n.], 2010. p. 111–118. ISSN 1066-6192. Citado na página 54.

GERMAN, R. **Performance Analysis of Communication Systems with Non-Markovian Stochastic Petri Nets**. New York, NY, USA: John Wiley & Sons, Inc., 2000. ISBN 0471492582. Citado na página 46.

GRASSMANN, W. K. Transient solutions in markovian queueing systems. **Computers & OR**, v. 4, n. 1, p. 47–53, 1977. Disponível em: <http://dx.doi.org/10.1016/0305-0548(77)90007-7>. Citado na página 42.

GREEN, L.; KOLESAR, P. The pointwise stationary approximation for queues with nonstationary arrivals. **Manage. Sci.**, INFORMS, v. 37, n. 1, p. 84–97, jan. 1991. ISSN 0025-1909. Disponível em: <http://dx.doi.org/10.1287/mnsc.37.1.84>. Citado na página 42.

GREEN, L.; KOLESAR, P.; SVORONOS, A. Some effects of nonstationarity on multiserver markovian queueing systems. **Oper. Res.**, INFORMS, v. 39, n. 3, p. 502–511, maio 1991. ISSN 0030-364X. Disponível em: <http://dx.doi.org/10.1287/opre.39.3.502>. Citado na página 42.

GROSS, D.; HARRIS, C. M. **Fundamentals of Queueing Theory (2Nd Ed.).** New York, NY, USA: John Wiley & Sons, Inc., 1985. ISBN 0-471-89067-7. Citado na página 41.

GROSS, D.; MILLER, D. The randomization technique as a modeling tool and solution procedure for transient markov processes. **Operations Research**, v. 32, n. 2, p. 343–361, 1984. Disponível em: <http://dx.doi.org/10.1287/opre.32.2.343>. Citado na página 42.

HACKMAN, S. Dynamic production function approximations. In: ____. **Production Economics: Integrating the Microeconomic and Engineering Perspectives**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008. p. 337–371. ISBN 978-3-540-75751-1. Disponível em: <http://dx.doi.org/10.1007/978-3-540-75751-1_19>. Citado na página 43.

HARJUNKOSKI, I.; NYSTRöM, R.; HORCH, A. Integration of scheduling and control—Theory or practice? **Computers & Chemical Engineering**, Elsevier, v. 33, n. 12, p. 1909–1918, 2009. Citado na página 48.

HELLERSTEIN, J. L.; DIAO, Y.; PAREKH, S.; TILBURY, D. M. **Feedback Control of Computing Systems**. 1. ed. Hoboken, NJ, USA: John Wiley & Sons, Inc., 2004. 429 p. ISBN 047126637X. Disponível em: <http://doi.wiley.com/10.1002/047166880X>. Citado 4 vezes nas páginas 44, 58, 63 e 69.

HELLERSTEIN, J. L.; PAREKH, S.; GRIFFITH, R.; KAISER, G.; PHUNG, D. A control theory foundation for self-managing computing systems. **IEEE Journal on Selected Areas in Communications**, v. 23, n. 12, p. 2213–2222, dez. 2005. ISSN 0733-8716. Disponível em: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1546093>. Citado 2 vezes nas páginas 47 e 48.

HENRIKSSON, D.; LU, Y.; ABDELZAHER, T. Improved prediction for web server delay control. **Proceedings. 16th Euromicro Conference on Real-Time Systems, 2004. ECRTS 2004.**, Ieee, p. 61–68, 2004. Disponível em: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1311001>. Citado na página 48.

HORN, P. **Autonomic computing: IBM's Perspective on the State of Information Technology**. IBM, 2001. 1–40 p. (Tertiary Autonomic computing: IBM's perspective on the state of information technology, 26). Disponível em: <http://www.research.ibm.com/autonomic/manifesto/autonomic_computing.pdf>. Citado 2 vezes nas páginas 47 e 50.

HORVáTH, A.; PAOLIERI, M.; RIDI, L.; VICARIO, E. Transient analysis of non-markovian models using stochastic state classes. **Performance Evaluation**, v. 69, n. 7–8, p. 315 – 335, 2012. ISSN 0166-5316. Selected papers from {QEST} 2010. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0166531611001520>. Citado na página 46.

HUANG, D.; HE, B.; MIAO, C. A survey of resource management in multi-tier web applications. **IEEE Communications Surveys Tutorials**, v. 16, n. 3, p. 1574–1590, Third 2014. ISSN 1553-877X. Citado na página 54.

INGOLFSSON, A.; AKHMETSHINA, E.; BUDGE, S.; LI, Y.; WU, X. A survey and experimental comparison of service-level-approximation methods for nonstationary m(t)/m/s(t) queueing systems with exhaustive discipline. **INFORMS Journal on Computing**, v. 19, n. 2, p. 201–214, 2007. Disponível em: <http://dx.doi.org/10.1287/ijoc.1050.0157>. Citado na página 41.

JAIN, J.; MOHANTY, G.; BÖHM, W. **A Course on Queueing Models**. [S.l.]: Taylor & Francis, 2006. (Statistics: A Series of Textbooks and Monographs). ISBN 9781584886464. Citado na página 47.

JAIN, R. **The art of computer systems performance analysis: techniques for experimental design, measurement, simulation, and modeling**. [S.l.]: John Wiley & Sons, 1990. Citado 2 vezes nas páginas 44 e 59.

JENNINGS, O. B.; M, A.; MASSEY, W. A.; WHITT, W. Server staffing to meet time-varying demand. **Management Science**, v. 42, p. 1383–1394, 1996. Citado na página 42.

JIANG, W.-w.; CUI, H.-y.; CHEN, J.-y. A fuzzy modeling based dynamic resource allocation strategy in service grid. **The Journal of China Universities of Posts and Telecommunications**, v. 16, Supplement 1, n. 0, p. 108–113, 2009. ISSN 1005-8885. Disponível em: <http://www.sciencedirect.com/science/article/pii/S1005888508603465>. Citado na página 54.

JOHNSON, N. L.; KOTZ, S. **Distributions in Statistics: Discrete Distributions**. Boston: Houghton Mifflin, 1969. Citado na página 42.

KAMRA, a.; MISRA, V.; NAHUM, E. Yaksha: a self-tuning controller for managing the performance of 3-tiered web sites. In: **Twelfth IEEE International Workshop on Quality of Service, 2004. IWQOS 2004.** IEEE, 2004. p. 47–56. ISBN 0-7803-8277-3. Disponível em: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1309356>. Citado na página 48.

KLEINROCK, L. **Theory, Volume 1, Queueing Systems**. [S.l.]: Wiley-Interscience, 1975. ISBN 0471491101. Citado na página 41.

KLEMS, M.; NIMIS, J.; TAI, S. Do Clouds Compute? A Framework for Estimating the Value of Cloud Computing. In: WEINHARDT, C.; LUCKNER, S.; ER, J. S. (Ed.). **Designing E-Business Systems. Markets, Services, and Networks**. Springer Berlin Heidelberg, 2009, (Lecture Notes in Business Information Processing, v. 22). p. 110–123. ISBN 978-3-642-01255-6. Disponível em: <http://dx.doi.org/10.1007/978-3-642-01256-3_10>. Citado na página 54.

KUSIC, D.; KANDASAMY, N. Risk-aware limited lookahead control for dynamic resource provisioning in enterprise computing systems. **Cluster Computing**, Springer US, v. 10, n. 4, p. 395–408, 2007. ISSN 1386-7857. Disponível em: <http://dx.doi.org/10.1007/s10586-007-0022-y>. Citado 3 vezes nas páginas 15, 54 e 55.

LAI, K.; RASMUSSON, L.; ADAR, E.; ZHANG, L.; HUBERMAN, B. A. Tycoon: An implementation of a distributed, market-based resource allocation system. **Multiagent Grid Syst.**, IOS Press, Amsterdam, The Netherlands, The Netherlands, v. 1, n. 3, p. 169–182, ago. 2005. ISSN 1574-1702. Disponível em: <http://dl.acm.org/citation.cfm?id=1233813.1233816>. Citado na página 54.

LEVA, A.; PAPADOPOULOS, A.; MAGGIO, M. A general control-theoretical methodology for runtime resource allocation in computing systems. In: **Decision and Control (CDC), 2013 IEEE 52nd Annual Conference on**. [S.l.: s.n.], 2013. p. 3487–3492. ISSN 0743-1546. Citado na página 48.

LEWIS, P. R.; MARROW, P.; YAO, X. Resource Allocation in Decentralised Computational Systems: An Evolutionary Market-Based Approach. **Autonomous Agents and Multi-Agent Systems**, Springer-Verlag, v. 21, n. 2, p. 143–171, 2010. Disponível em: <http://eprints.bham.ac.uk/539/>. Citado na página 54.

LIGHTSTONE, S. S.; SURENDRA, M.; DIAO, Y.; PAREKH, S.; HELLERSTEIN, J. L.; ROSE, K.; STORM, A. J.; GARCIA-ARELLANO, C. Control theory: a foundational technique for self managing databases. In: **Data Engineering Workshop, 2007 IEEE 23rd International Conference on**. [S.l.: s.n.], 2007. p. 395–403. Citado na página 50.

LIM, H. C.; BABU, S.; CHASE, J. S.; PAREKH, S. S. Automated control in cloud computing. In: **Proceedings of the 1st workshop on Automated control for datacenters and clouds - ACDC '09**. New York, New York, USA: ACM Press, 2009. p. 13. ISBN 9781605585857. Disponível em: <http://dl.acm.org/citation.cfm?id=1555275http://portal.acm.org/citation.cfm?doid=1555271.1555275>. Citado 2 vezes nas páginas 48 e 56.

LIU, J.; YANG, F. Evaluating the transient behavior of queueing systems via simulation and transfer function modeling. In: **Proceedings of the 2008 Winter Simulation Conference, Global Gateway to Discovery, WSC 2008, InterContinental Hotel, Miami, Florida, USA, December 7-10, 2008**. [s.n.], 2008. p. 516–524. Disponível em: <http://dx.doi.org/10.1109/WSC.2008.4736108>. Citado na página 41.

LIU, X.; SHA, L.; DIAO, Y.; FROEHLICH, S.; HELLERSTEIN, J. L.; PAREKH, S. Online response time optimization of apache web server. In: ____. **Quality of Service — IWQoS 2003: 11th International Workshop Berkeley, CA, USA, June 2–4, 2003 Proceedings**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003. p. 461–478. ISBN 978-3-540-44884-6. Disponível em: <http://dx.doi.org/10.1007/3-540-44884-5_25>. Citado na página 49.

LJUNG, L. (Ed.). **System identification (2nd ed.): theory for the user**. Upper Saddle River, NJ, USA: Prentice Hall PTR, 1999. ISBN 0-13-656695-2. Citado na página 58.

LU, C.; ABDELZAHER, T. F.; STANKOVIC, J. A.; SON, S. H. A Feedback Control Approach for Guaranteeing Relative Delays in Web Servers. In: **RTAS '01: Proceedings of the Seventh Real-Time Technology and Applications Symposium (RTAS '01)**. Washington, DC, USA: IEEE Computer Society, 2001. p. 51. Citado na página 48.

LU, C.; LU, Y.; ABDELZAHER, T. F.; STANKOVIC, J. A.; SON, S. H. Feedback Control Architecture and Design Methodology for Service Delay Guarantees in Web Servers. **IEEE Transactions on Parallel and Distributed Systems**, v. 17, n. 9, p. 1014–1027, set. 2006. ISSN 1045-9219. Disponível em: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1668065>. Citado 3 vezes nas páginas 15, 52 e 53.

LU, C.; STANKOVIC, J. A.; SON, S. H.; TAO, G. Feedback Control Real-Time Scheduling : Framework , Modeling , and Algorithms. **Real-Time Systems**, v. 23, p. 85–126, 2002. Citado na página 52.

LU, C.; STANKOVIC, J. A.; TAO, G.; SON, S. H. Design and Evaluation of a Feedback Control EDF Scheduling Algorithm. In: **RTSS '99: Proceedings of the 20th IEEE Real-Time Systems Symposium**. Washington, DC, USA: IEEE Computer Society, 1999. p. 56. ISBN 0-7695-0475-2. Citado 2 vezes nas páginas 48 e 52.

LU, Y.; ABDELZAHER, T.; LU, C.; SHA, L.; LIU, X. Feedback control with queueing-theoretic prediction for relative delay guarantees in web servers. In: **Real-Time and Embedded Technology and Applications Symposium, 2003. Proceedings. The 9th IEEE**. [S.l.: s.n.], 2003. p. 208–217. Citado 4 vezes nas páginas 15, 48, 52 e 53.

LUNZE, J.; LEHMANN, D. A state-feedback approach to event-based control. **Automatica**, Elsevier, v. 46, n. 1, p. 211–215, 2010. Citado na página 48.

LUZ, H. J. F. da; JúNIOR, L. A. P.; SOUZA, F. L. dos Santos de; MONACO, F. J. 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, 2016. Citado 2 vezes nas páginas 92 e 93.

MAMANI, E. L. C. **Metodologia de benchmark para avaliação de desempenho não-estacionária: um estudo de caso baseado em aplicações de computação em nuvem**. Tese (Doutorado) — ICMC USP, 2016. Citado 3 vezes nas páginas 39, 48 e 110.

MAMANI, E. L. C.; PEREIRA, L. A.; SANTANA, M. J.; SANTANA, R. H. C.; NOBILE, P. N.; MONACO, F. J. Transient performance evaluation of cloud computing applications and dynamic resource control in large-scale distributed systems. In: **High Performance Computing Simulation (HPCS), 2015 International Conference on**. [S.l.: s.n.], 2015. p. 246–253. Citado 3 vezes nas páginas 75, 92 e 108.

MANDELBAUM, W. A. M. A. Strong approximations for time-dependent queues. **Mathematics of Operations Research**, INFORMS, v. 20, n. 1, p. 33–64, 1995. ISSN 0364765X, 15265471. Disponível em: <http://www.jstor.org/stable/3690106>. Citado na página 43.

MASSEY, W. A.; WHITT, W. Peak congestion in multi-server service systems with slowly varying arrival rates. **Queueing Systems**, v. 25, n. 1, p. 157–172, 1997. ISSN 1572-9443. Disponível em: <http://dx.doi.org/10.1023/A:1019156418862>. Citado na página 42.

MENASCE, D. TPC-W: a benchmark for e-commerce. **IEEE Internet Computing**, v. 6, n. 3, p. 83–87, maio 2002. ISSN 1089-7801. Disponível em: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1003136>. Citado na página 108.

MENASCÉ, D. A.; ALMEIDA, V. A. F.; FONSECA, R.; MENDES, M. A. A methodology for workload characterization of e-commerce sites. In: **Proceedings of the 1st ACM Conference on Electronic Commerce**. New York, NY, USA: ACM, 1999. (EC '99), p. 119–128. ISBN 1-58113-176-3. Disponível em: <http://doi.acm.org/10.1145/336992.337024>. Citado na página 109.

MüLLER, H.; KIENLE, H.; STEGE, U. Autonomic Computing Now You See It, Now You Don't: Design and Evolution of Autonomic Software Systems. In: DE-LUCIA, A.; FERRUCCI, F. (Ed.). **Software Engineering**. Salerno, Italy: Springer-Verlag Berlin Heidelberg, 2009. v. 5413/2009, p. 32–54. ISBN 978-3-540-95887-1. Disponível em: <http://dx.doi.org/10.1007/978-3-540-95888-8_2>. Citado na página 47.

NELSON, B.; TAAFFE, M. The pht/pht/∞ queueing system: Part i—the single node. **INFORMS Journal on Computing**, v. 16, n. 3, p. 266–274, 2004. Disponível em: <http://dx.doi.org/10.1287/ijoc.1040.0070>. Citado na página 41.

____. The $[\text{pht/pht/}\infty]^k$ queueing system: Part ii—the multiclass network. **INFORMS Journal on Computing**, v. 16, n. 3, p. 275–283, 2004. Disponível em: <http://dx.doi.org/10.1287/ijoc.1040.0071>. Citado na página 41.

NOBILE, P. N. **Projeto de um broker de gerenciamento adaptativo de recursos em computação em nuvem baseado em técnicas de controle realimentado**. Tese (Doutorado) — ICMC-USP, 2013. Citado 5 vezes nas páginas 48, 83, 86, 99 e 110.

PADALA, P.; SHIN, K. G.; ZHU, X.; UYSAL, M.; WANG, Z.; SINGHAL, S.; MERCHANT, A.; SALEM, K. Adaptive control of virtualized resources in utility computing environments. **ACM SIGOPS Operating Systems Review**, v. 41, n. 3, p. 289, jun. 2007. ISSN 01635980. Disponível em: <http://portal.acm.org/citation.cfm?doid=1272998.1273026>. Citado na página 48.

PAPADOPOULOS, A. V.; MAGGIO, M.; TERRANEO, F.; LEVA, A. A dynamic modelling framework for control-based computing system design. **Mathematical and Computer Modelling of Dynamical Systems**, v. 21, n. 3, p. 251–271, 2015. Disponível em: <http://dx.doi.org/10.1080/13873954.2014.942785>. Citado na página 49.

PAREKH, S.; GANDHI, N.; HELLERSTEIN, J. Using control theory to achieve service level objectives in performance management. **Real-Time Systems**, v. 23, p. 127–141, 2002. Citado na página 94.

PEREIRA JR., L. A.; MAMANI, E. L. C.; SANTANA, M. J.; SANTANA, R. H. C.; MONACO, F. J.; NOBILE, P. N. Extending discrete-event simulation frameworks for non-stationary performance evaluation: Requirements and case study. In: **Proceedings of the 2015 Winter Simulation Conference**. Piscataway, NJ, USA: IEEE Press, 2015. (WSC '15), p. 3150–3151. ISBN 978-1-4673-9741-4. Citado 2 vezes nas páginas 75 e 92.

PEREIRA, L. A.; MAMANI, E. L. C.; SANTANA, M. J.; SANTANA, R. H. C.; NOBILE, P. N.; MONACO, F. J. Non-stationary simulation of computer systems and dynamic performance evaluation: A concern-based approach and case study on cloud computing. In: **Computer Architecture and High Performance Computing (SBAC-PAD), 2015 27th International Symposium on**. [S.l.: s.n.], 2015. p. 130–137. ISSN 1550-6533. Citado 2 vezes nas páginas 75 e 92.

REISS, C.; WILKES, J.; HELLERSTEIN, J. L. **Google cluster-usage traces: format + schema**. Mountain View, CA, USA, 2011. Revised 2012.03.20. Posted at <http://code.google.com/p/googleclusterdata/wiki/TraceVersion2>. Citado 2 vezes nas páginas 144 e 145.

RIANO, G. **Transient Behavior of Stochastic Networks: Application to Production Planning with Load-Dependent Lead Times**. Tese (Doutorado) — Georgia Institute of Technology, 2002. Citado na página 43.

ROTHKOPF, M.; OREN, S. A closure approximation for the nonstationary m/m/s queue. **Management Science**, v. 25, n. 6, p. 522–534, 1979. Disponível em: <http://dx.doi.org/10.1287/mnsc.25.6.522>. Citado na página 41.

SHA, L.; ABDELZAHER, T.; ARZéN, K.-E.; CERVIN, A.; BAKER, T.; BURNS, A.; BUTTAZZO, G.; CACCAMO, M.; LEHOCZKY, J.; MOK, A. K. Real Time Scheduling Theory: A Historical Perspective. **Real-Time Syst.**, v. 28, n. 2-3, p. 101–155, 2004. Citado na página 48.

SOUZA, A. C. Z.; PINHEIRO, C. A. M. **Introdução à Modelagem, Análise e Simulação de Sistemas Dinâmicos**. 1. ed. Rio de Janeiro: Ed. Interciencia, 2008. ISBN 978-85-7193-188-6. Citado na página 58.

SOUZA, F. **Extensão da geração de carga do Bench4Q para benchmark de desempenho em regime transiente**. Dissertação (Mestrado) — ICMC/USP, 2016. Citado na página 110.

STANKOVIC, J. A.; ABDELZAHER, T. F.; MARLEY, M.; TAO, G.; SON, S. H.; LU, C. Feedback control scheduling in distributed real-time systems. In: **Proceedings 22nd IEEE Real-Time Systems Symposium (RTSS 2001) (Cat. No.01PR1420)**. IEEE Comput. Soc, 2001. p. 59–70. ISBN 0-7695-1420-0. Disponível em: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=990596>. Citado na página 52.

STANKOVIC, J. A.; SON, S. H.; LU, C.; TAO, G. The case for feedback control real-time scheduling. In: **Proceedings of 11th Euromicro Conference on Real-Time Systems. Euromicro RTS'99**. IEEE Comput. Soc, 1999. p. 11–20. ISBN 0-7695-0240-7. Disponível em: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=777445>. Citado 3 vezes nas páginas 15, 51 e 52.

TAAFFE, M. R.; ONG, K. L. Approximating nonstationaryph(t)/m(t)/s/c queueing systems. **Annals of Operations Research**, v. 8, n. 1, p. 103–116, 1987. ISSN 1572-9338. Disponível em: <http://dx.doi.org/10.1007/BF02187085>. Citado na página 42.

URGAONKAR, B.; SHENOY, P.; CHANDRA, A.; GOYAL, P. Dynamic Provisioning of Multi-tier Internet Applications. In: **Second International Conference on Autonomic Computing (ICAC'05)**. IEEE, 2005. p. 217–228. ISBN 0-7965-2276-9. Disponível em: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1498066>. Citado 2 vezes nas páginas 48 e 53.

WANG, X.; CHEN, M. Cluster-level feedback power control for performance optimization. In: **High Performance Computer Architecture, 2008. HPCA 2008. IEEE 14th International Symposium on**. [S.l.: s.n.], 2008. p. 101–110. ISSN 1530-0897. Citado na página 48.

WANG, Y.; WANG, X.; CHEN, M.; ZHU, X. Partic: Power-aware response time control for virtualized web servers. **Parallel and Distributed Systems, IEEE Transactions on**, IEEE, v. 22, n. 2, p. 323–336, 2011. Citado na página 48.

WOLSKI, R.; PLANK, J. S.; BREVIK, J.; BRYAN, T. Analyzing Market-Based Resource Allocation Strategies for the Computational Grid. **International Journal of High Performance Computing Applications**, v. 15, p. 258–281, 2001. Citado na página 54.

XIA, F.; SUN, Y.; TIAN, Y. C. Feedback scheduling of priority-driven control networks. **Computer Standards & Interfaces**, Elsevier, v. 31, n. 3, p. 539–547, 2009. Citado na página 48.

XIONG, P.; WANG, Z.; JUNG, G.; PU, C. Study on performance management and application behavior in virtualized environment. In: **2010 IEEE Network Operations and Management Symposium - NOMS 2010**. IEEE, 2010. p. 841–844. ISBN 978-1-4244-5366-5. Disponível em: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5488362>. Citado na página 48.

YANG, F.; LIU, J. Transient analysis of general queueing systems via simulation-based transfer function modeling. In: **Proceedings of the 2010 Winter Simulation Conference, WSC 2010, Baltimore, Maryland, USA, 5-8 December 2010**. [s.n.], 2010. p. 1110–1122. Disponível em: <http://dx.doi.org/10.1109/WSC.2010.5679079>. Citado 2 vezes nas páginas 41 e 76.

____. Simulation-based transfer function modeling for transient analysis of general queueing systems. **European Journal of Operational Research**, v. 223, n. 1, p. 150–166, 2012. Disponível em: <http://dx.doi.org/10.1016/j.ejor.2012.05.040>. Citado 2 vezes nas páginas 41 e 46.

ZHANG, W.; WANG, S.; WANG, W.; ZHONG, H. Bench4q: A qos-oriented e-commerce benchmark. In: **Computer Software and Applications Conference (COMPSAC), 2011 IEEE 35th Annual**. [S.l.: s.n.], 2011. p. 38–47. ISSN 0730-3157. Citado 2 vezes nas páginas 39 e 108.

ZHU, X.; UYSAL, M.; WANG, Z.; SINGHAL, S.; MERCHANT, A.; PADALA, P.; SHIN, K. What does control theory bring to systems research? **ACM SIGOPS Operating Systems Review**, v. 43, n. 1, p. 62, jan. 2009. ISSN 01635980. Disponível em: <http://portal.acm.org/citation.cfm?doid=1496909.1496922>. Citado 2 vezes nas páginas 48 e 58.